Università degli studi di Pisa

Corso di Laurea in Ingegneria Informatica

Indirizzo “Sistemi ed applicazioni informatici”

*Tesi di laurea*

# “Progettazione e realizzazione di un sistema per firma digitale e autenticazione basato su smartcard”

*Candidato:* Tommaso Cucinotta

*Relatori:* Prof. Paolo Ancilotti

Ing. Giuseppe Anastasi

Dr. Paolo Bizzarri

Anno Accademico 1998/1999

# Indice

# Introduzione

È indubbio che l'evoluzione delle tecnologie informatiche sta accelerando l'affermazione della cosiddetta *società dell'informazione*, in cui la globalizzazione dei servizi informativi caratterizza in modo sempre più incisivo la vita pubblica e privata di ogni individuo. Tuttavia il fatto che chiunque, da qualunque parte del mondo, abbia (o avrà) la possibilità di accedere ad unico servizio informativo globale ha anche una controparte negativa, come ha ormai dimostrato l'evoluzione dell'Internet, in cui sempre più spesso si sente parlare di "shadowing" di siti web, di "spoofing" delle informazioni e di "crimini informatici" in genere.

Infatti il potenziamento dei canali di accesso ad un sistema facilita l'ingresso anche a soggetti che, illegalmente, cercano di carpire o addirittura falsificare informazioni riservate.

Si pensi, ad esempio, all'infinità di *form* quotidianamente riempiti ed inviati sull'Internet contenenti nomi, cognomi, indirizzi di e-mail, indirizzi fisici, aree d'interesse personale, o, peggio ancora, numeri di carte di credito, password di accesso a sistemi remoti, ecc… In

assenza di opportuni strumenti di protezione, utenti non autorizzati potrebbero venire in possesso di tali informazioni, utilizzandole a proprio vantaggio e/o arrecando danni morali o economici a persone e aziende.

La *crittografia simmetrica* è già da tempo utilizzata nell'ambito delle reti di calcolatori per la realizzazione di canali di comunicazione privati. Comunque questo schema, basato sull'esistenza di una *chiave segreta unica* nota soltanto alle parti comunicanti, è stato ormai rivoluzionato dalla scoperta degli *algoritmi crittografici asimmetrici*, in cui la *chiave* di codifica e quella di decodifica rimangono nettamente distinte, nel senso che il possesso dell'una non implica il possesso anche dell'altra. In questo modo è possibile mantenere privata una delle due chiavi[1], mentre si può rivelare pubblicamente l'altra[2], permettendo di fatto un'interazione reciproca affidabile.

Combinando algoritmi a chiave pubblica e a chiave simmetrica è possibile ottenere complessi protocolli crittografici che, opportunamente strutturati all'interno di un' *Infrastruttura a Chiave Pubblica (PKI[3]),* hanno permesso la nascita di sofisticati *servizi per la sicurezza* dei dati, come la *firma digitale*, le connessioni sicure via

[1] Da cui il nome di *chiave privata*.
[2] Da cui il nome di *chiave pubblica*.
[3] "Public Key Infrastructure".

Internet[4] (SSL), ecc…

Le PKI si stanno affermando, oggi, come la soluzione ottimale che permette agli utenti di un sistema informativo globale di fidarsi di esso, consentendo sia il reperimento di documenti *autentici,* sia la trasmissione di informazioni *riservate*. L'utilizzo di tali infrastrutture, oggi, può dare il via libera alle applicazioni telematiche del nuovo millennio, come il commercio elettronico su scala mondiale, il telelavoro, l'informatizzazione (e quindi semplificazione) delle procedure burocratiche e così via.

Comunque all'interno di una PKI la sicurezza dei dati del singolo utente è garantita fintantoché le sue chiavi private rimangono *segrete*. Questo è un requisito di fondamentale importanza per il funzionamento dell'intera infrastruttura, poiché la conoscenza delle chiavi private è tutto ciò di cui un "*hacker*" ha bisogno per decifrare messaggi cifrati o firmare documenti per conto del proprietario delle chiavi stesse.

Alla luce dei risvolti che il nostro sistema legale sta attraversando nell'area della legalizzazione dei documenti informatici e delle firme elettroniche, è evidente come, in un prossimo futuro, il furto o l'utilizzo non autorizzato di una chiave privata potrà avere conseguenze disastrose per il legittimo proprietario.

Lo sviluppo tecnologico, oggi, ha portato alla nascita delle

[4] Si pensi ad esempio al protocollo Secure Socket Layer.

*smartcard*[5]*,* dispositivi progettati per garantire la sicurezza delle chiavi private dei singoli utenti. Una smartcard è costituita da una sorta di microcalcolatore inserito in una scheda delle dimensioni di una carta di credito, dotato di *un'intelligenza*[6] sufficiente a garantire la sicurezza delle informazioni in essa custodite. Grazie al suo utilizzo un utente non solo ha la certezza di mantenere segrete le proprie chiavi private, ma allo stesso tempo ha la possibilità di utilizzarle con la massima comodità, essendo in grado di portarle sempre con sé.

È per questo che ho ritenuto opportuno, all'interno del vastissimo campo della sicurezza dei dati, concentrare l'attenzione del mio lavoro di tesi proprio su questa nuova tecnologia, ed in particolare sulla sua integrazione con le *PKI*.

*La mia tesi sulla sicurezza*

Dopo aver passato in rassegna diverse infrastrutture a chiave pubblica, alcune esistenti soltanto a livello di proposta, altre già operative in svariati settori per lo più di carattere commerciale, ho cercato di definire quale debba essere l'architettura di riferimento di un sistema software che sia di supporto per lo sviluppo di una PKI in grado di fornire i servizi di firma digitale e autenticazione basandosi

[5] Traduzione letterale: "scheda intelligente". Si veda la nota successiva.
[6] Il termine "intelligenza" si riferisce al fatto che una smartcard è un dispositivo dinamico, attivo, in grado di eseguire delle elaborazioni, contrariamente alle classiche schede magnetiche che hanno solamente funzionalità di memoria.

sulla tecnologia delle smartcard.

Per la realizzazione del progetto ho voluto seguire l'approccio del cosiddetto software *"Open Source*". Tale approccio mette in primo piano il bisogno di *"libertà"* nella produzione, sviluppo ed utilizzo del software, contrapponendosi fortemente al classico schema "*commerciale*" che prevede la segretezza delle tecnologie allo scopo di riservarsi potenziali profitti. Il vantaggio di un approccio "libero" consiste soprattutto nella possibilità di avere un continuo contatto e confronto con gli altri. Nel campo della sicurezza dei dati questo confronto è più che mai necessario, poiché permette di individuare molto più rapidamente eventuali "*debolezze*" di un sistema software. Questa filosofia del produrre software è ormai un fenomeno sempre crescente, oggi, e sta portando alla nascita di strumenti sempre più complessi che addirittura arrivano a concorrere con le applicazioni commerciali del settore, come sta ormai dimostrando la diffusione sempre più ampia di sistemi operativi "aperti" simili allo *Unix*.

Nell'ambito del software *"Open Source"* ho appunto trovato diversi strumenti per lo sviluppo di applicazioni sicure, ma per quanto riguarda l'integrazione fra infrastrutture a chiave pubblica e smartcard non c'era ancora uno strumento funzionante che permettesse la realizzazione dei servizi di sicurezza su enunciati. Ho cercato di realizzare questo strumento, peraltro estremamente complesso, avvalendomi dei vari progetti a disposizione all'interno della

"*comunità*" Open Source.

Il progetto *OpenCA*[7], ancora in piena fase di sviluppo, si propone di realizzare un'infrastruttura a chiave pubblica basata su di un'organizzazione gerarchica ad albero di autorità di certificazione, per l'emissione di certificati di chiave pubblica utilizzabili per i servizi di firma digitale, autenticazione e riservatezza.

Il progetto MUSCLE[8], invece, si propone di promuovere l'utilizzo delle smartcard sul sistema operativo Linux, mettendo a disposizione degli utenti i driver di basso livello per i diversi modelli di smartcard e lettori di smartcard presenti oggi sul mercato.

Infine, il progetto OpenSSL[9] mette a disposizione una libreria crittografica di base che, oltre ad implementare in software tutti gli algoritmi crittografici previsti dai vari standard sui certificati elettronici, fornisce tutti gli strumenti necessari per lavorare con i certificati stessi.

Una volta analizzata la struttura del software di OpenCA, ho proceduto con l'aggiunta del supporto per smartcard, utilizzando MUSCLE come punto di partenza per la comunicazione con la scheda. Inizialmente è stato necessario realizzare un driver di basso livello, specifico per la smartcard a mia disposizione, una "*Cyberflex*

[7] Open Certification Authority, home page all'indirizzo "http://www.openca.org".
[8] Progetto MUSCLE: "Movement for the Use of Smart Cards in a Linux Environment" (http://www.linuxnet.com).
[9] Progetto OpenSSL, home page all'indirizzo http://www.openssl.org.

*Access 16K*" della Schlumberger, che ha costituito la base per i moduli successivi.

Ho quindi sviluppato un *modulo PKCS-11* da installare all'interno di Netscape Communicator, che permette l'integrazione della "smartcard technology" con OpenCA, nonché l'utilizzo diretto dei certificati rilasciati dall'autorità di certificazione drettamente all'interno del browser per l'apposizione della firma digitale ai messaggi di e-mail e di news. Per dare all'utente anche la possibilità di firmare documenti (o file) qualsiasi, ho messo a punto una coppia di *programmi* che permettono di calcolare una firma digitale, e, successivamente, di verificarne la correttezza.

Inoltre, per la realizzazione di un *servizio di autenticazione* basato su smartcard, ho sviluppato un *modulo P.A.M.*[10] che, una volta installato, permette ad un amministratore di sistema di sostituire o integrare lo schema di autenticazione presente con quello basato sulle smartcard e sui certificati rilasciati da OpenCA.

Per quanto riguarda la gestione della smartcard, ho realizzato una shell interattiva a riga di comando che, oltre a supportare tutti i comandi previsti dallo standard *ISO 7816-4*, è anche in grado di impartire alla scheda i vari comandi necessari per un controllo

[10] Pluggable Authentication Module: è un componente modulare che, all'interno dei sistemi operativi Unix-Like, permette di verificare l'identità di un utente. Il modulo, opportunamente installato, risulta del tutto trasparente alle applicazioni, che continuano a funzionare perfettamente con il nuovo schema di autenticazione senza bisogno di alcuna modifica.

completo delle chiavi eventualmente presenti sulla scheda, nonché degli attributi di sicurezza dei vari file[11].

*Organizzazione della tesi*

Nel primo capitolo ho trattato in generale le problematiche relative alla *sicurezza* dei dati in transito in una rete di calcolatori "*aperta*". Ho cercato di evidenziare la necessità, per gli utenti, di avere a disposizione opportuni strumenti di protezione, mentre ho introdotto la crittografia come punto di partenza per la realizzazione di tali strumenti.

Nel secondo capitolo ho esposto in dettaglio i concetti fondamentali di algoritmo simmetrico, asimmetrico e di impronta digitale, cercando anche di evidenziare quali sono le possibili vie a disposizione della crittoanalisi per compromettere la sicurezza di dati protetti crittograficamente.

Nel terzo capitolo ho descritto i princìpi, per lo più matematici, di funzionamento dei più usati algoritmi crittografici, mentre nel quarto capitolo ho esposto *come* questi algoritmi vengono combinati fra loro in opportuni *protocolli crittografici*.

Nel quinto capitolo ho introdotto le infrastrutture a chiave pubblica e ho cercato di chiarire in che modo queste possano venire in aiuto

[11] Gli standard del gruppo ISO 7816 coprono tutte le problematiche relative alle smartcard. Il documento 7816-4 specifica essenzialmente un set di comandi standard per la

nella risoluzione del problema, di importanza centrale, della *distribuzione di chiavi* pubbliche.

Ho ritenuto opportuno esaminare, nel sesto capitolo, lo stato dell'arte della legislazione italiana per quanto riguarda l'accettazione della firma digitale da un punto di vista legale. In particolare ho riportato, fra l'altro, alcuni estratti della legge Bassanini, di fondamentale importanza per capire in che modo l'Italia abbia posto le basi per un'accettazione dei documenti digitali firmati elettronicamente al pari dei rispettivi cartacei firmati "a mano".

Nel settimo capitolo ho descritto come la tecnologia delle smartcard possa risolvere diversi problemi relativi alla gestione delle chiavi private degli utenti.

Nell'ottavo capitolo ho ampiamente descritto il progetto da me realizzato, evidenziandone sia l'architettura generale che le principali scelte effettuate a livello di implementazione. In particolare, con l'ausilio di diverse figure, ho esposto chiaramente le relazioni esistenti fra le varie componenti software e l'ambiente in cui sono state inserite, evidenziando in che modo ho utilizzato le librerie messe a disposizione dai diversi progetti "*Open Source*" sopra citati.

---

navigazione nel filesystem di una smartcard e per un'autenticazione primitiva dell'utente basata su PIN.

# Capitolo 1. Il problema della sicurezza nelle reti di calcolatori

## 1.1 Introduzione al problema

La continua evoluzione, diversificazione e moltiplicazione dei servizi e dei mezzi di comunicazione aggiunge sempre nuove minacce alla sicurezza delle informazioni che vi transitano. È per questo che sono da sempre esistiti opportuni meccanismi di protezione per informazioni critiche destinate ad attraversare mezzi di comunicazione insicuri. Pensiamo ai tempi dell'Impero Romano, in cui i corrieri portavano messaggi cifrati di cui essi stessi ignoravano il significato[1]. Pensiamo, anche, agli svariati tentativi di codifica dei dati adottati in passato nel campo della trasmissione digitale, basati su algoritmi più o meno complessi per la trasformazione di stringhe di bit in altre apparentemente incomprensibili.

[1] Si racconta che Cesare abbia utilizzato un meccanismo di codifica basato sulla permutazione di lettere.

Quindi da sempre i mezzi di comunicazione sono affiancati da opportuni strumenti di sicurezza. Per esempio, volendo spedire un messaggio ad un destinatario utilizzando il sistema postale tradizionale, sarebbero a disposizione diverse modalità per farlo, ognuna caratterizzata da un diverso grado di sicurezza:

- *cartolina postale*: non garantisce alcuna riservatezza, infatti chiunque è in grado di leggere il messaggio; garantisce, comunque, un livello minimo di protezione da modifiche[2] del messaggio, per la difficoltà di imitazione della grafia o della firma del mittente, almeno in assenza di strumenti e/o competenze specifiche
- *lettera in busta chiusa*: l'integrità dell'involucro, per il destinatario, garantisce che la lettera non è stata letta né modificata da altri; comunque un individuo in possesso di strumenti e/o competenze specifiche potrebbe essere in grado di leggere il contenuto della busta, mentre la grafia e la firma del mittente sulla lettera rendono più difficili modifiche o sostituzione del contenuto
- *lettera raccomandata*: il destinatario, al momento della ricezione, firma un documento che ritorna al mittente, dandogli possibilità di dimostrare l'avvenuta ricezione; in questo caso è

la firma che rende difficoltosi eventuali tentativi di falsificazione

- *lettera certificata da un notaio*: la firma e gli eventuali timbri apposti dal notaio garantiscono, agli occhi del destinatario, che la lettera è stata effettivamente scritta dall'individuo che figura come mittente; la garanzia di autenticità deriva dal fatto che imitare timbro, sigillo e firma apposti dal notaio è molto più difficile che imitare semplicemente la firma di un individuo.

Mettendo da parte i sistemi tradizionali di comunicazione, ed andando ad esaminare, invece, i nuovi strumenti a disposizione dell'uomo oggi, si noterà come la situazione non è molto diversa: nell'ambito di una rete di calcolatori, fin quando le informazioni da trasportare sono di scarso valore, ci si accontenta dei meccanismi più "primitivi", privi di alcun sistema di sicurezza. Ma non appena si considerano messaggi di un qualche valore, sia esso di natura economica, morale, politica, personale, ecc…, si rende necessaria l'adozione di opportuni strumenti di protezione.

## 1.2 Necessità dei servizi per la sicurezza di dati nelle reti aperte

Prima di passare ad esaminare gli strumenti a disposizione, oggi, per la protezione dei dati in una rete di calcolatori, introdurrò brevemente le problematiche di sicurezza relative a questa tipologia

[2] Si intende che una terza parte possa voler modificare il contenuto di un messaggio con l'intento di attribuirlo comunque al mittente. Le modifiche sono sempre possibili, solo che

di trasporto delle informazioni, con un semplice esempio.

Si consideri (Figura 1.1) un'entità $E_1$, su di un calcolatore $C_1$, che trasmette un messaggio $M$ ad un'entità $E_2$, su di un calcolatore $C_2$, connesso a $C_1$ mediante una rete $R$. Le due entità utilizzano primitive messe a disposizione da servizi di rete di livello trasporto, come ad esempio il TP4 dello stack OSI, o il TCP. Tali primitive garantiscono, a meno di malfunzionamenti, che il messaggio $M$ arrivi a destinazione senza alterazioni, percorrendo uno dei possibili cammini che nella rete $R$ porta da $C_1$ a $C_2$.

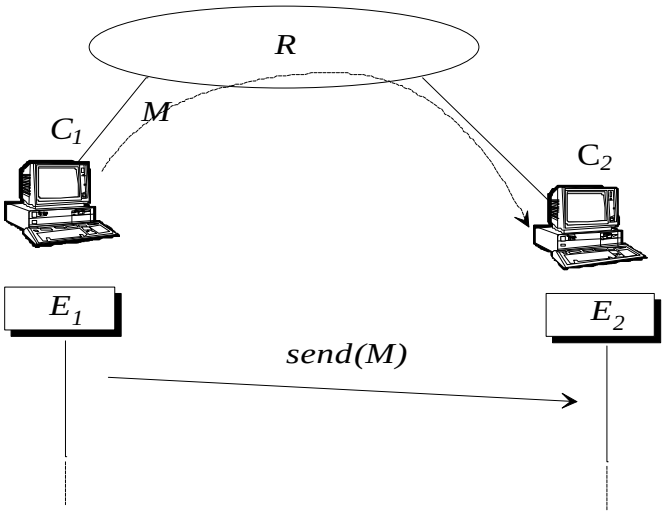


*Figura 1.1. Trasmissione del messaggio M da $C_1$ a $C_2$ attraverso la rete di calcolatori R: "`send(`ı`)`" rappresenta una generica primitiva di rete per la trasmissione di un pacchetto.*

Tale affermazione è valida nell'ipotesi che i calcolatori intermedi

se il destinatario le rileva esse sono perfettamente inutili.

su cui $M$ transita si attengano ai protocolli previsti (nell'esempio, lo stack TCP/IP). Per quanto concerne il problema della sicurezza e riservatezza dei dati, dobbiamo osservare che in tali protocolli non vi è alcuna verifica della validità di tale ipotesi. È possibile, quindi, che un'implementazione opportunamente modificata di tali protocolli su di una o più macchine connesse alla rete si comporti in modo non previsto dai protocolli stessi, ma compatibilmente con essi.

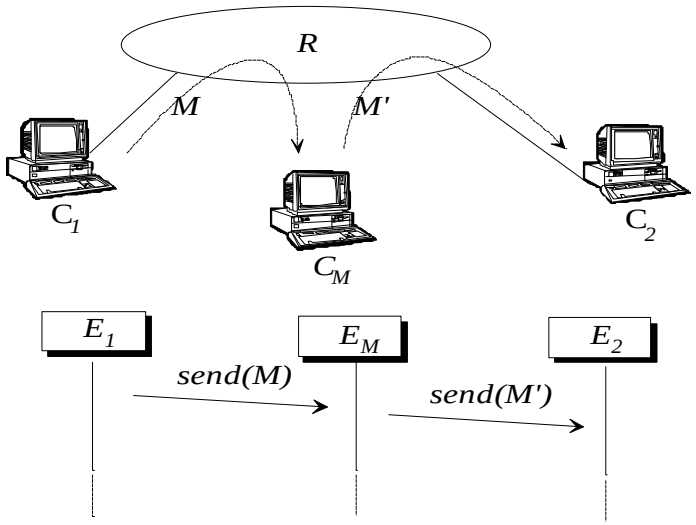


*Figura 1.2. L'entità "maliziosa" $E_M$, in azione sul calcolatore $C_M$, si interpone nella comunicazione tra $E_1$ ed $E_2$.*

In particolare, con riferimento ad un'entità $E_M$ su di un calcolatore intermedio $C_M$ (Figura 1.2) che utilizzi servizi di rete opportunamente modificati, nell'ipotesi che $E_1$ non aspetti risposta da parte del destinatario, sono possibili i seguenti scenari, illustrati in Figura 1.3:

- $E_M$, oltre ad implementare correttamente i protocolli di rete,

registra tutto o parte del traffico che lo attraversa, ed in particolare registra il messaggio $M$, rendendolo disponibile ad altre entità: $E_1$ ed $E_2$ non si accorgono che le informazioni scambiate sono state ricevute da altri

- $E_M$ modifica il contenuto di $M$ prima di inoltrarlo al nodo successivo nel cammino verso $E_2$: $E_1$ crede di aver spedito correttamente il messaggio ad $E_2$, mentre $E_2$ crede che il messaggio ricevuto $M$' sia effettivamente quello spedito da $E_1$ e non si accorge delle modifiche apportate da $E_M$

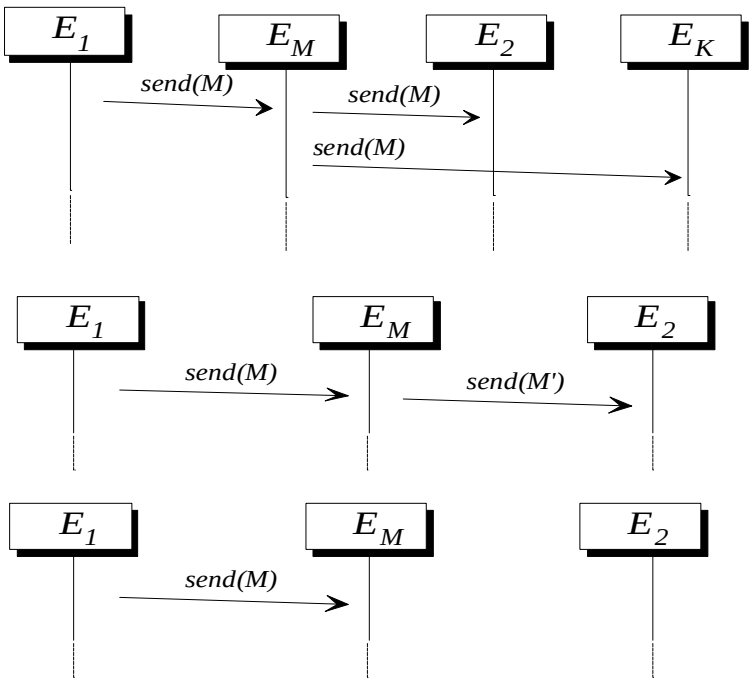


*Figura 1.3 - Possibilità di attacco da parte di $E_M$.*

- $E_M$ simula la ricezione di $M$ da parte di $E_2$, e non lo inoltra

affatto verso $C_2$: $E_1$ crede di aver spedito correttamente $M$ ad $E_2$, mentre $E_2$ non riceve nulla.

Nell'ipotesi in cui $E_1$ si aspetti una risposta $R$ da parte di $E_2$, sono invece possibili i seguenti scenari (Figura 1.4):

- $E_M$ intercetta e sopprime il messaggio $M$ da $E_1$ ad $E_2$, non simulando alcuna risposta: l'entità $E_1$, prima o poi, si accorge della mancata ricezione di $M$ da parte di $E_2$, e si comporta di conseguenza (probabilmente rileva un malfunzionamento di rete)
- $E_M$ si sostituisce a $E_2$ nel protocollo, senza inoltrare messaggi verso $E_2$, allo scopo di ottenere da $E_1$ informazioni destinate ad $E_2$: l'entità $E_1$ pensa di comunicare con $E_2$, e non si accorge di nulla
- $E_M$ si interpone nella comunicazione tra $E_1$ ed $E_2$, intercettando e/o modificando opportunamente i messaggi in entrambe le direzioni: sia $E_1$ che $E_2$ credono di comunicare tra loro e non si accorgono di nulla.

In una tale situazione la presenza dell'entità $E_M$ può nuocere o meno alle entità $E_1$ ed $E_2$ a seconda della natura delle informazioni che esse si scambiano attraverso la rete. Se, ad esempio, il

messaggio $M$ contiene informazioni di pubblico dominio, il fatto che una terza entità venga a conoscenza del suo contenuto durante la trasmissione non nuoce a nessuno.

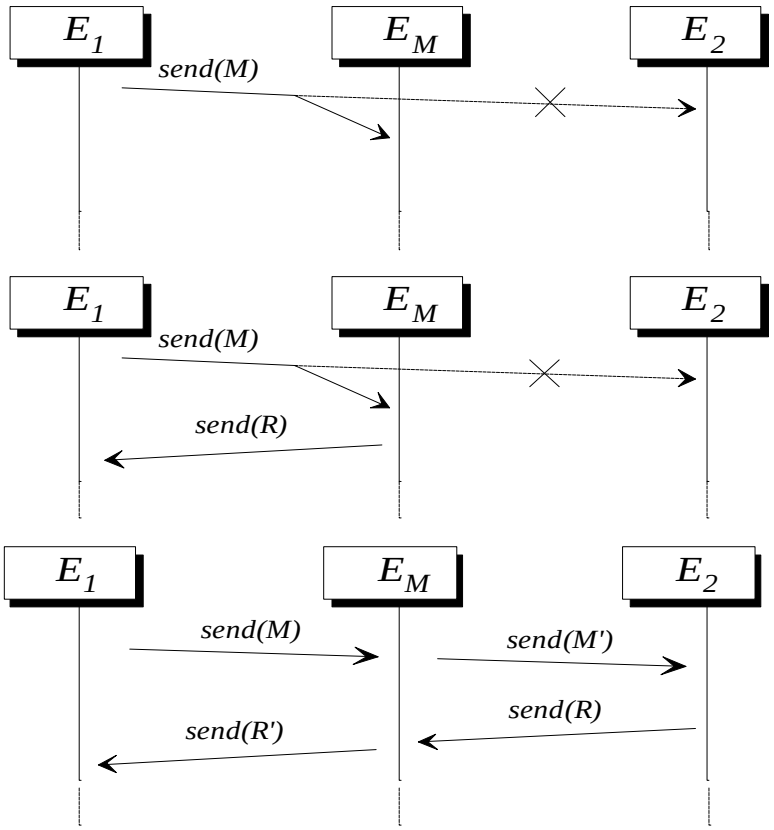


*Figura 1.4 - Possibilità di attacco da parte di $E_M$*

Se, invece, esso contiene dati personali riservati, come ad esempio quelli tipicamente richiesti da un form di una pagina HTML (nome, cognome, età, sesso, stato civile, professione, abitazione, residenza, numero di telefono, indirizzo di e-mail, numero di carta di credito), allora l'entità $E_M$ può, con un utilizzo non autorizzato di tali dati, arrecare danni a persone. Se, inoltre, $E_M$ è in grado di

modificare i messaggi che transitano tra entità remote, la situazione è ancora più grave: $E_M$ può impersonificare una o più entità e compiere attività a loro nome.

L'esempio illustrato evidenzia il bisogno di adottare precauzioni per lo scambio sicuro di informazioni su di una rete, ove il termine “sicuro” può significare diverse cose, a seconda della natura dei dati che transitano e dei requisiti delle applicazioni che comunicano.

La disciplina di base che permette la realizzazione di servizi per la sicurezza è la **crittografia**. Come verrà spiegato nel seguito, essa fornisce strumenti adeguati per affrontare e risolvere il problema della sicurezza delle informazioni in una rete aperta.

## 1.3 La soluzione: i servizi per la sicurezza dei dati

Sarebbe opportuno precisare, a questo punto, il significato del termine “sicurezza”, nel contesto delle reti telematiche. Relativamente all’esempio del paragrafo precedente, gli utenti in questione, oggi, potrebbero richiedere che il loro network-provider fornisca uno od una combinazione dei seguenti *servizi di sicurezza*:

- **autenticazione delle parti**, cioè la possibilità di verificare l'identità di due entità remote prima dell’apertura di una sessione di comunicazione
- **riservatezza dei dati**, cioè il consolidamento di un canale di

comunicazione che assicuri che i dati in transito su di esso non possano cadere nelle mani di entità non autorizzate

- **integrità dei dati**, cioè la possibilità di verificare che i dati ricevuti non sono stati alterati durante il transito sulla rete
- **autenticità dei dati**, cioè la possibilità di verificare che i dati ricevuti sono stati effettivamente spediti dall'entità che risulta come mittente e non hanno subito alcuna modifica o aggiunta dopo la loro spedizione
- **marcatura temporale**, cioè la certificazione, da parte del provider, dell'avvenuta trasmissione di un messaggio, con indicazione della data e dell'ora, in modo che il mittente, successivamente, possa dimostrare ad altri di aver effettuato la trasmissione

Inoltre, per proteggere il traffico di dati relativo a determinate categorie di applicativi, potrebbe essere necessaria, da parte del provider, l'attivazione di ulteriori servizi per la sicurezza, come ad esempio:

- **non ripudio dell'origine**, cioè la possibilità, da parte del destinatario di un messaggio, di dimostrare a terzi che il messaggio è stato effettivamente spedito dal mittente[3]
- **non ripudio di ricezione**, cioè la possibilità, successivamente

alla ricezione di un messaggio da parte di un'entità, di dimostrare a terzi (o al mittente) l'avvenuta ricezione[4].

Risulta evidente che nessuno dei protocolli tradizionali di rete riesce a garantire i servizi appena enunciati. Alcuni di tali servizi, come ad esempio la riservatezza, possono essere forniti ricorrendo a *tecniche di codifica* delle informazioni, in cui la segretezza degli algoritmi di codifica e decodifica garantisce che nessuno possa leggere il contenuto dei messaggi.

Oggi, invece, si può ricorrere alle moderne e più sofisticate **tecniche di crittografia**, che, come vedremo successivamente, presentano innumerevoli vantaggi rispetto agli approcci del passato.

## 1.4 Servizi, protocolli e primitive crittografiche

Ritengo opportuno puntualizzare, fin da questa sezione introduttiva, l'utilizzo che verrà fatto durante tutto il lavoro di alcuni termini fondamentali.

Con il termine **servizi crittografici** mi riferirò a tutti quei servizi per la sicurezza che possono essere espletati ricorrendo alla crittografia. Tali servizi, in quanto rientranti nella categoria dei servizi per la sicurezza, permettono il transito di dati su di una rete aperta prevenendo o rendendo individuabile qualsiasi loro utilizzo illecito o

[3] La dizione "*non ripudio*" si riferisce all'impossibilità, da parte del mittente del messaggio, di negare di averlo spedito.

comunque non autorizzato.

Affinché delle entità di rete possano usufruire di un servizio crittografico per la protezione dei dati, è necessario che rispettino ben precisi **protocolli** nella spedizione e ricezione dei vari messaggi.

Un protocollo potrebbe essere definito come[5] "una serie di passi che due o più parti devono eseguire in turno secondo una sequenza predeterminata, progettato per realizzare un compito". Un **protocollo crittografico** è un protocollo che, per essere realizzato, richiede che le entità partecipanti facciano uso di algoritmi crittografici.

Tipicamente i vari algoritmi crittografici sono implementati in uno strato software costituito da tutta una serie di **primitive crittografiche**.

In definitiva, è mediante la chiamata a varie *primitive crittografiche* che un'applicazione manda avanti passo-passo l'esecuzione di un *protocollo crittografico* allo scopo di proteggere i propri dati usufruendo di un qualche *servizio crittografico*.

---

[4] La dizione "*non ripudio*" si riferisce, qui, all'impossibilità, da parte dell'entità che riceve il messaggio, di negare di averlo ricevuto.

[5] Bruce Schneier, *"Applied Cryptography Second Edition : protocols, algorithms, and source code in C*", John Wiley & Sons, Inc.

# Capitolo 2. Fondamenti di crittografia

## 2.1 Cos'è la crittografia

La **crittografia** è "*la disciplina che studia la trasformazione di dati allo scopo di nascondere il loro contenuto semantico, impedire il loro utilizzo non autorizzato, o impedire qualsiasi loro modifica non rilevabile*"[6].

Le tecniche crittografiche più largamente conosciute sono quelle che permettono di ottenere *riservatezza* in una trasmissione, codificando messaggi in modo che soltanto i legittimi destinatari siano in grado di decodificarli. Tali tecniche forniscono algoritmi per cifrare una stringa di bit, detta anche **testo in chiaro**, ottenendone una seconda, detta anche **testo**[7] **in cifra**, in modo che soltanto gli utenti autorizzati siano in grado di decifrarla, grazie al possesso di opportune informazioni.

[6] Bruce Schneier, "*Applied Cryptography Second Edition : protocols, algorithms, and source code in C*", John Wiley & Sons, Inc.

La **crittoanalisi** è la disciplina che studia come recuperare tutto o parte del contenuto di un messaggio cifrato, *senza* avere a disposizione le informazioni in possesso degli utenti autorizzati a decifrarlo. La branca della matematica che racchiude crittografia e crittoanalisi prende il nome di **crittologia**[8].

## 2.2 Le tradizionali tecniche di codifica di dati

In passato l'unico modo per garantire la segretezza dei dati era l'utilizzo delle tradizionali *tecniche di codifica*. Tale approccio prevede l'esistenza di una coppia di funzioni di trasformazione di dati, l'una inversa dell'altra, implementate da un **algoritmo di codifica** ed un **algoritmo di decodifica**.

Per inviare un messaggio riservato $M$ su di un canale insicuro, si trasmette in realtà il messaggio $M$' ottenuto come applicazione dell'algoritmo di codifica su $M$ (Figura 2.1).

Il destinatario, una volta ricevuto $M$', applica l'algoritmo di decodifica, ottenendo in uscita il messaggio originale $M$. Le due funzioni sono note soltanto agli utenti che vogliono comunicare, e la loro segretezza dev'essere garantita per tutto il tempo per cui i dati devono rimanere segreti.

---

[7] In questo contesto la parola "testo" si riferisce ad una generica stringa di bit o byte, non soltanto a stringhe di testo.

[8] Bruce Schneier, *"Applied Cryptography Second Edition : protocols, algorithms, and source code in C*", John Wiley & Sons, Inc.

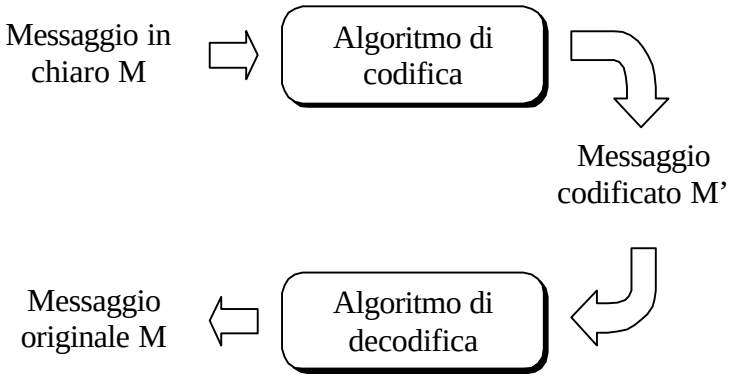


*Figura 2.1. Flusso dei dati previsto dal tradizionale schema a codifica e decodifica.*

Da un punto di vista *matematico*, le due funzioni di trasformazione sono funzioni bigettive, con insiemi di partenza e di arrivo coincidenti con l'insieme delle stringhe di bit[9].

La coppia di algoritmi è progettata in modo che, per una terza entità "ostile" in grado di registrare e/o modificare il traffico sulla rete, senza la conoscenza degli algoritmi stessi,

1. sia "impossibile", a partire dal messaggio codificato $M$', recuperare il messaggio originale $M$, od anche soltanto una parte di esso
2. sia "impossibile" modificare il messaggio codificato $M$' in

[9] Le tecniche di codifica delle informazioni sono un'invenzione antichissima dell'uomo, e venivano applicate manualmente per nascondere il contenuto di messaggi durante la loro trasmissione mediante corrieri. Tra le più note citiamo: codifica monoalfabetica (utilizzata fin dai tempi di Cesare), codifica omofonica, codifica a sostituzione di poligrammi, codifica a mescolamento di caratteri.

modo che dopo la decodifica si ottenga un secondo messaggio $M$" voluto; generalmente, se si modificasse un messaggio codificato senza conoscere gli algoritmi, dopo la decodifica si otterrebbe qualcosa di insensato, e ciò verrebbe immediatamente rilevato dal destinatario dopo la decodifica.

Il meccanismo di codifica si fonda sul concetto di *segretezza dell'algoritmo*, cioè garantisce la segretezza delle informazioni scambiate fintantoché gli algoritmi di codifica e decodifica rimangono segreti ai due o più interlocutori.

Tale approccio presenta diversi svantaggi:

- ciascun gruppo di utenti deve inventarsi una sua coppia di algoritmi privata, da tenere segreta e non rivelare a nessuno, quindi:
  - c'è un limitatissimo scambio di opinioni sulla "sicurezza" degli algoritmi; non è consigliabile delegare ad esterni specializzati la progettazione di nuovi algoritmi né l'analisi di algoritmi già in uso, in quanto ciò ne comprometterebbe la sicurezza
  - ogni gruppo di utenti deve realizzare un software privato che implementi gli algoritmi; non sarebbe una buona scelta delegare ad esterni specializzati l'implementazione degli algoritmi, perché ciò ne comprometterebbe la

sicurezza

- l'approccio non è applicabile per gruppi in cui ci sia un continuo ricambio di utenti, perché ogni volta che un utente esce dal gruppo bisogna che si riprogetti e reimplementi una nuova coppia di algoritmi

Nonostante ciò, tali tecniche sono largamente utilizzate in applicazioni che devono garantire un basso livello di sicurezza.

## 2.3 Le moderne tecniche crittografiche per la codifica di dati

La crittografia moderna adotta tecniche che, pur rientrando nella categoria delle tecniche di codifica di dati, per quanto concerne il problema della sicurezza si fonda su principi diametralmente opposti.

In questo caso si ha un algoritmo di codifica (*cifratore*[10]) che trasforma una sequenza di bit $M$ in una sequenza $M'$ utilizzando una sequenza di bit $K$, detta *chiave di cifratura*, ed un algoritmo di decodifica (o *decifratore*) che, nota una seconda chiave $K'$, detta *chiave di decifratura*, decodifica la sequenza $M'$ ottenendo nuovamente $M$ (Figura 2.2). Gli algoritmi di cifratura e decifratura sono tali da rendere “impossibile” il recupero del messaggio originale $M$ od anche soltanto di parte di esso senza la conoscenza della

chiave $K'$, ove la parola “impossibile” in questo contesto va intesa nel senso di “non praticabile”, come verrà chiarito in seguito.

Questo tipo di approccio ha rivoluzionato il modo di procedere tradizionale. Infatti con la crittografia, oggi, gli algoritmi di trasformazione possono essere pubblicamente disponibili, mentre la sicurezza delle informazioni si appoggia completamente alla *segretezza delle chiavi* di cifratura e decifratura.

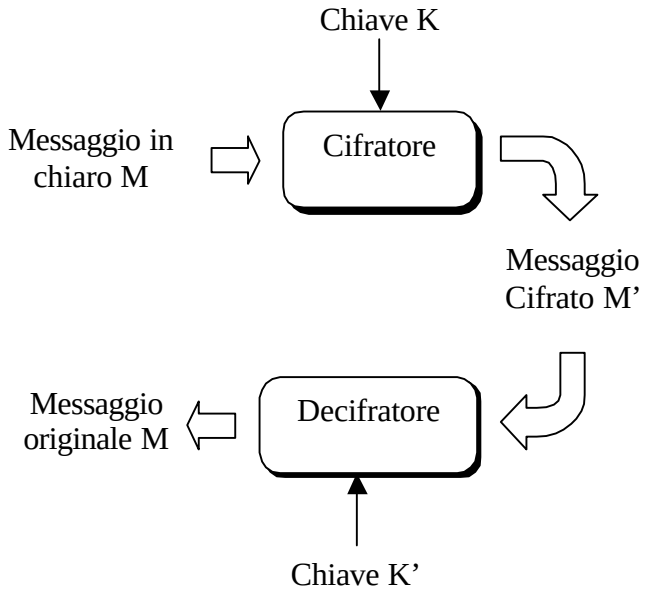


*Figura 2.2. Flusso dei dati previsto dalle moderne tecniche crittografiche di codifica.*

Il fatto che i dettagli di tali algoritmi siano disponibili al pubblico non costituisce alcuna minaccia per gli utilizzatori, perché senza la conoscenza della chiave di decifratura è comunque “impossibile”

[10] Ho scelto di adottare il termine “cifrare” anziché “codificare” per indicare operazioni di codifica condotte secondo moderni algoritmi crittografici, pur essendo i due termini

risalire al testo in chiaro. Al contrario, la pubblicazione degli algoritmi costituisce una *garanzia*, perché essi possono essere studiati da crittoanalisti di tutto il mondo ed eventuali loro “debolezze” (nel senso della sicurezza) sarebbero individuate molto più rapidamente.

Un ulteriore vantaggio delle tecniche crittografiche rispetto a quelle tradizionali di codifica di dati riguarda la possibilità di sviluppo di case software *specializzate* nella progettazione ed implementazione di algoritmi crittografici, permettendo così:

- il raggiungimento di un alto livello di specializzazione, quindi la produzione di software di qualità molto elevata
- l'adozione di tecniche sofisticate per la protezione di dati anche da parte di aziende che non hanno personale specializzato nel campo della sicurezza
- la nascita e la crescita di un mercato per la compravendita di software crittografico, in cui la libera concorrenza fra imprese permette l'abbassamento dei costi, e conseguentemente un’ampia diffusione di tale software, a tutto vantaggio dell'utenza.

All'interno degli algoritmi crittografici possiamo distinguere due insiemi fondamentali:

- **algoritmi simmetrici**, caratterizzati dal fatto che la chiave $K$’

---

sinonimi.

di decifratura è calcolabile da quella di cifratura[11] $K$ e viceversa

- **algoritmi asimmetrici**, caratterizzati dal fatto che le due chiavi sono diverse, ed inoltre o risulta “impossibile” calcolare $K$ da $K$’, o risulta “impossibile” calcolare $K$’ da $K$.

## 2.4 Algoritmi simmetrici

L'utilizzo di tali algoritmi, in letteratura noti anche come **algoritmi a chiave segreta**, o *a chiave singola*, o *a chiave unica*, richiede che le due entità che vogliono comunicare concordino segretamente una chiave $K$ da utilizzare per la codifica e decodifica dei messaggi. La sicurezza della comunicazione è garantita fintantoché la chiave rimane nota soltanto alle due entità.

Ciò che rende l'uso di questi algoritmi problematico è proprio la *consegna della chiave* dall'entità che la genera all'altra, infatti tale consegna dev’essere effettuata in modo sicuro. Il problema verrà ripreso dopo aver trattato gli algoritmi asimmetrici.

[11] In tali algoritmi, spesso, l'algoritmo di decifratura comprende la trasformazione preliminare della chiave di cifratura *K* in quella per la decifratura *K’*. Quindi si parla di un'unica *chiave simmetrica K*, da utilizzarsi sia per la cifratura che per la decifratura. Da qui il nome di *algoritmi simmetrici*.

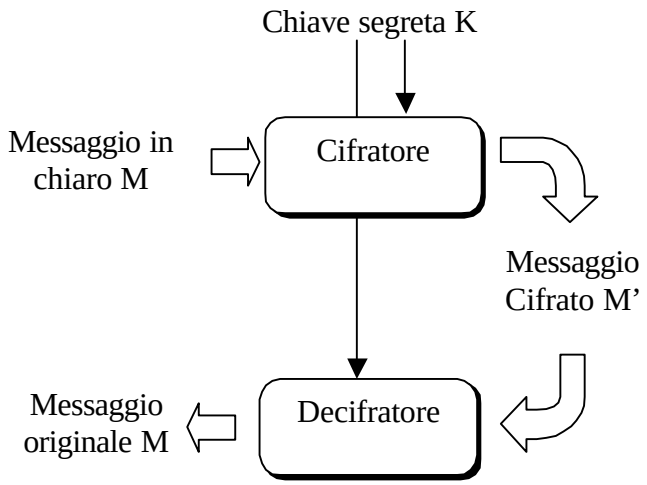


*Figura 2.3. Adozione di tecniche di crittografia simmetrica per l'occultamento del contenuto del messaggio M da trasmettere su di un canale insicuro.*

Gli algoritmi simmetrici sono classificabili in due categorie:

- algoritmi **a blocco**
- algoritmi **a flusso**

*Algoritmi simmetrici a blocco*

Un cifratore o decifratore a blocco suddivide i dati in ingresso in blocchi di lunghezza prefissata (tipicamente 64 bits), per poi cifrarli singolarmente mediante l'applicazione di una funzione di trasformazione invertibile.

Per poter applicare un algoritmo a blocco, il messaggio da cifrare deve essere di dimensione multipla di quella del blocco elementare, quindi vanno inseriti opportuni byte di allineamento se ciò non si verifica.

La funzione applicata ad ogni blocco dipende soltanto dal particolare valore della chiave fornita. Quindi, fissata una chiave $K$, uno stesso blocco di testo in chiaro $M_j$ verrà sempre trasformato in uno stesso blocco di testo cifrato $M'_j$. Questa modalità di operazione dei cifratori a blocco è nota come **ECB** (Electronic CodeBook).

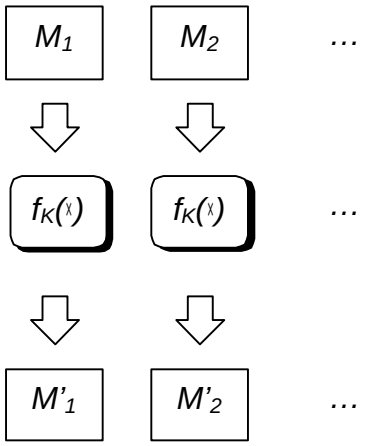


*Figura 2.4. Flusso dei dati in un cifratore in modalità ECB: ogni blocco $M_i$ subisce lo stesso algoritmo di trasformazione $f_k(\gamma)$.*

Una caratteristica positiva dell'ECB è una buona predisposizione al parallelismo. Disponendo di più unità di elaborazione in hardware, si possono cifrare più blocchi indipendentemente l'uno dall'altro, e contemporaneamente. La controparte negativa risiede nella maggior esposizione alle tecniche di crittoanalisi. Un crittoanalista può riconoscere due blocchi di testo in chiaro uguali confrontando i rispettivi blocchi in cifra, e questo può essere sufficiente a sferrare alcuni tipi di attacco noti come **block replay**.

Alcuni cifratori adottano dei meccanismi per prevenire questi tipi di attacco, implementando degli opportuni “anelli di retroazione”. Ciò significa che la funzione di trasformazione viene a dipendere, oltre che dalla chiave, anche da uno stato interno che viene aggiornato di volta in volta. In definitiva, in questo tipo di cifratori, la funzione di trasformazione applicata a ciascun blocco dipende, oltre che dalla chiave K, anche da tutti i blocchi precedenti.

Una modalità di operazione dei cifratori a blocco, nota come **CBC** (Cipher Block Chaining), prevede che ciascun blocco in chiaro, prima di essere cifrato, venga messo in *xor* con il blocco in cifra calcolato al passo immediatamente precedente.

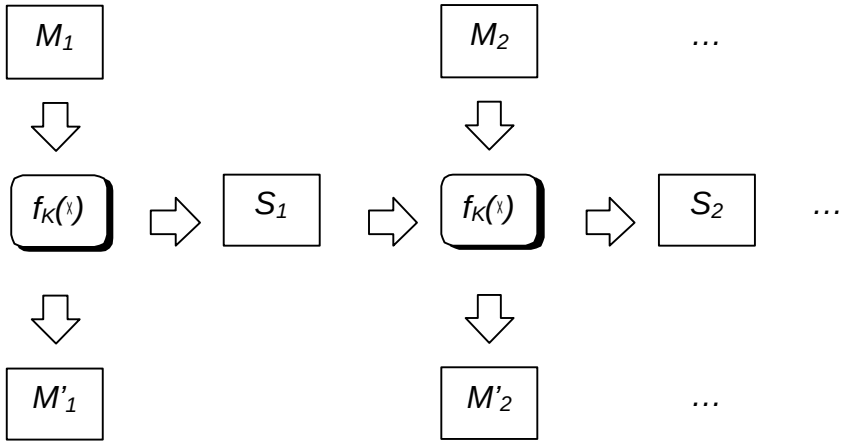


*Figura 2.5. Flusso dei dati in un cifratore che previene attacchi di tipo “block replay”. { $S_j$ } rappresenta la sequenza degli stati interni che il cifratore attraversa durante la codifica del messaggio.*

Adesso a due blocchi in chiaro identici non necessariamente corrispondono blocchi in cifra identici. Comunque due messaggi

identici verranno cifrati in modo identico. Per evitare ciò, viene inserito in testa ad ogni messaggio da cifrare un blocco casuale di dati, noto come **vettore di inizializzazione**, che non è necessario mantenere segreto.

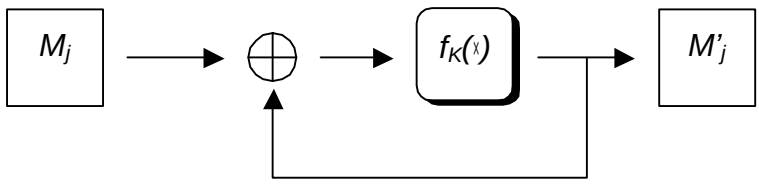


*Figura 2.6. Cifratore a blocco operante secondo la modalità CBC: ogni blocco d'ingresso $M_j$ viene posto in xor con il blocco precedentemente cifrato.*

*Algoritmi simmetrici a flusso*

Un cifratore o decifratore a flusso opera sul *flusso* dei dati in ingresso, nel senso che ad ogn7i passo cifra un singolo bit o byte, tipicamente mettendolo in *xor* con un singolo bit o byte proveniente da un **generatore di bit a chiave.** Quest'ultimo è in grado di produrre un flusso di bit o byte apparentemente casuale, dipendente dal valore della chiave fornita. Inoltre risulta "impossibile" prevedere la sequenza generata senza la conoscenza della chiave.

Nei cifratori a flusso detti **autosincronizzanti**, il generatore di bit prende in ingresso, ad ogni passo, anche un numero k di bit cifrati precedentemente. Quindi tali cifratori sono paragonabili ai cifratori a blocco in CBC.

Sostanzialmente i cifratori o decifratori autosincronizzanti sono degli automi a stato finito, che ad ogni passo prendono un bit o byte e forniscono in uscita un bit o byte, aggiornando il proprio stato interno. Sia la funzione di uscita, che quella di trasformazione di stato, dipendono dal particolare valore della chiave.

Comunque, fissata una chiave, è evidente che uno stesso bit o byte può essere cifrato in modo diverso a seconda di ciò che lo precede nel flusso. Questa proprietà rende più difficile la crittoanalisi di messaggi cifrati, in quanto blocchi identici di testo in chiaro vengono cifrati differentemente.

Con riferimento ad un flusso di dati che viene trasmesso su di un canale di comunicazione potenzialmente rumoroso, in conseguenza ad un errore di trasmissione di un bit o byte, il decifratore sbaglierà, al più, nella decifrazione dei k bit o byte successivi. Infatti la decifrazione di un singolo elemento dipende, oltre che dalla chiave, anche dai k elementi precedenti.

Questo problema non esiste nei cosiddetti **cifratori sincroni**, in cui, invece, il flusso dei bit generati è completamente indipendente dal flusso dei dati in ingresso.

*Algoritmi “ibridi”*

Si è già notata la somiglianza fra cifratori a blocco in CBC e cifratori a flusso autosincronizzanti. Un'ulteriore modalità di

operazione dei cifratori a blocco è la **CFB** (*Cipher FeedBack*). In essa, essenzialmente per problemi di memoria su dispositivi “poco intelligenti”, la cifratura elementare di un blocco viene effettuata su gruppi di 4 od 8 bit alla volta, man mano che questi bit giungono al cifratore. La situazione corrisponde all'adozione di un cifratore a flusso per cifrare un singolo blocco, all'interno di uno schema di cifratura a blocco.

## 2.5 Algoritmi asimmetrici

Tali algoritmi sono anche detti **a chiave pubblica** perché una delle due chiavi viene tipicamente resa pubblica, cioè viene distribuita in qualche modo a chiunque sia interessato al suo reperimento, mentre l'altra viene tenuta segreta. In ogni caso gli algoritmi sono progettati in modo che sia “impossibile” calcolare la chiave segreta conoscendo quella pubblica.

È possibile suddividere questi algoritmi in due categorie che trovano applicazione in campi molto diversi fra loro:

- algoritmi a chiave di *cifratura* pubblica: permettono a qualsiasi entità di spedire in modo *riservato* messaggi ad un'entità destinataria specifica, cifrandoli con la chiave di cifratura pubblica che essa rende disponibile a chiunque; il messaggio può poi transitare liberamente su di una rete aperta, poiché soltanto il destinatario, in possesso della corrispondente

chiave privata di decifratura, può recuperare il messaggio originale

- algoritmi a chiave di *decifratura* pubblica: permettono a chiunque di decifrare un messaggio cifrato da un'entità, utilizzando la chiave di decifratura che l'entità rende pubblicamente disponibile; in questo caso il mittente cifra il messaggio *non per nasconderne il contenuto* (infatti chiunque può decifrarlo), ma per *garantirne l'autenticità*.

*Algoritmi a chiave di cifratura pubblica*

Gli algoritmi asimmetrici che rientrano in questa categoria permettono di ottenere *riservatezza* nelle comunicazioni fra entità di rete, analogamente agli algoritmi simmetrici. Comunque, mentre questi ultimi permettono la realizzazione di un canale di comunicazione *bidirezionale* privato con una singola chiave, gli algoritmi asimmetrici invece permettono la realizzazione di un canale *unidirezionale* con una singola coppia di chiavi, ma su tale canale possono essere inviati dati da chiunque, mentre soltanto il destinatario (il “possessore” della coppia di chiavi) può riceverli.

Un'ulteriore differenza fra algoritmi simmetrici ed asimmetrici per la riservatezza delle comunicazioni è costituita dal fatto che i primi non costituiscono una soluzione *scalabile* relativamente all'ampliamento dell'insieme di utenti che hanno bisogno di

scambiarsi messaggi segreti.

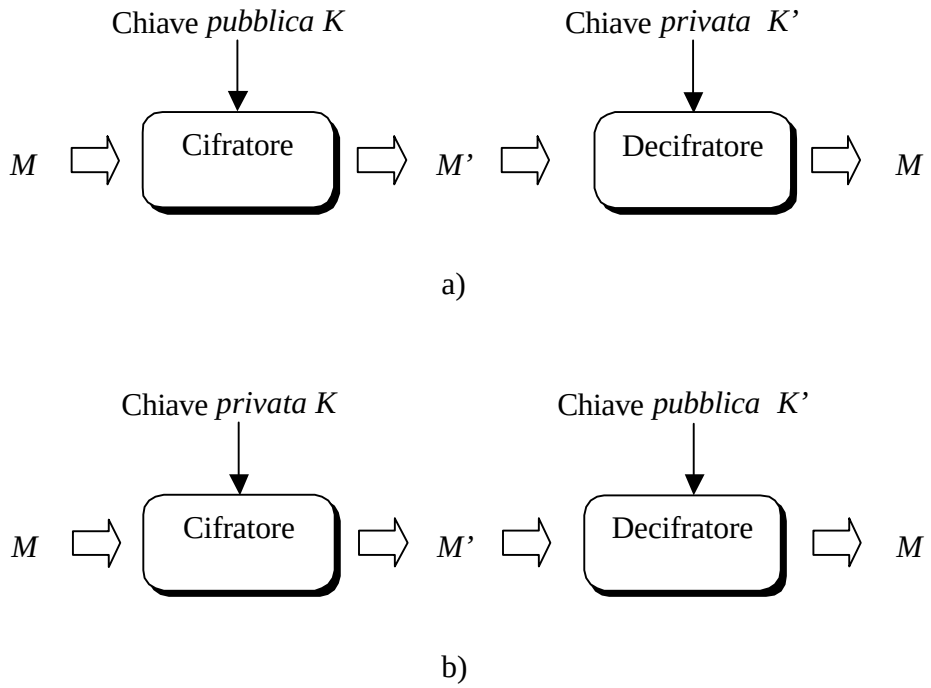


*Figura 2.7. Utilizzo della crittografia asimmetrica per i servizi di riservatezza dei dati (a) e autenticazione (b).*

Infatti è necessaria una chiave segreta distinta per ogni coppia di utenti, con una crescita *quadratica* del numero di chiavi col numero di utenti. Con gli algoritmi asimmetrici, invece, è sufficiente che ogni utente abbia soltanto una chiave pubblica di cifratura (ed una privata corrispondente), con la quale può ricevere dati privati da tutti gli altri, quindi si ha una crescita *lineare* del numero di chiavi col numero di utenti.

Bisogna comunque tener presente che, a parità di volume di dati da cifrare, gli algoritmi simmetrici sono molto più veloci di quelli asimmetrici. È per questo che le tecniche di cifratura simmetrica,

dopo la scoperta di quelle a cifratura asimmetrica da parte di Diffie ed Hellman, non sono divenute obsolete. Infatti sarebbe assurda l'adozione di algoritmi asimmetrici per cifrare l'intero traffico di informazioni che due o più entità si scambiano su di una rete, perché si otterrebbe un degrado delle prestazioni ingiustificato, data l'esistenza degli algoritmi simmetrici molto più efficienti.

Nei casi pratici si ha una combinazione di un algoritmo simmetrico ed uno asimmetrico: il traffico informativo vero e proprio fra due entità viene cifrato interamente con una chiave simmetrica, che preliminarmente una delle entità genera e spedisce segretamente all'altra, utilizzandone la chiave pubblica di cifratura.

Tipicamente, per ogni nuova sessione di comunicazione fra una coppia di entità, viene generata “al volo” una nuova chiave, che in letteratura prende il nome di **chiave di sessione**.

*Algoritmi a chiave di decifratura pubblica*

In questa categoria rientrano le cosiddette “tecniche di firma digitale”, che permettono di *autenticare* dei dati dopo la loro trasmissione su di una rete, o riproduzione, senza necessariamente mantenerne la segretezza. *Autenticare* dei dati significa sia garantire due cose:

1. che provengono effettivamente dal loro mittente presunto
2. che dopo la loro emissione non abbiano subito alcuna

modifica né aggiunta

È opportuno ricordare che le operazioni crittografiche vengono sempre applicate su insiemi di dati opportunamente ridondanti, in modo che sia sempre possibile, dopo un'operazione di decifratura, verificare che ci si trovi di fronte ai dati attesi.

Solo dopo questa verifica si ha la certezza che i dati appena decifrati sono stati effettivamente cifrati con la chiave privata corrispondente a quella pubblica appena utilizzata.

L'autenticità, quindi, è garantita se decifrando i dati con la chiave pubblica del presunto mittente, si ottiene qualcosa di “sensato”. In realtà, il successo dell'operazione di decifratura ci permette di concludere soltanto che la chiave utilizzata per la cifratura è quella corrispondente alla chiave pubblica da noi usata per la decifratura.

Ma chi ci garantisce che la chiave di decifratura da noi utilizzata sia effettivamente la chiave pubblica del presunto mittente e non quella di un'altra entità? Tale garanzia, come vedremo più avanti, ci viene fornita da una **autorità di certificazione**, mediante l'emissione di un **certificato**.

Il fatto che questa categoria di algoritmi asimmetrici sia utilizzata in applicazioni che non richiedono necessariamente segretezza nelle comunicazioni, permette di effettuare una notevole *ottimizzazione* delle elaborazioni. Infatti è sufficiente che, sia nella fase di cifratura

che in quella di decifratura, si operi non direttamente sul messaggio, ma sulla sua cosiddetta **impronta digitale**.

## 2.6 Impronte digitali

L'impronta digitale di una stringa di bit $S$ è il risultato dell'applicazione, ad essa, di una **funzione "hash" crittograficamente sicura**. Ciò che si ottiene è una seconda stringa di bit $S$', di dimensione prefissata indipendente dalla lunghezza della stringa d'ingresso, e dalla quale risulti "impossibile" risalire anche ad una soltanto delle stringhe che hanno $S$' come impronta. Nella terminologia delle funzioni hash, la stringa d'ingresso $S$ prende anche il nome di **preimmagine**, mentre quella d'uscita $S$' prende il nome di **valore hash**.

Da un punto di vista *matematico*, un algoritmo che calcola impronte digitali realizza una funzione *f(x) non iniettiva* dall'insieme $V$ delle possibili stringhe di bit di dimensione qualsiasi, all'insieme $W$ delle stringhe di bit di una lunghezza costante prefissata (Figura 2.8). Tipicamente tale lunghezza risulta sempre inferiore alla lunghezza della preimmagine cui l'algoritmo viene applicato, da cui la necessaria non iniettività di $f$. La funzione calcolata da un tale algoritmo risulta essere una *funzione hash*, cioè esegue un mapping il più "casuale" possibile di preimmagini in valori hash: l'alterazione di

un solo bit nella preimmagine causa tipicamente l'alterazione di metà dei bit del corrispondente valore hash, distribuiti “casualmente” all'interno di esso.

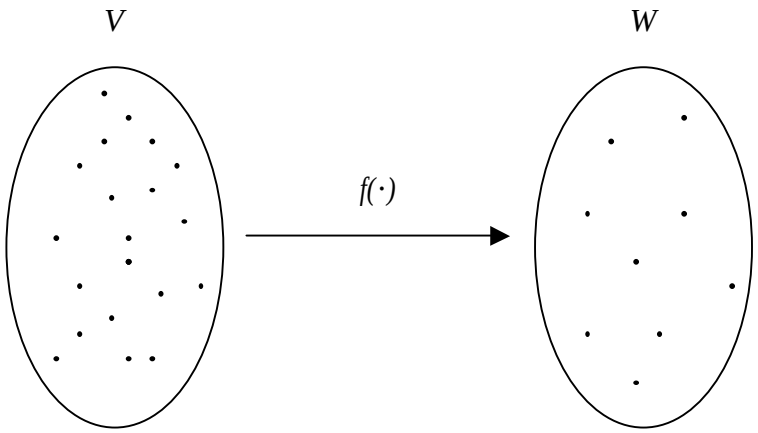


*Figura 2.8. Rappresentazione insiemistica di una funzione “hash”.*

Da un punto di vista *informatico*, il fatto che un tale algoritmo sia “crittograficamente sicuro” significa che esso realizza una funzione **a senso unico** (one way function), cioè “praticamente calcolabile in un solo senso”. Più precisamente, una **funzione a senso unico** $f(x)$ risulta definita dalle seguenti proprietà:

1. dato un $x \ in \ V$, ed $y = f(x)$, calcolare $f(x)$ a partire da $x$ risulta un'operazione “semplice”, mentre calcolare da $y$ uno degli $x'$ tali che $y=f(x')$ risulta “impossibile”
2. la funzione è **priva di collisioni**, cioè risulta “impossibile” trovare due valori $x_1 e \ x_2 \ in \ V$ tali che $f(x_1) = f(x_2)$.

Si noti che quella appena enunciata non costituisce una definizione nel senso *matematico* del termine, in quanto contiene parole come “semplice” ed “impossibile”. Come già accennato altrove in questa trattazione, i due termini si riferiscono alla possibilità o meno di avere a disposizione, ad un *costo abbordabile*, un *sistema reale* in grado di effettuare determinate operazioni *in un tempo utile*.

*Utilizzo delle impronte digitali*

È possibile verificare l'uguaglianza due stringhe di bit confrontando le rispettive impronte digitali, anziché le stringhe stesse. Se due stringhe hanno diversa impronta digitale, sicuramente differiscono per almeno un bit. Ma se due stringhe hanno la stessa impronta, nulla ci garantisce che le due stringhe siano identiche. Comunque il punto, qui, è che esistono diversi livelli di verifica dell'uguaglianza di due stringhe:

1. le due stringhe sono *certamente* uguali se superano un test di confronto bit a bit
2. le due stringhe sono *ragionevolmente* uguali se le loro impronte digitali coincidono;

Come accennato al paragrafo precedente, per garantire l'autenticità di un messaggio si può applicare un algoritmo asimmetrico a chiave di cifratura privata alla sua impronta digitale, anziché direttamente al messaggio, per motivi di ottimizzazione.

Infatti, in tal modo, il cifratore asimmetrico è attraversato soltanto dall'impronta digitale, che risulta generalmente molto più corta rispetto al messaggio originale.

L'impronta cifrata così ottenuta, allegata al messaggio originale in chiaro, garantisce integrità ed autenticità del messaggio, nell'ipotesi che soltanto il suo autore conosca la chiave privata di codifica. Un utente qualsiasi può accertare la provenienza di un tale messaggio mediante i seguenti passi:

1. l'utente estrae dal messaggio il nome del mittente (*mittente presunto)*
2. l'utente si procura la chiave pubblica di decodifica del mittente presunto (il problema di come avvenga tale reperimento verrà trattato in seguito)
3. l'utente decifra l'impronta cifrata allegata al messaggio originale, ottenendo la sua *presunta* impronta digitale
4. l'utente, avendo a disposizione il messaggio in chiaro, ne calcola l'impronta digitale
5. l'utente confronta l'impronta digitale appena calcolata con quella decifrata: se le impronte coincidono, l'autenticità è garantita.

Se un utente "ostile" volesse modificare il contenuto del messaggio, dovrebbe alterare conseguentemente anche l'impronta

cifrata ad esso allegata, ma, essendo quest'ultima ottenuta mediante una cifratura asimmetrica con una chiave privata, egli non sarebbe in grado di ricifrare tale impronta dopo la modifica. Allora l'unica possibilità di creare un messaggio falso apparentemente autentico sarebbe quella di cercare un secondo messaggio avente la stessa impronta digitale dell'originale, ed allegarvi l'impronta cifrata allegata all'originale stesso. La prima proprietà su esposta per l'impronta digitale garantisce l'insuccesso di un'eventuale ricerca di questo tipo.

Una tipologia di attacco più sottile è quella di un utente che, coscientemente, firma un messaggio riservandosi l'eventuale possibilità di dimostrare, in futuro, di averne firmato un altro. Ciò sarebbe possibile se egli fosse in grado di individuare, in partenza, due messaggi $M_1$ ed $M_2$ rispondenti ai suoi scopi e con la stessa impronta digitale. In un primo momento egli potrebbe pubblicare $M_1$ con allegata la sua firma digitale (ed eventualmente anche la firma digitale di un notaio che attesti la data di sottoscrizione del messaggio). Come già detto, in realtà verrà cifrata soltanto l'impronta digitale di $M_1$. In un secondo momento egli potrebbe mostrare il messaggio $M_2$, con allegate le firme ottenute precedentemente, ed il processo di verifica delle firme si concluderebbe con successo. L'utente potrebbe asserire di aver firmato il messaggio $M_2$, non $M_1$. Una tale situazione sarebbe estremamente dannosa se pensiamo

che il messaggio in questione potrebbe essere un contratto. È appunto per evitare situazioni del genere che si richiede l'assenza di collisioni nella definizione di un algoritmo di impronta digitale.

## 2.7 Crittoanalisi

Le tecniche di crittoanalisi cercano di realizzare tutto ciò che nelle precedenti sezioni è stato aggettivato come "impossibile". Lo scopo del crittoanalista dipende dall'algoritmo di crittografia che si prende in considerazione:

- per messaggi cifrati con chiavi simmetriche, la crittoanalisi cerca di recuperare il messaggio in chiaro originario e, se possibile, anche la chiave di cifratura, allo scopo di decifrare ulteriori messaggi cifrati con la stessa chiave e/o modificare i messaggi scambiati fra le entità durante la comunicazione
- per messaggi cifrati con chiavi pubbliche, la crittoanalisi cerca di recuperare il messaggio in chiaro originario e, se possibile, la chiave segreta di cifratura, allo scopo di modificare il contenuto dei messaggi inviati da altri
- per messaggi cifrati con chiavi segrete, la crittoanalisi cerca di individuare la chiave segreta di cifratura, allo scopo di modificare il contenuto del messaggio facendolo sembrare autentico

Nella terminologia della crittografia, un tentativo di crittoanalisi di

un messaggio cifrato è detto **attacco**. Un'assunzione fondamentale fatta in crittoanalisi è che il crittoanalista abbia una conoscenza dettagliata degli algoritmi di cifratura e decifratura, e delle loro implementazioni in uso, anche se nel mondo reale ciò può non essere sempre vero. Ci sono quattro categorie di attacchi crittoanalitici[12], che, in ordine di efficacia, risultano:

- **attacco "a solo testo in cifra"[13]**: si ha a disposizione un numero *n* di messaggi in cifra $\{ C_1, ..., C_n \}$, tutti cifrati usando lo stesso algoritmo e la stessa chiave $K$:

  $$C_1 = E_K(M_1), \ldots C_n = E_K(M_n)$$

  scopo dell'attacco è recuperare $M_i$ per qualche $i$, recuperare la chiave $K$, oppure individuare un algoritmo alternativo che permetta di decifrare messaggi successivi cifrati con $K$

- **attacco "a testo in chiaro noto"[14]**: si ha a disposizione, oltre alla sequenza dei messaggi in cifra, anche la sequenza dei messaggi in chiaro $\{ M_1, ..., M_n \}$ e scopo dell'attacco è recuperare la chiave $K$, o individuare un algoritmo alternativo che permetta di decifrare messaggi successivi cifrati con $K$

[12] Bruce Schneier, *"Applied Cryptography Second Edition : protocols, algorithms, and source code in C"*, John Wiley & Sons, Inc.
[13] *"Ciphertext only attack"*.
[14] *"Known plaintext attack"*.

- **attacco "a scelta di testo in chiaro"**[15]: oltre al caso precedente, l'attaccante ha anche la possibilità di scegliere i messaggi $\{ M_1 , ..., M_n \}$

- **attacco "a scelta adattiva di testo in chiaro"**[16]: simile al caso precedente, con la differenza che il crittoanalista può scegliere i messaggi successivi da cifrare anche in base ai messaggi in cifra ottenuti per i messaggi scelti precedentemente nella sequenza.

Si osservi che gli attacchi "a testo in chiaro noto" sono molto più frequenti di quanto si potrebbe pensare: infatti in quasi ogni attacco il crittoanalista conosce informazioni sulla natura delle informazioni scambiate nel messaggio da recuperare, cioè conosce il *formato* del messaggio, che spesso impone preamboli e/o conclusioni standard, note quindi a priori. Inoltre quasi sempre un testo cifrato presenta delle ridondanze, si pensi ad esempio a del codice sorgente, o ad un documento di testo, ed in questi casi un rimedio sempre efficace è la *compressione* del testo all'ingresso del cifratore.

Il risultati possibili di un attacco crittoanalitico possono essere classificati come segue[17]:

---

[15] *"Choosen plaintext attack"*.
[16] *"Adaptive choosen-plaintext attack"*.
[17] Bruce Schneier, *"Applied Cryptography Second Edition : protocols, algorithms, and source code in C*", John Wiley & Sons, Inc.

- **rottura totale**[18]: si è recuperata la chiave segreta
- **deduzione globale**[19]: si è individuato un algoritmo alternativo che decodifica i messaggi in cifra senza conoscere la chiave di decifratura, o che cifra messaggi senza conoscere la chiave di cifratura
- **deduzione locale**[20]: si è decifrato un singolo messaggio in cifra
- **deduzione di informazione**: si è ottenuta qualche informazione sul testo in chiaro o sulla chiave segreta

Un'ultima osservazione sugli attacchi crittoanalitici riguarda una forma particolare di attacco che è teoricamente sempre efficace contro tutti gli algoritmi crittografici a chiave: l'**attacco a forza bruta**[21], che effettua una ricerca esaustiva della chiave sconosciuta. Esso, nel caso di algoritmi simmetrici ed asimmetrici a chiave di cifratura pubblica, consiste nel provare a decifrare un testo in cifra con tutte le possibili chiavi, fino a quando non si ottiene qualcosa di "sensato", che risponde cioè alla tipologia o formato di testo in chiaro che ci si aspetta.

Nel caso di algoritmi asimmetrici a chiave di cifratura segreta, un attacco a forza bruta, dopo aver decifrato il messaggio con la chiave

---

[18] *"Total break"*.
[19] *"Global deduction"*.
[20] *"Local deduction"*.
[21] *"Brute force attack"*.

pubblica, prova a ricifrarlo con tutte le possibili chiavi, fino a quando si ottiene esattamente il messaggio cifrato originario. In questo caso l'attacco, per riuscire, non ha bisogno di alcuna conoscenza preliminare sul contenuto del messaggio, a differenza del caso precedente.

## 2.8 Considerazioni sulla sicurezza degli algoritmi crittografici

Da un punto di vista puramente teorico, tutti gli algoritmi crittografici a chiave sono “violabili” tramite ricerca esaustiva della chiave. Fortunatamente ciò è vero soltanto teoricamente, in quanto da un punto di vista pratico si può fissare la lunghezza della chiave ad un valore sufficientemente elevato da rendere impraticabile una ricerca esaustiva con i mezzi e le tecnologie a disposizione dell'uomo, per tutto il tempo per cui interessa proteggere i dati.

La ricerca è resa impraticabile dalla limitatezza di risorse a disposizione del crittoanalista, intese sia come tempo a disposizione, che come unità di elaborazione a disposizione.

A titolo di esempio, in Tabella 2.1 è riportato il tempo richiesto per effettuare una ricerca esaustiva di chiave, assumendo che sia possibile effettuare un milione di tentativi al secondo, al variare della lunghezza di chiave[22]. È importante osservare che, ad un guadagno

[22] Bruce Schneier, “*Applied Cryptography Second Edition : protocols, algorithms, and source code in C*”, John Wiley & Sons, Inc.

in termini di sicurezza derivante dall'incremento della lunghezza della chiave, corrisponde un degrado in termini di prestazioni degli algoritmi crittografici. Quindi non è possibile incrementare a piacere la dimensione delle chiavi, soprattutto se gli algoritmi di cifratura devono mantenere un throughput elevato.

*Tabella 2.1. Tempo necessario per una ricerca esaustiva di chiave, nell'ipotesi che sia possibile effettuare un milione di tentativi al secondo.*

| **Lunghezza di chiave** | **Numero chiavi** | **Tempo di ricerca** |
|---|---|---|
| 32 bits | $4,3*10^{9}$ | 1,2 ore |
| 40 bits | $1,1*10^{12}$ | 13 giorni |
| 48 bits | $2,8*10^{14}$ | 8,9 anni |
| 56 bits | $7,2*10^{16}$ | 2.300 anni |
| 64 bits | $1,8*10^{19}$ | 580.000 anni |

Un'altra considerazione riguarda gli algoritmi specifici adottati: ogni algoritmo garantisce un livello di sicurezza diverso, legato, oltre che alla dimensione della chiave utilizzata, anche alla capacità intrinseca dell'algoritmo di resistere ad attacchi crittoanalitici (esistenza di classi di chiavi più “deboli” di altre, esistenza di sequenze particolari in ingresso che rivelano parte della chiave, ecc…). Chiaramente algoritmi più robusti avranno un costo maggiore in termini di risorse (tempo di elaborazione, costo, ecc…).

Allora bisogna trovare un *compromesso* fra *sicurezza* e *costo*. Ma qual è il livello minimo di sicurezza di cui un'applicazione ha

bisogno? La risposta è di carattere *economico*, oltre che *tecnico*: la lunghezza minima dev'essere tale da garantire almeno che:

- il costo richiesto per “violare” l'algoritmo sia superiore al valore dei dati protetti
- il tempo richiesto per “violare” l'algoritmo sia superiore al tempo per cui i dati cifrati devono restare segreti

Infatti, sotto queste ipotesi, qualsiasi tentativo di crittoanalisi dei dati, anche se realizzato con successo, sarebbe un'inutile “perdita” di tempo. Chiaramente il verificarsi delle due condizioni enunciate non basta a garantire la sicurezza dei nostri dati: la crittoanalisi è una scienza sempre in evoluzione, ed è sempre presente la minaccia della scoperta di una nuova tecnica di analisi che abbassi drasticamente il costo del recupero di un messaggio cifrato.

Inoltre non si può assolutamente trascurare la continua evoluzione tecnologica in corso, che porta ad una sempre maggiore ed imprevedibile capacità di elaborazione disponibile a parità di costo.

# Capitolo 3. I principali algoritmi crittografici

In questo capitolo verranno fatti brevi cenni agli algoritmi crittografici più usati oggi. Per ogni algoritmo si cercherà di illustrare il principio basilare di funzionamento, senza entrare troppo nei dettagli, per i quali si rimanda ad un testo specifico.

## 3.1 Algoritmi simmetrici

*Data Encryption Standard*

L'algoritmo **DEA** (*Data Encryption Algorithm*), adottato nel *Data Encryption Standard*, ha rappresentato lo standard per la crittografia a chiave simmetrica dal 1976, data della sua adozione negli USA come standard federale, fino ai primi anni '90.

DEA rientra nella classe dei cifratori a blocchi, con dimensione del blocco pari a 64 bits. Ogni blocco viene cifrato passando attraverso 16 stadi (detti **rounds**), in ognuno dei quali vengono opportunamente effettuate operazioni di sostituzione e permutazione dei bits del

blocco, in base al valore della chiave (a 56 bits). In particolare, in ogni stadio vengono utilizzati soltanto 48 dei 56 bits totali della chiave, opportunamente scelti. In sostanza, quindi, *DEA* opera una serie di 16 cifrature in cascata, ognuna adoperando una *sottochiave* diversa calcolata a partire dalla chiave fornita.

L'algoritmo, essendo basato su operazioni di permutazione, *shift* e *xor* di bits, è progettato in modo che la decifratura di un blocco possa ottenersi in modo quasi identico alla cifratura. L'unica differenza risiede nel fatto che le 16 sottochiavi devono essere fornite ai vari stadi in ordine inverso rispetto a quello adottato nella cifratura.

Per quanto riguarda la sicurezza di *DES*, c'è da dire che è sempre esistito il dubbio che l'NSA[23] avesse nascosto una “trappola” nell'algoritmo, in modo da poter decifrare facilmente messaggi senza la chiave. Da un'indagine effettuata nel '78 dal “U.S. Senate Committee on Intelligence”, sembra che il ruolo dell'NSA nella progettazione di *DES* sia stato soltanto quello di aver certificato l'algoritmo finale come privo di qualsiasi debolezza matematica o statistica.

L'unica accortezza da seguire nell'utilizzo di *DES* è quella di evitare l'uso di poche[24] *chiavi deboli*.

[23] National Security Agency.
[24] Nel testo di Schneier ne vengono citate 64.

*IDEA*

L'International Data Encryption Algorithm è un algoritmo simmetrico sviluppato all'ETH di Zurigo. Utilizza chiavi a 128 bit, ed è considerato molto sicuro, oltre ad essere uno degli algoritmi più conosciuti e diffusi.

Esso prevede l'applicazione una serie di 8 trasformazioni simili, dette **rounds**, al blocco elementare a 64 bits di testo in chiaro. In ogni round il blocco viene suddiviso in 4 sottoblocchi a 16 bit, che, insieme a 6 sottochiavi a 16 bit (estratte dalla chiave a 128 bit), vengono sottoposte a varie operazioni di *XOR*, addizione modulo $2^{16}$ e moltiplicazione modulo $2^{16}+1$. Si ottengono 4 sottoblocchi che vanno in ingresso al round successivo, e così via fino all'ottavo round. In fase di decodifica l'unica differenza è nel calcolo delle 6 sottochiavi.

IDEA è attualmente brevettato negli Stati Uniti e nella maggior parte degli stati europei. Comunque è concesso liberamente il suo utilizzo per scopi non commerciali. Molte implementazioni di IDEA sono liberamente disponibili, come ad esempio in SSLeay, PGP, SSH, IDEA86 e Crypto++.

*Enigma*

È stato il cifratore utilizzato dai Tedeschi nella seconda guerra mondiale. Con i computer moderni la sua violazione è diventata

banale. Si osservi che esso è l'algoritmo utilizzato dal comando UNIX *crypt*, che quindi non dovrebbe essere più utilizzato.

*RC4*

Questo è un cifratore simmetrico progettato dalla *RSA Data Security, Inc*., che utilizza chiavi di lunghezza arbitraria. Il “cuore” di RC4 è essenzialmente un generatore di numeri pseudo-casuali crittograficamente sicuro, il cui flusso di dati in uscita è posto in *XOR* con i dati in ingresso. Naturalmente, il seme iniziale del generatore è calcolato a partire dalla chiave fornita.

Nonostante la sua sicurezza non sia ancora stata completamente accertata, questo algoritmo è utilizzato in molte applicazioni per la notevole velocità.

Il Governo degli Stati Uniti non blocca l'esportazione di software che usi RC4 con chiavi di lunghezza pari o inferiore a 40 bits, ed infatti chiavi così corte possono essere violate da governi, criminali ed amatori. A conferma di ciò, si osservi che l'implementazione esportabile di SSL della Netscape è stata violata in circa 8 giorni da almeno due gruppi indipendenti.

## 3.2 Algoritmi a Chiave Pubblica

*Merkle-Hellman*

Whitfield Diffie e Martin Hellman furono i primi a presentare il

concetto di crittografia a chiave pubblica alla National Computer Conference del 1976, mentre Ralph Merkle pubblicò il suo primo contributo nel 1978. Il primo algoritmo a chiave pubblica, sviluppato da Merkle ed Hellman, permetteva soltanto la cifratura di messaggi, ma successivamente fu adattato anche per l'apposizione di firme digitali.

La loro idea si fondava sulle difficoltà nella risoluzione del **problema dello zaino**. Infatti quest'ultimo, rientrando nella classe dei problemi *NP-completi*, presenta una *complessità* di risoluzione esponenziale. Sinteticamente, una volta suddiviso il flusso di bit da cifrare in blocchi $B_k$ di lunghezza prefissata $L$, Diffie ed Hellman pensarono di interpretare ciascun blocco come soluzione di un problema dello zaino[25] ad L oggetti, ove la sequenza degli $L$ pesi $P_i$ è prefissata. Per ogni blocco $k$ è possibile calcolare il peso totale $S_k$ tale che il problema dello zaino così costruito ha quel blocco come soluzione:

$$S_k = S_i B_k(i) \times P_i$$

La sequenza dei pesi totali è il testo cifrato, mentre la sequenza dei pesi è la chiave privata.

Se il problema è caratterizzato da una sequenza di pesi

[25] L'i-esimo bit del blocco rappresenta la presenza o assenza dell'i-esimo oggetto nello zaino.

**superincrementante**, tale cioè che ogni peso è strettamente maggiore della somma dei pesi precedenti, allora la complessità di risoluzione diviene lineare. Quindi il destinatario di un messaggio, possedendo la sequenza dei pesi totali, può risolvere ciascun problema in tempo lineare, ottenendo il blocco di testo in chiaro corrispondente.

Ma allora sarebbe necessaria la conoscenza della sequenza dei pesi per cifrare un messaggio, quindi dovrebbe essere essa stessa la chiave pubblica, ciò che metterebbe chiunque nelle condizioni di invertire la trasformazione.

Invece Merkle ed Hellman hanno pensato di pubblicare una sequenza di pesi opportunamente modificata, in modo che l'utente generico potesse calcolare *non* le vere somme totali, ma delle *somme modificate*. Assegnati due interi $n$ ed $m$, la chiave pubblica risulta essere la sequenza $\{Q_i\}$, ove $Q_i = (P_i \times n) \bmod m$.

In tal modo, la somma $R_k$ calcolata per il $k$-esimo blocco risulta:

$$R_k = (S_k \times n) \bmod m$$

Il destinatario, conoscendo $n$ ed $m$, può calcolare

$$R_k \times n^{-1} = S_k,$$

e quindi ottenere il $k$-esimo blocco del testo in chiaro come

soluzione di un problema a pesi superincrementanti. Ogni altro utente, anche nell'ipotesi che indovinasse gli interi $n$ ed $m$, non conoscendo i veri pesi, si troverebbe di fronte ad un problema dello zaino, nel campo degli interi modulo $m$, a pesi *non superincrementanti*, quindi di complessità esponenziale.

Per l'utilizzo reale, tale algoritmo deve considerare blocchi di almeno 250 bits, con pesi di almeno 200-400 bits, ed un valore di $m$ ad almeno 100-200 bits.

Nonostante la buona impostazione teorica, l'algoritmo di Merkle ed Hellman ha mostrato di avere diverse “debolezze” che ne diminuiscono sensibilmente l'affidabilità. Altrettanto è stato trovato per le diverse varianti proposte successivamente.

*Rivest-Shamir-Adleman (RSA)*

L'algoritmo progettato da Rivest, Shamir ed Adleman è il più diffuso fra quelli a chiave pubblica. Può essere utilizzato sia per la cifratura di messaggi che per le firme digitali. Generalmente è considerato sicuro quando si utilizzino chiavi di lunghezza pari o superiore a 1024 bits.

Questo algoritmo si fonda sull'aritmetica delle congruenze, e la sua sicurezza deriva dalla difficoltà di risoluzione del problema della fattorizzazione di numeri “molto grandi” (oltre 100 cifre). Un'eventuale

miglioramento degli algoritmi di fattorizzazione porterebbe immediatamente ad un aumento di vulnerabilità di RSA.

Ecco una breve illustrazione dell'algoritmo. Un utente, per generare la propria coppia di chiavi, si attiene ai seguenti passi:

1. sceglie due numeri primi molto grandi, $p$ e $q$ e calcola il loro prodotto $n = p \times q$
2. sceglie un intero $e$ relativamente primo con $m = (p-1) \times (q-1)$
3. calcola l'intero $d = e^{-1} \bmod m$, costituente la *chiave segreta*
4. "butta via" $n$ e non rivela a nessuno gli interi $p$ e $q$
5. rende la coppia $(e,n)$ pubblica (essa costituisce la *chiave pubblica*)

Un utente, per recapitare un messaggio riservato, compie le seguenti azioni:

- si procura la chiave pubblica del destinatario $K = ( e , n )$
- suddivide il flusso di bit in ingresso in stringhe di bit tali che gli interi $\{ m_k \}$ da esse rappresentati risultino minori od uguali ad $n$
- il $k$-esimo intero $m_k$ viene cifrato nell'intero

$$c_k = m_k^e \bmod n$$

e può essere decifrato dal destinatario con l'operazione

$$c_k^{\,d} \bmod n = m_k$$

Si noti che il calcolo della chiave privata $d$ viene effettuato, mediante l'applicazione dell'algoritmo esteso di Euclide, conoscendo $m$.

Un utente qualsiasi, invece, conosce soltanto $n$ ed $e$, e non può calcolare $d = e^{-1} \bmod m$. Si congettura, ma non è mai stato dimostrato da alcuno, che l'unica possibilità di attacco all'algoritmo sia la fattorizzazione di $n$, che permetterebbe il calcolo di $m$ e quindi di $d$. Fortunatamente la fattorizzazione di un intero è un problema di complessità esponenziale, quindi, scegliendo una lunghezza di $n$ sufficiente, si rende il calcolo impraticabile.

Comunque non è escluso che possa esistere una procedura alternativa di complessità non esponenziale che, a partire dalla chiave pubblica, riesca a decifrare il testo cifrato.

È possibile usare *RSA* per applicazioni di firma digitale: la firma è ottenibile con il calcolo $c_k = m_k^{\,d} \bmod n$ , utilizzando la chiave privata, mentre per la verifica si effettua l'operazione inversa $c_k^{\,e} \bmod n = m_k$ , utilizzando la chiave pubblica.

*RSA è* molto vulnerabile ad attacchi a scelta di testo in chiaro, infatti un utente malizioso potrebbe compiere i seguenti passi:

1. procurarsi la chiave pubblica del destinatario $K = (e,n)$

2. intercettare e registrare un messaggio cifrato

$$c = m^e \bmod n$$

3. scegliere un numero casuale $r < n$, e calcolare

$$y = (r^e c) \bmod n$$

4. chiedere al destinatario di decifrare $y$, ottenendo

$$u = y^d \bmod n$$

5. recuperare il messaggio in chiaro mediante il calcolo

$$r^{-1} u \bmod n = m$$

Nel caso in cui si usi RSA per la firma digitale, un utente malizioso potrebbe farsi firmare un messaggio qualsiasi $m'$ mediante il seguente attacco:

1. genera un numero casuale $x$, e calcola $y = x^e \bmod n$

2. calcola il messaggio $m = y\, m'$

3. richiede la sottoscrizione di $m$, ottenendo $c = m^d \bmod n$

4. calcola la firma di $m'$, mediante il calcolo (modulo $n$)

$$c \times x^{-1} = (x^e)^d \times (m')^d\, x^{-1} = (m')^d$$

Questi sono soltanto alcuni degli attacchi possibili. Comunque

RSA è ritenuto sufficientemente sicuro a condizione che si prendano alcuni accorgimenti:

- non si utilizza RSA per firmare documenti qualsiasi, ma si firma *sempre* l'impronta digitale[26] del documento
- non si condivide un $n$ comune a più utenti
- non si cifrano messaggi troppo corti: ogni blocco elementare dovrebbe avere la stessa lunghezza (in bits) di $n$
- non si utilizzano “piccoli” valori per $d$
- non si utilizza la stessa coppia di chiavi per firmare e cifrare.

Molte implementazioni di RSA sono liberamente disponibili, come ad esempio in RSAREF, RSAEURO, nel codice sorgente di SSLeay, PGP e SSH, nella libreria Crypto++ o in quella di OpenSSL.

RSA è brevettato negli Stati Uniti fino all'anno 2000, mentre è liberamente utilizzabile altrove.

*Rabin*

L'algoritmo progettato da M. O. Rabin, sempre basato sull'aritmetica delle congruenze, fonda la sua sicurezza sulla complessità del problema del calcolo della radice quadrata in modulo, che è computazionalmente equivalente al problema della fattorizzazione di un intero.

In fase di decodifica mediante la chiave privata, in questo algoritmo, bisogna scegliere fra quattro possibili soluzioni. Quindi il messaggio in chiaro dev'essere sufficientemente ridondante da permettere di essere riconosciuto dagli altri. Un modo di garantire ciò è aggiungere una intestazione conosciuta al messaggio prima di cifrarlo.

Rabin può essere utilizzato sia per cifrare che per firmare.

Anche questo algoritmo è violabile mediante attacchi a scelta di testo in chiaro, quindi, nel caso lo si utilizzi per firme digitali, bisogna sempre firmare l'hash del documento, mai il documento in sé.

*ElGamal*

È un altro algoritmo a chiave pubblica, utilizzabile sia per la cifratura che per la firma, basato sulla difficoltà nella risoluzione del problema del logaritmo discreto in un campo finito.

Per generare una coppia di chiavi di firma, un utente:

1. sceglie un numero primo $p$
2. sceglie due numeri casuali $g$ ed $x$, entrambi minori di $p$
3. calcola $y = g^x \bmod p$
4. mantiene $x$ come chiave privata, mentre rivela la terna $(y,g,p)$ come chiave pubblica

[26] Vedere il paragrafo 2.6

Per firmare un messaggio $M$, un utente:

1. sceglie un numero casuale $k$ relativamente primo con $(p-1)$
2. calcola $a = g^k \bmod p$
3. utilizza l'algoritmo esteso di Euclide per trovare un $b$ t.c.

$$M = (x\ a + k\ b) \bmod (p - 1)$$

4. la firma è costituita dalla coppia $(a,b)$, mentre l'intero $k$ dev'essere mantenuto segreto

Per verificare la firma, è sufficiente verificare che

$$y^a\ a^b \bmod p = g^M \bmod p$$

Per cifrare un messaggio $M$, un utente:

1. sceglie un numero casuale $k$ relativamente primo con $p-1$
2. calcola $a = g^k \bmod p$
3. calcola $b = Y^k\ M \bmod p$
4. trasmette la coppia $(a,b)$ come testo cifrato.

Il destinatario recupera $M$ mediante il calcolo $\left(\frac{b}{a}\right)^x \bmod \quad p = M$ .

*Digital Signature Algorithm*

L'algoritmo *DSA*, proposto nel 1991 dal NIST[27] per l'utilizzo nell'ambito dello standard *DSS* (*Digital Signature Standard*), è stato progettato appositamente per l'apposizione di firme digitali.

L'algoritmo utilizza i seguenti parametri:

1. un numero primo $p$ “molto grande”
2. un numero primo $q$ “molto grande” che sia fattore di $(p-1)$
3. un numero $h < (p-1)$ tale che

$$g = \left( h^{\frac{p-1}{q}} \mod p \right) > 1$$

4. una funzione hash a senso unico $H(x)$ (nello standard è prevista SHA)

I parametri $p,q,g$ sono pubblici, mentre la chiave privata è formata da un numero $x < q$ e quella pubblica è calcolata come $y = g^x \bmod p$.

Un utente, per firmare un messaggio $m$:

1. genera un numero casuale $k < q$
2. calcola la firma $(r,s)$:

$$r = (g^k \bmod p) \bmod q$$

[27] National Institute of Standards and Technologies.

$$s = (k^{-1} (H(m) + x^r)) \bmod q$$

Un utente, per verificare una firma:

- calcola $w = s^{-1} \bmod q$
- calcola $u_1 = (H(m) \times w) \bmod q$
- calcola $u_2 = (r \times w) \bmod q$
- calcola $v = \left(\left(g^{u_1} \cdot g^{u_2}\right) \bmod p\right) \bmod q$
- verifica che si abbia $v = r$

Come risulta evidente, la sicurezza di questo algoritmo si basa sulla difficoltà nella risoluzione del problema del logaritmo discreto. Infatti, teoricamente, la chiave privata sarebbe ottenibile da quella pubblica calcolandone il logaritmo discreto, modulo $p$, in base $g$. Naturalmente le lunghezze, in bits, dei numeri in gioco è tale da rendere impraticabile un calcolo del genere ($p$ va dai 512 ai 1024 bits).

Un'altra caratteristica di DSA è la possibilità, scoperta da G. Simmons, di nascondere in firme DSA un *canale subliminale* che permetta di far transitare dati segreti.

L'algoritmo DSA è brevettato in tutto il mondo fino all'anno 2008, comunque non è molto utilizzato per l'esistenza di algoritmi ritenuti più sicuri.

*Considerazioni finali*

Nelle sezioni precedenti abbiamo preso in rassegna i principali algoritmi crittografici utilizzati fino ad oggi. Si osservi che, fondamentalmente, tutti gli schemi a chiave pubblica affondano le basi della loro sicurezza nella difficoltà di risoluzione di un problema *NP-completo* fra:

- il problema dello zaino
- il problema del logaritmo discreto
- il problema della fattorizzazione

La chiave privata, in questi algoritmi, è un'informazione supplementare che permette la risoluzione di tali problemi con una complessità notevolmente inferiore a quella normalmente richiesta da chi possiede solamente la chiave pubblica.

Non si può dimenticare, però, che *nessuno* è mai riuscito, finora, a dimostrare matematicamente l'inesistenza di algoritmi di complessità inferiore nella risoluzione di tali problemi. Tutt'oggi, quindi, la fiducia negli algoritmi crittografici non deriva da considerazioni logico matematiche, ma da una "congettura", fondata sul fatto che crittoanalisti e matematici di tutto il mondo, in decenni, non hanno ancora trovato algoritmi di complessità inferiore.

In definitiva, quindi, la crittografia è ancora in attesa di una scoperta sensazionale che dimostri inequivocabilmente la non

equivalenza della classe dei problemi *NP-completi* con quella dei problemi *polinomiali*. Non si dimentichi, però, che un'eventuale (ma improbabile) dimostrazione dell'equivalenza delle due classi di problemi porterebbe ad un'inevitabile crollo della crittografia stessa.

Comunque non si può ignorare che la complessità di risoluzione di un problema matematico non implica necessariamente la sicurezza di un algoritmo che ci si appoggia. Infatti la teoria della complessità, usualmente, considera singole isolate istanze di un problema, mentre invece un crittoanalista, tipicamente, ha a disposizione un grande insieme di problemi correlati (molti testi cifrati tutti con la stessa chiave) da risolvere. Non a caso esiste una branca specifica della crittoanalisi, la **crittoanalisi differenziale**, che si propone di attaccare un algoritmo crittografico esaminando *non* un singolo messaggio, ma un *insieme di più* messaggi, tutti cifrati con la stessa chiave, allo scopo di dedurre informazioni sulla chiave o sui messaggi stessi.

Un ulteriore elemento da non trascurare è il fatto che per il calcolo teorico della complessità di un algoritmo si prende sempre in considerazione il **worst case**. Nella realtà, invece, un crittoanalista si trova di fronte a dei casi particolari, specifici, che potrebbero prestarsi a soluzioni alternative di complessità inferiore (si pensi, ad esempio, alle varie debolezze esistenti in quasi tutti gli algoritmi crittografici nei

casi di valori particolari dei parametri o delle chiavi[28]).

*Cifratori basati su funzioni hash*

Esistono, infine, tutta una serie di cifratori simmetrici ottenibili utilizzando come *core* una funzione hash crittograficamente sicura. Il valore hash viene posto in *xor* con un blocco di dati d'ingresso di pari dimensione, poi si ricalcola un nuovo valore applicando la funzione hash ad un'opportuna combinazione del vecchio valore hash, numeri sequenziali e testo in chiaro precedente.

Un esempio di cifratore così ottenuto è *l'MDC/SHA*, implementato, ad esempio, nella libreria Crypto++ o in quella di OpenSSL.

## 3.3 Funzioni Hash crittograficamente sicure

*MD5*

Il Message Digest 5 è un algoritmo sviluppato dal RSA Data Secutiry, Inc, che può essere utilizzato per calcolare il valore hash (a 128 bits) di stringhe di bit arbitrariamente lunghe.

MD5 è implementato, ad esempio, in PGP, SSLeay, RSAREF, SSH e Crypto++.

[28] Per l'algoritmo DES, ad esempio, esistono due particolari valori della chiave che fanno degenerare le funzioni di cifratura e decifratura nella *funzione identità*.

*SHA*

Secure Hash Algorithm, detto anche Secure Hash Standard (SHS), è un algoritmo pubblicato dal Governo degli Stati Uniti. Produce un valore hash a 160 bits, è considerato abbastanza sicuro, ed è disponibile in molte librerie crittografiche.

## 3.4 Generatori di numeri pseudo casuali

Tali algoritmi sono utilizzati per la generazione di chiavi, quindi la loro qualità è un punto critico per la sicurezza del sistema risultante. Il generatore di numeri casuali rischia di diventare il tallone d'Achille di tutto un sistema crittografico, se non progettato accuratamente.

Tipicamente, un generatore di numeri pseudo-casuali si comporta come una macchina a stati, in cui lo **stato interno** in un determinato istante caratterizza (quasi) completamente la sequenza di numeri che verranno generati successivamente. Infatti ogni numero, nella sequenza, è generato in dipendenza da:

- lo stato interno del generatore
- il numero generato per ultimo, o la sequenza degli ultimi $K$ numeri generati (questo punto è incluso nel precedente, in quanto tale sequenza sarebbe parte dello stato interno stesso)
- tutta una serie di parametri che possono essere o meno prevedibili, ad esempio:

- l’istante esatto in cui si richiede di generare un numero
- bytes progressivamente letti da un grande file contenente dati registrati da una sorgente “casuale” o pseudo random
- altri parametri considerati a contenuto “casuale”, come ad esempio il PID del processo in esecuzione sotto un sistema UNIX o UNIX-like

Allora la conoscenza dello stato interno potrebbe essere sufficiente per la costruzione di un generatore “parallelo”, in grado di prevedere *con certezza* quali saranno i prossimi numeri generati. Ciò, insieme alla conoscenza dell’algoritmo di generazione delle chiavi, porterebbe alla previsione sicura delle chiavi generate in futuro su di un sistema.

È opportuno osservare che esistono alcune implementazioni di generatori di numeri pseudo casuali in cui il numero generato in un istante coincide con il valore dello stato interno del generatore stesso. È questo il caso, ad esempio, dei classici generatori di numeri fondati sull’aritmetica in modulo, in cui ad ogni nuova richiesta si moltiplica lo stato interno per un valore fisso e lo si restituisce in uscita modulo un numero primo prefissato. Se il numero in uscita è sullo stesso numero di bit su cui si eseguono i conti internamente, nel generatore, allora si dà la possibilità al richiedente di prevedere a

priori l'intera sequenza dei numeri successivi. Anche nel caso in cui si estraesse una sottostringa di bit dallo stato interno, si avrebbe un "baco" di sicurezza, perché comunque si fornirebbero informazioni preziose sulle possibili direzioni che il generatore può prendere in futuro.

Questo problema risulta di fondamentale importanza se si pensa che su alcuni sistemi esistono generatori di numeri pseudo-random "di sistema", cioè incorporati nel S.O. Il cedere alla tentazione di usare una chiamata di sistema anziché scrivere un codice specifico "ex novo", durante l'implementazione di un algoritmo di generazione di chiavi, potrebbe risultare in un security-flaw dalle conseguenza incalcolabili. Ad esempio, un utente "malizioso" con accesso al sistema come utente non privilegiato, chiedendo al sistema di generare un numero casuale, potrebbe carpire lo stato interno del generatore di sistema, arrivando a prevedere il valore esatto di una chiave generata subito dopo dal sistema per conto di un altro utente.

Naturalmente è necessario prendere delle precauzioni affinché nessuno possa carpire la benché minima informazione sullo stato interno di un generatore crittograficamente sicuro.

Per questo in molte implementazioni si cercano soluzioni in cui la sequenza generata dipenda anche da fattori *esterni* al sistema stesso su cui l'algoritmo gira, o comunque non prevedibili a priori.

La soluzione più sofisticata consiste nel dotare i calcolatori di *hardware dedicato* per la generazione di segnali "casuali", in cui l'infallibilità dei circuiti logici si fonde con l'imprevedibilità di alcuni fenomeni fisici. Tali dispositivi, ad esempio, campionano i segnali provenienti da opportuni sensori in grado di misurare le correnti di perdita di diodi o transistor, il rumore di fondo di sistemi di acquisizione audio, o qualsiasi altro fenomeno fisico intrinsecamente "rumoroso".

Una soluzione un po' meno sofisticata, ma comunque efficace, è quella praticabile nei casi in cui, durante la generazione della chiave, il sistema ha a disposizione un utente davanti ad un terminale: chiedendo all'utente di battere "a caso" dei tasti, il sistema può registrare i tempi che intercorrono fra la pressione dei vari tasti, ottenendo un'ulteriore fonte di casualità con cui effettuare il "seeding" del generatore di numeri casuali. Un elaboratore è in grado di misurare gli intertempi con precisione superiore al millisecondo, con il risultato che anche una battitura con cadenza regolare risulterebbe, in realtà, estremamente "random".

Alcuni sistemi utilizzano un file (opportunamente protetto), che viene continuamente aggiornato con dati estratti dall'interazione del sistema con tutti gli utenti: sorgenti di bit "casuali" possono essere, allora, tutti i segnali provenienti da tutte le periferiche interattive per l'utente: mouse, tastiera, trackball, ecc… I dati numerici forniti da

queste periferiche vengono continuamente rimescolati al contenuto del file protetto, predisponendo per il generatore pseudo-random dati sempre diversi con cui effettuare l'inizializzazione ed il calcolo dei successivi numeri.

# Capitolo 4. Protocolli crittografici

## 4.1 Introduzione ai protocolli crittografici

Abbiamo introdotto gli algoritmi crittografici di base che permettono di avere uno scambio di informazioni sicuro fra entità su di una rete aperta. Ma avere a disposizione tali algoritmi non è sufficiente: bisogna che le entità utilizzino tali algoritmi secondo ben precisi *protocolli*, altrimenti si corre il rischio di non ottenere il livello di sicurezza desiderato.

Un **protocollo** è una serie di passi, coinvolgenti due o più entità, progettato per realizzare un compito[29].

Un **protocollo crittografico** è un protocollo che richiede alle entità partecipanti di ricorrere ad algoritmi crittografici per lo svolgimento di uno o più dei passi previsti.

[29] Bruce Schneier, "*Applied Cryptography Second Edition : protocols, algorithms, and source code in C*", John Wiley & Sons, Inc.

## 4.2 Tipologie di protocolli crittografici

Una prima classificazione dei protocolli crittografici esistenti può essere fatta in base allo *scopo* del protocollo:

- protocolli per lo scambio sicuro di chiavi
- protocolli per la trasmissione di dati riservati fra due o più entità
- protocolli per l'autenticazione di entità remote
- protocolli per la cooperazione di due o più entità reciprocamente diffidenti allo scopo di estrarre numeri "casuali"
- protocolli per lo svolgimento di transazioni elettroniche sicure
- protocolli per lo svolgimento di votazioni sicure
- protocolli per l'apposizione di marche temporali a serie di dati

I protocolli per lo scambio riservato di dati possono utilizzare sia la crittografia a chiave simmetrica, sia quella a chiave pubblica. Nel primo caso le due entità remote condividono una chiave crittografica simmetrica $K$, e semplicemente comunicano cifrando ogni messaggio prima della trasmissione, e decifrando ogni messaggio ricevuto. Eventuali alterazioni del messaggio durante la trasmissione sarebbero rilevate dopo la decifrazione, perché il messaggio ottenuto risulterebbe privo di significato, cioè non rispetterebbe il formato

previsto. Chiaramente si suppone che il messaggio abbia un grado di *ridondanza* sufficiente a permettere di distinguerlo da una sequenza "casuale" di bit[30]. Tale protocollo ha bisogno di un protocollo preliminare che permetta alle due entità di scambiarsi la chiave in modo sicuro.

Utilizzando la crittografia a chiave pubblica è possibile trasmettere un messaggio riservato cifrandolo con la chiave pubblica di cifratura dell'entità di destinazione, che utilizzerà la corrispondente chiave privata per decifrarlo. Anche qui si suppone che il destinatario del messaggio sia in grado di distinguere un messaggio decifrato "corretto" da una stringa casuale, che si otterrebbe se ci fossero delle alterazioni al messaggio cifrato durante la trasmissione. Anche in questo caso è necessario un protocollo per lo scambio di chiavi che si svolga preliminarmente, in modo da far giungere in modo sicuro la chiave pubblica dell'entità destinataria all'entità mittente.

Bisogna notare che, nella pratica, la prima tecnica è quella preferita per lo scambio di dati sicuro fra due entità. Ciò è dovuto alla maggior velocità di elaborazione dei protocolli a chiave simmetrica rispetto a quelli a chiave asimmetrica, a parità di dati cifrati. Quindi la seconda tecnica viene adoperata essenzialmente quando il messaggio da recapitare è "corto", come avviene tipicamente nei **protocolli per lo scambio di chiavi**.

[30] Si ricordi che alterando un singolo bit di un messaggio cifrato, dopo la decifrazione si

Tali protocolli vengono adoperati o per ottenere la condivisione di una chiave simmetrica fra due o più entità, o per distribuire in modo sicuro la chiave pubblica di un'entità agli utenti che la richiedono.

Nel primo caso, la chiave scambiata, detta anche **chiave di sessione** perché tipicamente utilizzata per una singola sessione di comunicazione, permette alle entità di stabilire una connessione riservata utilizzabile per scambiarsi dati senza rischiare di rilevare tali dati a terzi.

## 4.3 Protocolli arbitrati

Una categoria particolare di protocolli è quella dei **protocolli arbitrati**, in cui una delle entità che vi prendono parte è una terza parte disinteressata, detta **arbitro**, su cui le altre parti possono far affidamento. L'arbitro è *una parte disinteressata* nel senso che non ha alcun interesse nel protocollo in cui partecipa, e *non ha alleanze* con alcuna delle entità coinvolte, quindi si suppone che esso rispetti sempre ed in ogni circostanza il protocollo stesso"[31].

Inoltre l'arbitro è sempre nelle condizioni di comunicare in modo sicuro con le altre entità, quindi, ad esempio, per spedire un messaggio in modo sicuro sarà sufficiente recapitarlo all'arbitro, che a sua volta lo recapiterà al destinatario, agli occhi del quale è l'arbitro

---

ottiene l'alterazione di circa metà dei bit del messaggio, distribuiti "casualmente" in esso.

[31] Bruce Schneier, "*Applied Cryptography Second Edition : protocols, algorithms, and source code in C*", John Wiley & Sons, Inc.

stesso che garantisce l'autenticità del messaggio.

In un protocollo di questo tipo l'arbitro deve esplicare le sue funzioni “on line”, *durante* lo svolgimento del protocollo stesso fra le entità interessate.

Per determinate categorie di applicazioni, che richiedono lo svolgimento di un gran numero di protocolli crittografici fra entità, una soluzione del genere sarebbe improponibile per il sovraccarico cui incorrerebbe il server dell'arbitro. Per questo sono stati sviluppati altri tipi di protocollo adatti per quelle situazioni in cui non sia necessario verificare subito che il tutto avvenga correttamente. In questi protocolli, detti protocolli **“adjudicated”**, si richiede l'intervento di una terza parte fidata soltanto quando insorgono sospetti o diffidenze nell'affidabilità delle parti o dei canali di comunicazione. Tali protocolli permettono, in queste situazioni, di ricorrere alla terza parte, detta “giudice”, per stabilire se un messaggio è stato effettivamente trasmesso da una certa entità, o se ci sono stati altri tipi di falsificazione dei messaggi.

## 4.4 Il ruolo dell'arbitro: un esempio pratico

Per chiarire quali possono essere le funzioni di un arbitro in un protocollo crittografico, si riconsideri momentaneamente il problema della distribuzione di una chiave simmetrica fra due entità: $E_1$ vuole stabilire una comunicazione sicura con $E_2$, utilizzando una chiave

simmetrica di cifratura $K$, che ha appena generato. Se $E_1$ spedisse direttamente $K$ ad $E_2$, un'entità intermedia potrebbe intercettare la comunicazione, ed ottenere la chiave.

Invece $E_1$ deve spedire la chiave all'arbitro $A$, indicandogli di voler comunicare con $E_2$, e $A$ recapiterà $K$ ad $E_2$. Il punto, qui, è che le comunicazioni fra entità ed arbitro sono sicure. Ciò può essere garantito, ad esempio, dalla condivisione da parte di ciascuna entità $E_i$ di un chiave simmetrica di cifratura $K_i$ con l'arbitro.

Il protocollo è quindi il seguente:

1. $E_1$ prepara un messaggio $M$ contenente $K$ ed il nome del destinatario $E_2$
2. $E_1$ cifra $M$ con la chiave $K_1$, condivisa con l'arbitro
3. $E_1$ spedisce il messaggio cifrato ad $A$
4. L'arbitro decifra il tutto, recupera $K$, e confeziona un messaggio $M'$ contenente $K$ ed il nome del mittente $E_1$
5. L'arbitro cifra $M'$ con la chiave $K_2$ condivisa con $E_2$, e spedisce il tutto ad $E_2$
6. $E_2$ decifra il messaggio, ed ottiene la chiave $K$ e la consapevolezza che $K$ è adesso una chiave segreta nota

soltanto a lui ed $E_1$[32]

7. $E_2$ ed $E_1$ comunicano in modo sicuro utilizzando $K$ per cifrare e decifrare tutti i messaggi

La critica immediatamente opponibile all'esempio è che non si è risolto affatto il problema della distribuzione delle chiavi: infatti ogni entità $E_i$ deve preliminarmente possedere una chiave segreta $K_i$ condivisa con l'arbitro, il ché ripropone esattamente lo stesso problema.

Comunque, ad un'osservazione più attenta, si può notare che il problema è stato notevolmente semplificato. Sic consideri, infatti, una comunità di $n$ utenti, nel caso peggiore in cui ogni entità voglia comunicare con tutte le altre, e supponiamo che l'unico mezzo di trasmissione sicura di una chiave sia la spedizione tramite corriere.

In assenza dell'arbitro sono necessarie $\frac{n(n-1)}{2}$ spedizioni sicure, mentre in sua presenza ne sono sufficienti $n$.

## 4.5 Tipologie di attacchi ad un protocollo

Un attacco crittografico può essere diretto contro[33]

- gli algoritmi crittografici utilizzati nei protocolli

[32] In realtà anche l'arbitro conosce la chiave, ma le ipotesi su enunciate sull'arbitro ne garantiscono l'affidabilità.

- le particolari implementazioni degli algoritmi e dei protocolli
- i protocolli stessi

Il primo tipo di attacchi è già stato trattato nella sezione sulla crittoanalisi a pagina 36, mentre il secondo tipo esula dagli argomenti trattati in questo lavoro di tesi.

Rimangono gli attacchi a protocolli, che possono essere suddivisi in due grandi categorie:

1. **attacchi passivi**: una o più entità non previste dal protocollo osserva e/o registra i messaggi scambiati, ma non può intervenire o modificare lo svolgimento del protocollo
2. **attacchi attivi**: una o più entità non previste dal protocollo, oltre a leggere i messaggi scambiati, ha anche la facoltà di modificarli, sopprimerli e di inserirne di nuovi

Come verrà evidenziato nel prossimo paragrafo, se in una rete si riuscisse a garantire l'impossibilità di attacchi attivi, il problema della sicurezza dei dati in transito sarebbe drasticamente semplificato. Purtroppo questa situazione può verificarsi esclusivamente in realtà piccole, locali ad uno o pochi laboratori, ove un controllo fisico sulle macchine potrebbe essere sufficiente ad offrire tali garanzie.

Altrettanto non può dirsi per le grandi reti *aperte*, come l'Internet,

---

[33] Bruce Schneier, "*Applied Cryptography Second Edition : protocols, algorithms, and source code in C*", John Wiley & Sons, Inc.

in cui ogni messaggio può attraversare centinaia di elaboratori prima di giungere a destinazione, ed inoltre la presenza di algoritmi di routing dinamico dei pacchetti rende addirittura impossibile prevedere su quali macchine un messaggio transiterà.

In tali reti la minaccia di attacchi attivi è sempre presente, e ciò rende necessaria l'adozione di opportune e sofisticate precauzioni.

## 4.6 Utilizzo della crittografia a chiave pubblica per la distribuzione di chiavi

Le tecniche crittografiche a chiave pubblica permettono di semplificare ulteriormente il problema della distribuzione di chiavi di sessione. Supponiamo infatti che un'entità $E1$ voglia comunicare in modo sicuro con un'entità $E_2$. Nell'ipotesi che la chiave pubblica di cifratura di $E_2$ sia nota ad $E_1$, il seguente protocollo realizza lo scambio sicuro di una chiave di sessione:

1. $E_1$ genera una chiave di sessione $K$ e prepara un messaggio contenente $K$ ed il nome di $E_1$
2. $E_1$ cifra il messaggio con la chiave pubblica di cifratura di $E_2$
3. $E_1$ spedisce il messaggio ad $E_2$
4. $E_2$ decifra il messaggio con la propria chiave privata ed ottiene la chiave $K$

Naturalmente, affinché tutto funzioni correttamente, è necessario che le chiavi pubbliche di cifratura delle entità comunicanti siano distribuite preliminarmente ed in modo sicuro.

Si osservi dapprima che, se la rete è soggetta soltanto ad attacchi *passivi*, è sufficiente che $E_1$ richieda ad $E_2$ la sua chiave pubblica, e che quest'ultima gliela spedisca senza adottare alcuna precauzione. Una terza entità $E_M$ potrebbe eventualmente intercettare e leggere il messaggio, ottenendo anch'essa la chiave pubblica di $E_2$. Ciò non comprometterebbe la sicurezza del protocollo su esposto, in quanto $E_M$ rimarrebbe comunque all'oscuro della chiave privata necessaria alla decifrazione della chiave di sessione $K$ che $E_1$ aveva spedito.

Se, invece, la rete è soggetta anche ad attacchi *attivi*, le cose cambiano drasticamente: $E_M$, dopo aver intercettato il messaggio di richiesta di chiave pubblica da parte di $E_1$, ha due possibilità:

- sopprimere tale messaggio di richiesta, e sostituirsi completamente ad $E_2$, rispondendo con la *propria* chiave pubblica di cifratura $K_M$, facendola passare per quella di $E_2$, *oppure:*
- intercettare e modificare il messaggio di risposta da parte di $E_2$, sostituendo la propria chiave pubblica di cifratura a quella di $E_2$ contenuta nel messaggio

In entrambi i casi $E_1$, al passo 2, cifra utilizza inconsapevolmente la chiave di sessione per cifrare $K_M$, ciò che permette ad $E_M$, una volta intercettato il messaggio, di decifrarlo ed ottenere $K$.

Nel primo caso, inoltre, $E_2$ non si accorge assolutamente di nulla, ed $E_1$ continua a comunicare con $E_M$ credendo di comunicare con $E_2$. Nel secondo caso, invece, $E_M$ può realizzare un attacco del tipo **man in the middle**, interponendosi nella comunicazione fra $E_1$ ed $E_2$. Infatti è sufficiente che, intercettato il messaggio contenente la chiave di sessione, $E_M$ lo cifri nuovamente con la chiave pubblica di $E_2$, recapitandolo al destinatario. A questo punto $E_1$ ed $E_2$ penserebbero di comunicare su di un canale sicuro, mentre invece $E_M$, oltre a leggere e decifrare ogni messaggio cifrato con $K$, avrebbe addirittura la possibilità di modificarlo a piacimento.

Come può avvenire, allora, il recapito delle chiavi pubbliche di cifratura in una rete aperta?

Come si vedrà nel capitolo successivo, è sufficiente che le chiavi pubbliche vengano firmate da una terza parte (**autorità di certificazione**), in opportuni documenti che attestino l’appartenenza di una chiave pubblica ad un'entità. Tali documenti possono essere distribuiti pubblicamente senza alcuna misura di sicurezza, perché la firma digitale ad essi apposta ne garantisce l'autenticità, a patto che

ogni entità sia in grado di verificarla.

Ed ecco che ancora una volta si ripropone il problema: la chiave pubblica per la verifica di tali firme va distribuita in modo sicuro. Questa è l'unica distribuzione per cui bisogna ricorrere a metodi alternativi intrinsecamente sicuri (corrieri, ad esempio). Il numero di spedizioni fidate necessario, comunque, rimane lineare con il numero di utenti, non quadratico, ed il problema della distribuzione di chiavi risulta drasticamente semplificato. Questa operazione di semplificazione assume la massima importanza alla luce di una futura globalizzazione dei servizi per la security, in cui si avranno decine di milioni di utenti serviti da una stessa infrastruttura.

# Capitolo 5. Infrastrutture a chiave pubblica

## 5.1 Introduzione

Si è visto come le tecniche di crittografia moderna mettano a disposizione gli strumenti necessari alla realizzazione di tutti quei servizi di sicurezza che tradizionalmente non sono disponibili su dati elaborati e conservati in forma digitale.

In particolare si sono introdotti i concetti fondamentali riguardanti le tecniche di crittografia simmetrica ed asimmetrica, e si è visto come tali tecniche, adoperate opportunamente in *protocolli crittografici*, riescano a garantire riservatezza e integrità dei dati, autenticazione delle parti e non ripudio relativamente a scambi di informazioni su reti aperte.

In ogni caso si è evidenziato come l”utilizzo di tali protocolli si fondi sulla conoscenza preliminare di opportune chiavi “segrete”, cioè note soltanto ad alcune delle entità, e sconosciute a tutte le altre. Ogni entità, prima di poter usufruire dei servizi di sicurezza

disponibili, ha bisogno di ottenere una qualche chiave utilizzando canali di comunicazione sicuri, come ad esempio corrieri fidati.

Nel caso particolare di reti piccole, in cui il pericolo di attacchi attivi è inesistente grazie a controlli effettuati sull'accesso fisico ai calcolatori, si è visto che esiste un protocollo di distribuzione di chiavi, basato sulla crittografia asimmetrica, che permette l'apertura di una connessione sicura senza il possesso di alcun dato preliminare.

Comunque tale soluzione non è assolutamente proponibile nel caso di una rete globale, di respiro nazionale o mondiale, in cui si hanno milioni di elaboratori connessi in rete, controllati da una moltitudine di aziende o enti locali.

Nella prossima sezione verrà introdotto quello che è l'approccio moderno a questa categoria di problemi, consistente nella realizzazione di un'**infrastruttura a chiave pubblica**, in cui un'autorità fidata si preoccupa di identificare ed autenticare preliminarmente tutti gli utenti che vogliano usufruire dell'infrastruttura, ed è poi essa stessa a garantire l'appartenenza di una chiave pubblica ad uno specifico utente, mediante l'emanazione di un **certificato**.

## 5.2 Perché le infrastrutture a chiave pubblica

Affinché un utente possa verificare la firma digitale allegata ad un

documento informatico, è necessario che si procuri la chiave pubblica di firma dell'autore del documento, a partire dal suo *nome*, presente nel documento stesso. Analogamente, affinché un utente possa recapitare un messaggio riservato ad un secondo utente, è necessario che si procuri la chiave pubblica di cifratura del destinatario.

È quindi necessario che le chiavi pubbliche, sia di firma che di cifratura, siano disponibili ad ognuno. Ciò potrebbe essere ottenuto mediante la realizzazione di un database distribuito contenente le associazioni fra nomi degli utenti e rispettive chiavi. Comunque, dato che ciò che serve praticamente è soltanto la ricerca di una chiave a partire dal *nome* dell'utente, è sufficiente adottare un **Directory Service** distribuito, che è un software molto meno complesso rispetto ad un database. Un tale software permette di reperire informazioni di qualsiasi genere riguardanti un'entità fornendo il nome dell'entità stessa.

Supponendo, quindi, di avere un meccanismo del genere, ci troviamo ancora di fronte ad un problema di sicurezza: anche supponendo che il directory sia ben protetto contro eventuali attacchi provenienti dalla rete, comunque la chiave pubblica richiesta dovrà viaggiare su di un canale insicuro.

Si risolve completamente il problema se l'autorità di certificazione memorizza nel directory l'associazione fra nome dell'entità e

rispettiva chiave pubblica in un documento firmato digitalmente, chiamato **certificato**. Infatti, in questo modo, non è necessario adottare alcuna precauzione per la trasmissione di una tale documento all'utente che lo richiede, perché quest'ultimo *verifica la firma* della autorità di certificazione *prima* di utilizzare la chiave contenuta nel certificato. Inoltre il directory, pur essendo distribuito, non ha bisogno di adottare particolari misure di sicurezza per i suoi protocolli interni, perché ogni eventuale alterazione dei certificati in esso contenuti sarebbe rilevata dall'utente finale, o dal software di gestione del directory stesso.

In verità rimane ancora un problema: l'utente, per verificare la firma digitale dell'AC, ha bisogno della rispettiva chiave pubblica di firma (detta anche **chiave di certificazione**). Tale chiave va necessariamente distribuita in modo sicuro mediante mezzi alternativi. Ad esempio si potrebbe consegnare tale chiave all'utente nel momento in cui egli si registra recandosi fisicamente alla sede più vicina della AC.

Un **certificato** è un documento attestante che una specifica chiave pubblica appartiene ad uno specifico utente, nel senso che quell'utente è l'unico a possedere la chiave privata corrispondente. Tale garanzia è certificata dalla firma digitale di un'**autorità di certificazione**, che è una terza parte disinteressata di cui tutti gli utenti si fidano.

Ogni utente deve consegnare in modo sicuro la propria chiave pubblica alla AC, la quale, dopo opportuni controlli, rilascerà un certificato di chiave pubblica per l'utente, rendendolo disponibile in un deposito pubblico di certificati.

Le comunicazioni sicure preliminari necessarie al corretto funzionamento dell'infrastruttura avvengono durante la fase di **registrazione**. L'utente si recherà fisicamente presso la sede più vicina dell'AC che fornisce i servizi di sicurezza di cui ha bisogno, fornendo i propri dati identificativi. In questa fase l'AC potrà consegnare all'utente la propria chiave pubblica di certificazione, mentre l'utente potrà consegnare alla AC la chiave pubblica che intende farsi certificare.

Nel caso di una piccola comunità di utenti sarebbe sufficiente una singola autorità di certificazione, e quindi ogni utente potrebbe verificare senza problemi la firma elettronica apposta dall'AC sui certificati di chiave pubblica di altri utenti, perché sarebbe sempre in possesso della chiave di verifica della firma del certificato. Nel caso più realistico, oggigiorno, di comunità globali, una tale soluzione è improponibile. Allora occorre che ci siano più autorità di certificazione, ognuna locale ad una piccola comunità di utenti, opportunamente collegate fra loro.

## 5.3 Tipologie di collegamenti fra autorità di certificazione

Esistono due modelli fondamentali di collegamento fra AC:

- **il modello reticolare**, in cui ogni AC nasce come autorità locale, di cui si fidano tutti e soli gli utenti appartenenti ad essa (Figura 5.1)
- **il modello ad albero**, in cui esiste una AC globale, di cui tutti si fidano, che distribuisce opportunamente le proprie mansioni ad AC locali, ognuna facente capo ad una comunità ristretta di utenti o ad altre AC ancora più localizzate (Figura 5.2).

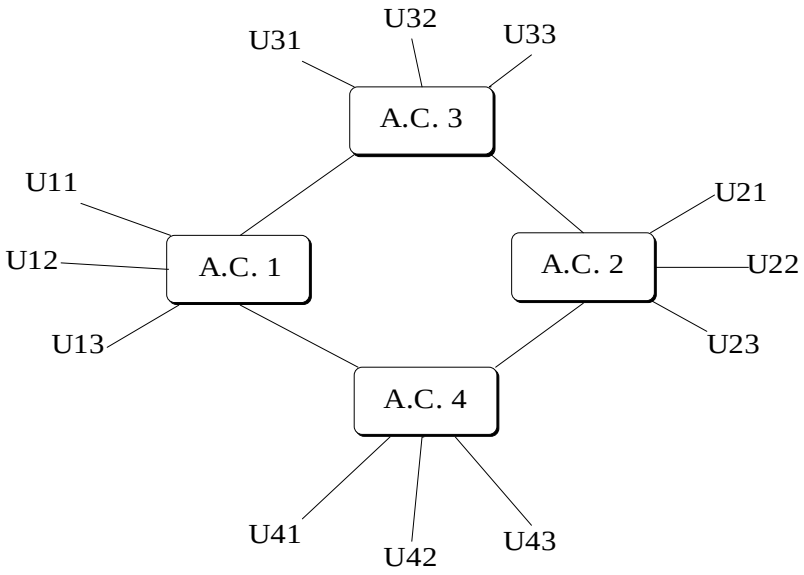


*Figura 5.1. PKI strutturata secondo il modello reticolare.*

*Modello reticolare*

In tal caso l'interoperabilità fra AC diverse è ottenuta mediante

accordi fra le AC stesse. Una AC $A_1$, dopo aver verificato che la AC $A_2$ adotta politiche di sicurezza uguali, o comunque che garantiscono un livello di sicurezza almeno pari a quello garantito dalle proprie politiche, rilascia e firma un certificato, detto **cross-certificate**, che:

1. garantisce che una chiave certificata dall'AC $A_2$ è valida
2. contiene la chiave di verifica di certificati dell'AC $A_2$

Tale certificato, una volta reso disponibile a tutti gli utenti appartenenti alla AC $A_1$, permette loro di fidarsi di una chiave certificata dalla AC $A_2$. Il protocollo previsto per la verifica, da parte di un'entità appartenente all'AC $A_1$, di un certificato rilasciato dalla AC $A_2$, è il seguente:

1. $E_1$ estrae, dal certificato da verificare C, il nome dell'AC $A_2$
2. $E_1$ richiede alla propria AC il cross-certificate relativo ad $A_2$
3. $E_1$ verifica la firma del cross certificate, utilizzando la chiave di verifica pubblica di $A_1$, già in suo possesso
4. $E_1$ estrae dal cross-certificate la chiave di verifica pubblica di $A_2$
5. $E_1$ utilizza la chiave appena estratta per verificare la firma di certificazione apposta sul certificato $C$

Nel modello reticolare le AC possono sistemarsi in un grafo orientato, in cui un arco dalla AC $A_1$ alla AC $A_2$ indica che $A_1$ ha rilasciato un cross-certificate verso $A_2$. In tale modello un utente può verificare la firma di certificati rilasciati da una qualsiasi AC che, sul grafo appena descritto, sia raggiungibile[34] dalla propria. Il protocollo generico, che un'entità $E_1$ appartenente ad una AC $A_1$ deve seguire per verificare un certificato $C$ emesso da una AC $A_N$ diversa, è il seguente:

1. $E_1$ estrae da $C$ il nome della AC $A_N$
2. $E_1$ chiede alla propria AC l'indicazione di un cammino orientato $\{ A1, A2, \ldots , A_N \}$ che porti da $A_1$ ad $A_N$
3. $E_1$ chiede alla propria AC il cross-certificate relativo alla AC $A_2$
4. $E_1$ verifica tale certificato con la chiave $K_1$ di $A_1$, già in suo possesso, ed estrae da esso la chiave $K_2$ di $A_2$
5. $E_1$ richiede[35] il cross-certificate rilasciato da $A_2$ verso $A_3$, e lo verifica con la chiave $K_2$
6. $E_1$ procede in modo analogo, ottenendo e verificando un

[34] Cioè esiste, sul grafo, un cammino orientato che porta dalla AC di appartenenza dell'utente in questione, alla AC che ha rilasciato il certificato da verificare.
[35] Il problema del reperimento di certificati in rete verrà approfondito più avanti.

cross-certificate per ogni arco del cammino da $A_1$ ad $A_N$, fino a quando ottiene la chiave $K_N$ di $A_N$, certificata dall'AC che la precede immediatamente sul cammino.

Bisogna notare che questo modello prevede che, agli occhi di un utente, sia sempre la propria AC a garantire l'autenticità delle firme di altre AC, in riga con un modello di distribuzione della fiducia che prevede che ogni entità si fidi soltanto della propria AC di appartenenza.

La situazione ideale, per un modello del genere, si avrebbe se tutte le autorità di certificazione adottassero le stesse politiche sicurezza, o comunque politiche compatibili[36].

Questo potrebbe ottenersi soltanto se ci fosse un'autorità unica che dettasse politiche di sicurezza e standard da rispettare. Ma in tal caso si avrebbe comunque un accentramento dell'infrastruttura, e quindi un'incompatibilità di fondo con quello che è il maggior vantaggio del modello reticolare, cioè avere una rete di AC tutte indipendenti fra loro.

Invece la situazione più realistica prevede che ogni AC adotti politiche di sicurezza proprie, compatibili con la realtà locale in cui i propri utenti si trovano ad operare, e ciò rende la fase di rilascio dei cross-certificate problematica. Infatti è necessario che ciascuna

[36] Nel senso che garantiscono l'espletamento di servizi di sicurezza di pari livello di qualità.

autorità conosca ed analizzi a fondo le politiche della autorità “vicina”, prima del rilascio del certificato.

In generale bisogna considerare che il livello di sicurezza di una AC, quindi l'affidabilità dei certificati da essa rilasciati, risulta determinato almeno dai seguenti fattori:

1. gli algoritmi crittografici impiegati nei processi di firma digitale
2. le politiche di accesso agli elaboratori adottate dall'AC
3. il software utilizzato dalla AC
4. le procedure per l'identificazione ed autenticazione preliminare degli utenti.

Il livello di sicurezza di una AC $A_2$ “vicina”, quindi, potrebbe essere maggiore o minore di quello della AC $A_1$ che deve rilasciare un cross-certificate. Nel primo caso, il cross-certificate dovrebbe indicare che la AC $A_2$ offre almeno le stesse garanzie offerte da $A_1$, mentre nel secondo caso esso dovrebbe contenere un'indicazione di restrizione del livello di sicurezza.

*Modello ad albero*

Nel modello ad albero esiste un'unica autorità di certificazione globale di cui tutti gli utenti dell'infrastruttura informativa si fidano, però ogni utente non interagisce direttamente con essa, ma con altre autorità di certificazione, ciascuna locale ad una comunità ristretta di

utenti.

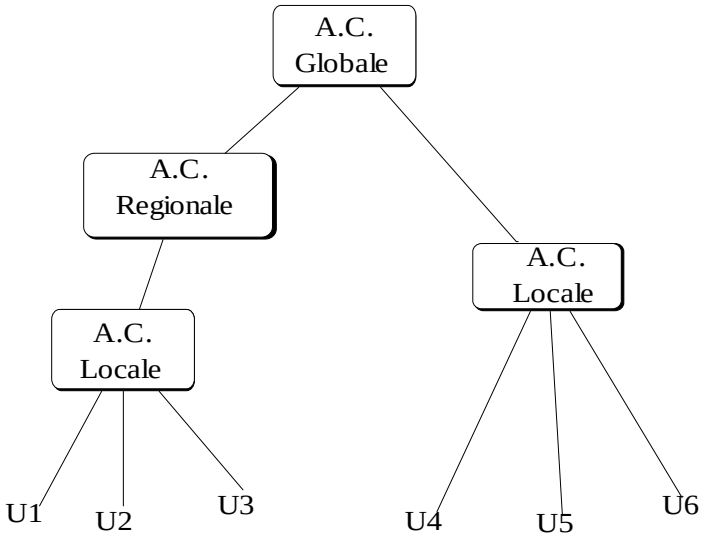


*Figura 5.2. PKI strutturata secondo il modello ad albero.*

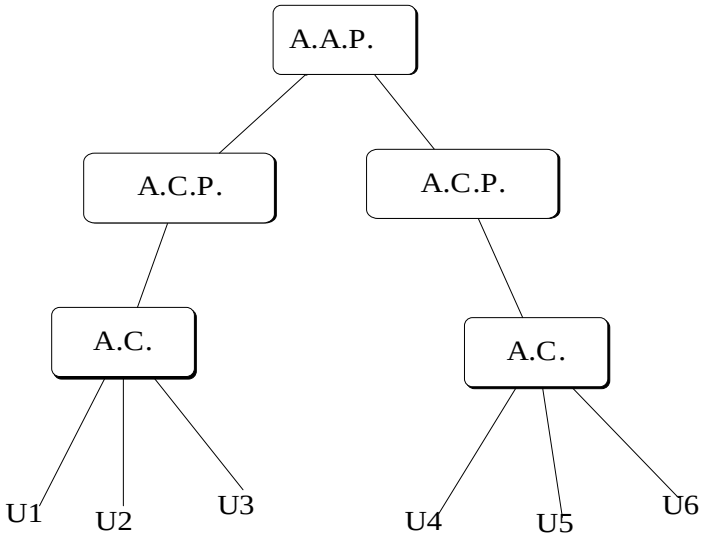


*Figura 5.3. Gerarchia di AC nel modello ad albero.*

Questo modello prevede che le AC siano distribuite

gerarchicamente, secondo una struttura ad albero, in cui ogni livello dell'albero corrisponde ad autorità di certificazione con un ruolo diverso all'interno dell'infrastruttura.

In particolare, in una tipica struttura ad albero, le AC possono essere classificate[37] come (Figura 5.3):

- **Autorità di Approvazione delle Politiche (AAP)[38],** radice dell'albero delle AC, stabilisce le politiche di sicurezza globali dell'infrastruttura e detta linee guida cui tutte le entità subordinate devono attenersi; una AAP è anche una AC, in quanto emette certificati per l'abilitazione delle autorità sottostanti
- **Autorità di Certificazione delle Politiche (ACP)[39],** stabilisce politiche di sicurezza locali ad una singola organizzazione o comunità di interesse; una ACP è anche una AC, in quanto emette certificati per l'abilitazione delle autorità sottostanti
- **Autorità di Certificazione (AC)** propriamente detta, emette certificati di chiave pubblica o di altro tipo[40] per gli utenti che appartengono ad essa
- **Autorità di Registrazione[41] (AR)**, agisce da intermediario fra

[37] Prof. Dr. Hartmut Pohl, "*Guidelines for the use of Names and Keys in a Global TTP Infrastructure*", ISIS Institute for Information Security at Essen, Germany (May 1997).
[38] Policy Approval Authority.
[39] Policy Certification Authority
[40] Vedremo più avanti che esistono diverse tipologie di certificato che un'AC può emettere.
[41] Organisational Registration Authority.

una AC e gli utenti che le appartengono; verifica l'identità degli utenti al momento della loro registrazione presso la AC, ed eventualmente fa ulteriori verifiche a seconda dei certificati richiesti.

Osservando che un albero è anche un grafo orientato, si capisce che la struttura ad albero è un caso particolare di quella più generica a rete. Infatti i certificati che una AC di un certo livello rilascia alle AC ”figlie” sono a tutti gli effetti dei cross-certificate, in quanto hanno esattamente lo stesso ruolo che un tale certificato ha in un'infrastruttura reticolare.

Ciò che però distingue tale modello dal precedente è il fatto che normalmente una AC “padre” rilascia un cross-certificate per ogni AC “figlia”, mentre non è necessario che le AC “figlie” rilascino un cross-certificate per la AC “padre”. Questo perché, essendo la struttura accentrata, è possibile consegnare all'utente finale, in modo sicuro, direttamente la chiave pubblica di certificazione della AC radice della struttura (AAP). Il possesso di tale chiave permette la verifica di un qualsiasi certificato rilasciato da una qualsiasi AC dell'albero, non importa se sia quella di appartenenza dell'utente o meno. Chiaramente ogni utente si troverà sicuramente già in possesso del certificato di chiave pubblica della propria AC, quindi la verifica di un certificato da essa emesso risulterà immediata. Invece, per la verifica di un certificato emesso da un'altra AC bisognerà che l'utente

richieda e verifichi tutti i cross-certificate che portano dalla radice dell'albero alla AC in questione.

I problemi di cooperazione fra AC evidenziati nel modello a rete sono notevolmente attenuati nel modello ad albero. Infatti, anche se ogni AC è relativa ad una organizzazione o comunità particolare, comunque sono sempre le AC dei livelli superiori (AAP e ACP) a dettare le regole cui tutte le AC devono sottostare.

*Modello con struttura ibrida*

In questo caso si hanno più infrastrutture ad albero che, per necessità di cooperazione, si riuniscono in un'unica rete di AC secondo le modalità del modello infrastrutturale a rete. Soltanto le AC di radice devono rilasciarsi mutuamente dei cross-certificate, e questo garantisce l'interoperabilità fra tutte le AC appartenenti agli alberi le cui radici si sono mutuamente certificate.

La situazione finale è simile a quella di una rete di AC, ma con una differenza sostanziale: ogni utente continua a “fidarsi” soltanto della AC radice dell'albero contenente la sua AC “locale” di appartenenza[42].

Quindi, per la verifica di un certificato, sono possibili tre situazioni:

1. il certificato è stato emesso dalla stessa AC di appartenenza

[42] Cioè gli viene recapitata in modo sicuro soltanto la chiave della AC radice dell'albero, nel momento in cui egli si registra presso la sua AC di appartenenza.

dell'utente: occorre procurarsi e verificare tutti i certificati che portano dalla AC di radice alla AC locale, operazione questa che sicuramente è già stata effettuata in precedenza, quindi generalmente l'utente possiede già la chiave pubblica di certificazione dell'AC locale

2. il certificato è stato emesso da una AC $A$ dell'albero di appartenenza dell'utente: occorre procurarsi e verificare tutti i certificati che portano dalla AC radice ad $A$

3. il certificato è stato emesso da una AC $A$ appartenente ad un altro albero: in questo caso bisogna prima procurarsi e verificare il cross certificate che certifica l'A.C. di radice dell'albero di appartenenza di $A$, e successivamente procedere "scendendo" nell'albero fino ad $A$

Una struttura ibrida di questo tipo è quella destinata a formarsi in una realtà di ampiezza continentale o mondiale, in cui ogni stato sviluppa una infrastruttura nazionale gerarchica indipendentemente dagli altri, ma poi, se le varie infrastrutture sono "compatibili" fra loro, una serie di certificazioni mutue riesce a garantire l'interoperabilità fra di esse.

È evidente che cercare di imporre un'infrastruttura globale gerarchica sarebbe un'impresa impossibile, perché ci dovrebbe essere un'unica AAP che detti le regole e le politiche generali per la

sicurezza che dovrebbero essere seguite ovunque. Tale approccio è fortemente ostacolato, tra l'altro, anche dalla presenza, nelle diverse nazioni, di legislazioni fortemente diversificate per quanto riguarda i documenti informatici, la loro conservazione, validazione e protezione. Tale problematica verrà ulteriormente dettagliata nel Capitolo 6.

## 5.4 Modelli di Fiducia (Trust Model)

Nelle sezioni precedenti si è evidenziato come la differenza sostanziale fra un'infrastruttura a rete ed una ad albero risieda nelle modalità con cui l'utente arriva a fidarsi di un certificato emesso da una AC a lui sconosciuta, partendo dalla fiducia che egli ripone in una singola AC a lui nota.

In letteratura l'insieme di tali modalità, e non solo, viene generalmente indicato come **modello di fiducia** *(Trust Model*) dell'infrastruttura. Più precisamente, un modello di fiducia[43] "specifica in che modo si può porre fiducia nell'asserzione che un utente remoto è effettivamente chi dice di essere (autenticazione) e che ha effettivamente i diritti di accesso al servizio o informazione che sta richiedendo (autorizzazione)". Quindi il modello di fiducia si pone alla base di un'infrastruttura a chiave pubblica, specificando quali procedure riescono a garantire il corretto funzionamento dei servizi di

sicurezza supportati dall'infrastruttura stessa.

È estremamente interessante mettere a confronto due modelli di fiducia diametralmente opposti fra loro: quello adottato dal package PGP, e quello previsto in PEM.

*Il modello di fiducia adottato in PGP*

PGP è probabilmente il pacchetto crittografico a chiave pubblica più diffuso nell'Internet oggi. In esso l'utente viene messo “al centro” del modello di fiducia, nel senso che è soltanto lui a decidere in chi porre la propria fiducia o meno. L'idea alla base del modello è quella che soltanto noi stessi siamo in grado di stabilire con esattezza il livello di affidabilità associato alle informazioni che possediamo. Inoltre, esattamente come avviene nel mondo reale, se si conosce un'altra persona quanto basta per ritenere che le informazioni da lei possedute abbiano un livello di affidabilità simile al nostro, allora si possono prendere le informazioni possedute da questa persone per buone.

Praticamente, ogni utente ha la possibilità di configurare la propria copia del software PGP specificando l’insieme delle chiavi pubbliche *affidabili*, detto **portachiavi pubblico**. Esso è fondamentalmente costituito da una serie di coppie chiave-possessore. Ogni utente gestisce personalmente il portachiavi,

[43] D. W. Chadwick, A. J. Young, N. Kapidzic, “*Merging and Extending the PGP and PEM*

aggiungendo ad esso una coppia $(K, U)$ solamente dopo aver accertato di persona l'appartenenza della chiave $K$ all'utente $U$.

Il software prevede, comunque, la possibilità di "esportare" una o più delle associazioni chiave-possessore, presenti nel proprio portachiavi pubblico, in appositi file firmati detti **certificati di chiave**. Questi file possono, quindi, essere distribuiti pubblicamente, permettendo di allargare considerevolmente l'insieme delle chiavi fidate di altri utenti. Infatti un certificato di chiave $(K, U)$, firmato da un utente $U_1$, dice ad un altro utente $U_2$ che il legame fra chiave e possessore in esso contenuto è ritenuto affidabile da $U_1$. Quest'ultimo, con la propria firma, garantisce l'appartenenza di $K$ ad $U$. Spetta comunque ad $U_2$ ponderare il valore di tale garanzia, sulla base della propria conoscenza dell'affidabilità di $U_1$. In definitiva il software permette ad ognuno di specificare, fra gli utenti presenti nel proprio portachiavi, quali sono da considerarsi **presentatori fidati**. I certificati firmati da presentatori fidati vengono considerati affidabili dal software, quindi il legame chiave-possessore in essi contenuto è considerato valido. Al contrario, i certificati firmati da utenti non marcati come presentatori vengono semplicemente ignorati.

Se si ritiene che un utente non è capace di verificare le associazioni chiave-possessore, quindi, è sufficiente non includerlo

---

*Trust Models – The ICE-TEL Trust Model*"

fra i presentatori, ed il software automaticamente scarterà i certificati da lui firmati.

Con questo meccanismo il PGP permette un sostanziale allargamento delle coppie chiave-possessore sicure, infatti ogni utente ha la libertà di includere nel proprio portachiavi pubblico le chiavi certificate da altri utenti ritenuti affidabili. Inoltre egli, firmando una o più delle associazioni del proprio portachiavi di cui è particolarmente sicuro, può dare ad altri utenti che si fidano di lui la possibilità di includere tali associazioni nei loro portachiavi.

Questo meccanismo può essere reiterato quante volte si vuole, costruendo delle vere e proprie “catene di certificazione”, come accade nelle PKI con struttura reticolare.

Il modello di fiducia del PGP, comunque, è in grado di distinguere fra chiavi che l’utente garantisce di persona e chiavi importate con il meccanismo dei certificati firmati, dando alle prime un livello di affidabilità maggiore che alle seconde. Sostanzialmente ogni utente ha la facoltà di decidere il numero massimo di certificati che il software può attraversare nel valutare l’affidabilità di una chiave pubblica. Un livello pari a zero significherebbe che il software si fida soltanto delle chiavi aggiunte personalmente dall’utente. Un livello pari ad uno significa che sono affidabili anche le chiavi firmate dai *presentatori*, e così via.

Da un punto di vista del modello di fiducia, la situazione che si ha in PGP è esattamente un “collassamento” del modello infrastrutturale a rete, in cui ad ogni autorità di certificazione appartiene un unico utente, anzi utente ed AC di appartenenza coincidono, confermando quindi la centralità dell’utente nello stabilire se un’informazione risulta affidabile o meno. L’emissione da parte di un utente di un certificato firmato equivale, nelle PKI reticolari, all’emissione di un cross-certificate da una AC verso un’altra AC.

Probabilmente è proprio grazie all'accentramento del modello intorno al singolo individuo che PGP ha trovato il suo punto di forza: ognuno ha la possibilità di controllare esattamente su quali basi si fonda la propria fiducia in qualcosa. Questo “riflette esattamente il modo in cui gli esseri umani interagiscono fra loro nei normali rapporti sociali”[44].

Inoltre ognuno ha la possibilità di adottare il software senza alcun protocollo preliminare: due utenti possono scambiarsi informazioni riservate semplicemente generando le proprie coppie di chiavi, e scambiandosi reciprocamente le chiavi pubbliche durante un incontro. Chiaramente un tale approccio è destinato a non essere scalabile su grandi comunità di utenti, e questo essenzialmente perché il “raggio“ entro cui resta confinata la fiducia “distribuita” dal singolo individuo non può crescere oltre certi limiti.

È interessante notare come questo modello di fiducia non preclude la possibilità di avere delle vere e proprie Autorità di Certificazione, che in PGP non sono nient'altro che utenti specializzati nella verifica dell'appartenenza di chiavi, di cui quindi ognuno può fidarsi tranquillamente segnalandoli come presentatori fidati.

*Modello di fiducia in PEM*

Agli antipodi di PGP, il modello di fiducia di PEM prevede un'infrastruttura ad albero ove ognuno deve affidarsi completamente ad una AC radice unica, l'Internet Policy Registration Authority (IPRA), e tutto avviene come descritto nel paragrafo a pagina 87 a proposito del modello ad albero. Tale modello supera le difficoltà riscontrate in PGP nello stabilire un "canale di fiducia" con un utente remoto, perché delega ad autorità specifiche il compito di verificare l'affidabilità di ogni singola coppia chiave-utente.

Una volta affidatisi all'IPRA, sarà essa stessa a distribuire la fiducia degli utenti nelle AC sottostanti nell'albero, fino ad arrivare ai singoli utenti.

La controparte negativa di tutto questo risulta in una maggiore complessità nei preliminari all'utilizzo dei servizi di sicurezza: è necessario che ogni utente si registri presso un'AC e si faccia

[44] Philip Zimmermen, "*Manuale d'uso del PGP™*".

rilasciare un certificato di chiave pubblica, quindi è necessario che l'AC effettui tutti i controlli del caso, prima di rilasciare il certificato. Tali procedure, chiaramente, finiscono per aumentare l'*inerzia* che ostacola un utente nell'utilizzo di PEM.

*Modello di fiducia in ICE-TEL*

Nel progetto ICE-TEL è stato proposto un modello di fiducia intermedio fra i due presentati. Infatti in questo modello è sia possibile fidarsi di utenti singoli, come avviene in PGP, sia fidarsi di autorità di certificazione, o addirittura di intere infrastrutture ad albero.

Il trust-model è stato progettato per rispettare i seguenti requisiti[45]:

1. è possibile usufruire dei servizi di sicurezza senza l'uso di certificati o infrastrutture di certificazione
2. è possibile utilizzare i certificati secondo lo standard X.509, versione 3 che, in particolare, aggiunge la possibilità di indicare per quali utilizzi la chiave è destinata
3. è possibile creare dei "domini di sicurezza" costituiti indifferentemente da singoli individui, piccoli gruppi di utenti o addirittura organizzazioni arbitrariamente complesse

[45] D. W. Chadwick, A. J. Young, N. Kapidzic, "*Merging and Extending the PGP and PEM Trust Models – The ICE-TEL Trust Model*"

4. i domini di sicurezza possono arbitrariamente crescere, restringersi o riorganizzarsi in qualsiasi istante creando inconvenienti minimi agli utenti
5. gli utenti e/o l'amministratore di un dominio di sicurezza possono scegliere di quali altri domini fidarsi
6. il funzionamento del trust-model non deve dipendere dall'esistenza di alcuna singola parte dell'infrastruttura, come invece avviene per le infrastrutture centralizzate

L'architettura di ICE-TEL, comunque, prevede la possibilità di avere delle Autorità di Certificazione, che si differenziano dai presentatori fidati di PGP perché pubblicano chiaramente le loro **security policy**, indicanti:

- come l'AC controlla l'identità degli utenti prima di emettere un certificato
- come l'AC protegge la sua chiave privata
- quando, sotto quali circostanze e come un'AC pubblica la sua Lista di Revoca dei Certificati
- quali garanzie l'AC offre e quali responsabilità assume nei confronti dei certificati che emette, e per quali scopi dovrebbero essere usati.

Lo standard X.509 versione 3 prevede che le security policy vengano indicate nei certificati mediante identificatori numerici che

individuano univocamente precisi documenti pubblicati dalle A.C.

L'implementazione di questi concetti è abbastanza semplice: ICE-TEL stabilisce che ogni utente possegga un proprio ambiente locale, detto **Personal Security Environment** *(PSE)*, in cui sono memorizzate:

1. chiavi pubbliche (insieme ai relativi nomi) di utenti fidati
2. chiavi pubbliche di autorità di certificazione di radice (dette anche **Trusted Points**) di infrastrutture ad albero, insieme agli identificatori delle politiche di sicurezza che l'AC adotta
3. gli identificatori delle security policy che l'utente considera per lui appropriate, cioè che, secondo lui, garantiscono un livello di sicurezza accettabile

Quando il software verifica la validità di un certificato, controlla anche che le security policy sotto le quali il certificato è stato emesso coincidano o siano compatibili con quelle che l'utente ha richiesto nel proprio PSE.

Una caratteristica che differenzia il modello di ICE-TEL dagli altri è la possibilità di avere i cosiddetti **embedded security domain**. In questo caso si hanno due domini, uno con un elevato grado di sicurezza, l'altro con un grado di sicurezza inferiore. I due domini sono trattati nel modello non come domini indipendenti, nel qual caso ci sarebbero dovuti essere dei cross-certificate, ma il modello

che offre maggiori garanzie risulta *immerso* nell'altro. Ciò viene ottenuto permettendo agli utenti del dominio ad elevato grado di sicurezza di avere due percorsi di certificazione: il primo comincia dal trusted-point ad alto livello di sicurezza e contiene gli identificatori delle relative politiche (più restrittive), l'altro comincia dal trusted-point a basso livello di sicurezza e contiene gli identificatori delle relative politiche (più permissive). Dato che ad ogni messaggio cifrato o firmato viene accodato il percorso di certificazione, il software sceglie di accodare l'uno o l'altro a seconda del dominio di sicurezza di appartenenza del destinatario del messaggio.

In definitiva, il modello di fiducia previsto nel progetto ICE-TEL risulta essere un buon compromesso fra la struttura altamente centralizzata prevista in PEM e quella opposta che si ha PGP. Non precludendo la possibilità di sviluppo di una PKI ad albero, il modello lascia piena libertà al singolo utente di aderire o meno all'infrastruttura, ed in ogni caso di decidere lui per primo se fidarsi o meno di una chiave pubblica. Inoltre, con l'adozione dei certificati in formato X.509 versione 3, tale modello prevede la possibilità di avere tutti i tipi di certificato che si sono vagliati nel capitolo precedente, permettendo la certificazione di chiavi per l'autenticazione, per la firma, ecc…

# Capitolo 6. Aspetti legali della “Firma Digitale”

## 6.1 Introduzione

Nei capitoli precedenti si è osservato che i nuovi mezzi di comunicazione a disposizione, oggi, hanno rivoluzionato completamente lo stile di vita degli individui. Comunque, ormai, l’interconnessione globale dei calcolatori sull’Internet non costituisce più una novità per gli utilizzatori, che sempre più sentono il bisogno non solo di trasmettersi informazioni in forma digitale, ma anche di creare ed elaborare documenti *direttamente* in forma elettronica. Infatti, tutt’oggi, nonostante l’evoluzione tecnologia, esistono tantissime procedure, soprattutto in ambito legale, che fondano le basi della loro validità legale sull’esistenza di documenti cartacei. È questo il caso, ad esempio, dei contratti di qualsiasi tipo, che ancora oggi devono essere stipulati in forma cartacea, per essere firmati “a mano” dalle parti contraenti. Questo si rende necessario perché il sistema legale, finora, prevedeva ancora che fosse il documento

cartaceo a dimostrare l'avvenuta conclusione di un accordo fra più parti.

In quest'ottica si capisce come le nuove tecnologie informatiche per la firma digitale e l'autenticazione perdano una parte sostanziale della loro utilità, se non adeguatamente supportate da un sistema giuridico in grado di accettarle e di dare piena validità legale alle informazioni trasmesse, duplicate e conservate in forma elettronica.

## 6.2 Documento cartaceo e firma manoscritta

Un documento cartaceo consiste di quattro parti[46], fisicamente legate fra loro in modo inscindibile:

1. il *supporto*, tipicamente un semplice foglio di carta
2. il *contenuto* del documento (testo, figure, ecc...)
3. l'*identificazione* dell'autore del documento
4. qualcosa che assicuri l'*autenticità* del documento

La forma più diffusa per l'autenticazione di un documento è la *sottoscrizione*, cioè l'apposizione, da parte dell'autore, del suo nome e cognome in una firma, utilizzando una penna. La garanzia di autenticità viene dal fatto che ogni individuo, a seconda delle proprie caratteristiche psico-fisiche, firma in un modo unico, difficilmente riproducibile o imitabile da altri.

[46] "*Legal and Regulatory Issues for the European Trusted Services Infrastructure*", ETS.

Quindi una firma scritta ha le seguenti funzioni[46]:

- *identificare* “con certezza” l'autore del documento: la firma fa sì che chiunque legga il documento possa associarne il contenuto al firmatario
- manifestare fisicamente la *volontà* del firmatario ad essere identificato come l'autore del documento: la manifestazione fisica serve come *prova* verso terze parti

Ogni sistema legale dà un certo valore legale alla firma scritta, valore che in ogni caso viene accresciuto se la firma è apposta in presenza di un notaio.

Un aspetto comune in tutti i sistemi giuridici è l'assenza di una regola precisa per le modalità di firma. Ognuno può usare il proprio nome e cognome, o solo le iniziali, o un simbolo particolare. È comunque interesse di ognuno rendere la propria firma sufficientemente “unica”, per evitare il rischio della falsificazione.

Comunque, nella letteratura legale di oggigiorno, è opinione comune che la firma scritta sia stata “superata” dalla tecnologia. Infatti con l'ausilio di strumenti moderni, quali scanner e stampanti ad altissima risoluzione, è possibile riprodurre perfettamente ogni firma e copiarla innumerevoli volte su qualsiasi documento. Questa è anche una delle tematiche a favore dell'utilizzo di firme elettroniche, che ormai vengono ritenute molto più sicure.

## 6.3 Documento elettronico e firma digitale

È possibile distinguere[47]:

- documenti elettronici *strictu sensu*, cioè memorizzati in forma digitale e non percepibile da una persona senza l'ausilio di un computer
- documenti elettronici *lato sensu*, cioè prodotti utilizzando un computer ed una stampante, e comunque firmati a mano, che quindi non ci interessano

Il principale problema di un documento elettronico (strictu sensu) è che, per sua natura, è possibile modificarlo, cancellando o aggiungendo informazioni, senza che ciò sia rilevabile.

Le tecniche di firma digitale, basate, come si è già visto, su algoritmi crittografici a chiave pubblica, eliminano questo inconveniente. Infatti dal confronto fra un documento e la rispettiva firma digitale è possibile:

1. verificare l'*autenticità di provenienza* del documento, cioè identificarne “con certezza” l'autore
2. verificare l'*integrità* del documento.

Non è necessario che documento e firma si trovino sullo stesso supporto digitale: le due parti possono anche essere separate, senza che per questo venga meno il legame esistente fra di esse.

È opportuno osservare, però, che una firma digitale lega un documento *non direttamente all'autore, ma soltanto ad una coppia di chiavi crittografiche*. Quindi è necessario un ulteriore meccanismo che sancisca il legame fra la coppia di chiavi ad una particolare persona, fisica o legale che sia. Tale meccanismo, come evidenziato nel capitolo 4, è quello dei certificati, rilasciati da apposite autorità di certificazione.

Da un punto di vista legale, una AC deve:

- *identificare* in modo affidabile una persona
- verificarne la *capacità legale*: in ogni nazione ci sono delle condizioni che impediscono di firmare contratti, quali ad esempio la minore età, o l'incapacità mentale, o la bancarotta
- attestare l'*appartenenza* di una chiave pubblica ad una *persona fisica*

Bisogna notare che sia una persona fisica che una persona legale possono avere un certificato di chiave di firma, ma, come avviene per le firme manoscritte, è sempre una persona fisica che, alla fine, appone la firma.

## 6.4 Validità legale di un documento elettronico

In tutti i sistemi legali è possibile, da parte di privati che vogliano

[47] E. Giannantonio, "*Manuale di diritto dell'informatica*", Padova, 1994.

stipulare contratti elettronici, attribuire ai documenti con firma digitale lo stesso valore legale di quelli cartacei con firma manoscritta: è sufficiente che le parti in questione firmino preventivamente una scrittura privata (in forma cartacea), in cui dichiarano di considerare del tutto equivalenti i due tipi di documento.

Ciò, purtroppo, non è sufficiente allo sviluppo effettivo su larga scala di attività come il commercio elettronico e simili. Infatti, mentre per piccole comunità *locali* è possibile che ognuno stipuli un contratto del tipo sopra citato, per comunità *globali* tale approccio minerebbe alla base lo sviluppo delle suddette attività. La caratteristica fondamentale per una reale esplosione dell'*e-commerce* è infatti proprio la possibilità di stipulare contratti commerciali direttamente in forma elettronica, senza alcun incontro fisico preliminare fra le parti, che potrebbero trovarsi a migliaia di chilometri di distanza.

Quindi bisogna che sia il sistema legale di un paese provveda a dettare regole, in forma di leggi, che stabiliscano una volta per tutte come bisogna comportarsi davanti ad un documento firmato elettronicamente, quale sia la validità legale di una firma digitale, se sia più o meno sicura di una firma manoscritta su di un documento cartaceo, ecc...

Uno dei problemi principali che ostacola l'uguaglianza legale di un documento elettronico ad uno cartaceo è che in ogni sistema esiste il concetto di **originale** di un documento, che è il documento cartaceo

su cui è stata apposta direttamente la firma. Qualsiasi **copia** del documento è distinguibile dall'originale, ed ha un valore legale inferiore rispetto ad esso.

Per un documento elettronico, invece, tale distinzione perde di significato. Infatti ogni copia di un documento elettronico è perfettamente identica, in ogni sua parte, all'originale. L'unica differenza consiste nel diverso supporto fisico su cui la copia si trova ad essere memorizzata.

Un'attenta (e forse un po' troppo speculativa) analisi delle modalità di elaborazione e memorizzazione da parte dei calcolatori svela, in realtà, che l'unico originale di un documento elettronico si trova nella memoria principale del calcolatore al momento della creazione del documento stesso, e quindi ogni sua archiviazione su qualsiasi supporto (disco rigido, disco ottico, dischetto magnetico, ecc…) risulta in un'archiviazione di una *copia* del documento stesso. Come se non bastasse, è impossibile ignorare il fatto che, nel momento in cui si visiona un documento attraverso il monitor di un computer, ciò che si guarda non è affatto il documento vero e proprio, ma soltanto la sua *rappresentazione*, peraltro ottenuta copiando ancora una volta il documento dal supporto fisico alla memoria volatile del computer in questione.

In sostanza, nel mondo dei calcolatori, qualsiasi documento si prenda in considerazione è *sempre e comunque* una copia, mai

un'originale. Si potrebbe affermare che un documento elettronico, nella sua forma “originale”, non può *mai* essere visionato direttamente.

Questo discorso evidenzia la necessità di avere una legislazione specifica per documenti in forma elettronica, perché la loro diversa *natura* crea dei problemi nell'adattamento delle leggi esistenti per i documenti cartacei, le quali, invece, si fondano in maniera inscindibile sui concetti di originale e copia.

Fortunatamente la situazione, in Italia, sta lentamente cambiando, grazie ad una evoluzione del panorama legale, iniziata dal momento del varo della legge Bassanini.

## 6.5 La legge Bassanini

Il *decreto legge del 10 Novembre 1997 n.513* esamina il problema dei documenti elettronici e le firme digitali, chiarendo quale sia il valore legale di tali documenti. Con questo decreto l'Italia si pone in una posizione di avanguardia rispetto ad altri paesi europei.

Si riporta successivamente un estratto di tale decreto, in cui ho evidenziato i punti da me ritenuti fondamentali.

*ART.2 DOCUMENTO INFORMATICO*

“Il documento informatico da chiunque formato, l'archiviazione su supporto informatico e la trasmissione con strumenti telematici, sono

validi e rilevanti a tutti gli effetti di legge se conformi alle disposizioni del presente regolamento".

*ART.4 FORMA SCRITTA*

"Il documento informatico, munito dei requisiti previsti dal presente regolamento, soddisfa il requisito legale della *forma scritta*".

*ART.5 EFFICACIA PROBATORIA DEL DOCUMENTO INFORMATICO*

"Il documento informatico, sottoscritto con firma digitale ai sensi dell'articolo 10, ha efficacia di scrittura privata ai sensi dell'articolo 2702 del codice civile".

*ART.6 COPIE DI ATTI E DOCUMENTI*

"I duplicati, le copie, gli estratti del documento informatico, anche se riprodotti su diversi tipi di supporto, sono validi e rilevanti a tutti gli effetti di legge se conformi alle disposizioni del presente regolamento".

"I documenti informatici contenenti copia o riproduzione di atti pubblici, scritture private e documenti in genere, compresi gli atti e documenti amministrativi di ogni tipo, spediti o rilasciati dai depositari pubblici autorizzati e dai pubblici ufficiali, hanno piena efficacia, ai sensi degli articoli 2714 e 2715 del codice civile, se ad essi è apposta o associata la firma digitale di colui che li spedisce o rilascia,

secondo le disposizioni del presente regolamento”.

“Le copie su supporto informatico di documenti, formati in origine su supporto cartaceo o, comunque, non informatico, costituiscono, ad ogni effetto di legge, gli originali da cui sono tratte se la loro conformità all'originale è autenticata da un notaio o da altro pubblico ufficiale a ciò autorizzato, con dichiarazione allegata al documento informatico e asseverata con le modalità indicate dal decreto di cui al comma 1 dell'articolo 3”.

“La spedizione o il rilascio di copie di atti e documenti di cui al comma 2 esonera dalla produzione e dalla esibizione dell'originale formato su supporto cartaceo quando richieste ad ogni effetto di legge”.

*ART.10 FIRMA DIGITALE*

“A ciascun documento informatico, o a un gruppo di documenti informatici, nonché al duplicato o copia di essi, può essere apposta, o associata con separata evidenza informatica, una firma digitale”.

“L'apposizione o l'associazione della firma digitale al documento informatico equivale alla sottoscrizione prevista per gli atti e documenti in forma scritta su supporto cartaceo”.

“La firma digitale deve riferirsi in maniera univoca ad un solo soggetto ed al documento o all'insieme di documenti cui è apposta o associata”.

“L'apposizione di firma digitale integra sostituisce, ad ogni fine previsto dalla normativa vigente, l'apposizione di sigilli, punzoni, timbri, contrassegni e marchi di qualsiasi genere”.

*ART.11 CONTRATTI STIPULATI CON STRUMENTI INFORMATICI O PER VIA TELEMATICA*

“I contratti stipulati con strumenti informatici o per via telematica mediante l'uso della firma digitale secondo le disposizioni del presente regolamento sono validi e rilevanti a tutti gli effetti di legge”.

*ART.12 TRASMISSIONE DEL DOCUMENTO INFORMATICO*

“Il documento informatico trasmesso per via telematica si intende inviato e pervenuto al destinatario se trasmesso all'indirizzo elettronico da questi dichiarato”.

“La trasmissione del documento informatico per via telematica, con modalità che assicurino l'avvenuta consegna, equivale alla notificazione per mezzo della posta nei casi consentiti dalla legge”.

*ART.16 FIRMA DIGITALE AUTENTICATA*

“Si ha per riconosciuta, ai sensi dell'articolo 2703 del codice civile, la firma digitale, la cui apposizione è autenticata dal notaio o da altro pubblico ufficiale autorizzato”.

“L'autenticazione della firma digitale consiste nell'attestazione, da

parte del pubblico ufficiale, che la firma digitale è stata apposta in sua presenza dal titolare, previo accertamento della sua identità personale, della validità della chiave utilizzata e del fatto che il documento sottoscritto corrisponde alla volontà della parte e non è in contrasto con l'ordinamento giuridico ai sensi dell'articolo ...”

“L'apposizione della firma digitale da parte del pubblico ufficiale integra e sostituisce ad ogni fine di legge l'apposizione di sigilli, punzioni, timbri, contrassegni e marchi comunque previsti”.

## 6.6 Considerazioni finali

Come risulta evidente dalla lettura di questi estratti di legge, il Nostro Paese sta finalmente affrontando in maniera definitiva il problema della migrazione verso i documenti elettronici, con un decreto che dà ad un documento firmato elettronicamente lo stesso valore legale che avrebbe avuto il corrispettivo cartaceo firmato a mano.

Il decreto non solo prende in considerazione il valore della sola firma digitale, ma si preoccupa anche della firma digitale autenticata, cioè apposta in presenza di un notaio, nel qual caso, chiaramente, le garanzie di sicurezza agli occhi del destinatario di un tale documento saranno sicuramente maggiori. Infatti il notaio ha la responsabilità di verificare con la massima scrupolosità sia le generalità del firmatario che la corretta corrispondenza fra chiave privata usata per

l’apposizione della firma e chiave pubblica dichiarata nel certificato dell’utente. Non bisogna dimenticare, infatti, che il meccanismo della firma digitale permette comunque il “prestito” della propria chiave di firma ad un utente, che così assume piena libertà di firmare per conto del possessore. La firma digitale di un individuo è, in definitiva, perfettamente equivalente all’apposizione di un *timbro* personale. Ognuno ha la facoltà di timbrare da sé un documento, ma ha anche la libertà di dare il timbro ad altri individui. Ciò, ovviamente, non può accadere nel caso di firme autenticate da un notaio.

Quindi, data la facilità con cui è possibile “prestare” la propria firma digitale, è indispensabile che, accanto alla nascita delle nuove tecnologie, ci sia un processo di “educazione” e responsabilizzazione degli utenti, in modo che ognuno possa comprendere a pieno le conseguenze anche legali che potrebbe avere il cedimento della propria firma ad altri.

# Capitolo 7. L'importanza delle smartcard nelle PKI

## 7.1 Introduzione

Nei capitoli precedenti è stato evidenziato come nell'ambito di una infrastruttura a chiave pubblica il concetto di *segretezza* delle chiavi private giochi un ruolo predominante. Infatti è di fondamentale importanza che sia le chiavi private di firma che quelle di cifratura siano note soltanto ai legittimi proprietari e che non cadano nelle mani sbagliate. In realtà, per questioni di praticità, è necessario non solo che una chiave privata sia mantenuta segreta, ma anche che sia facilmente *trasportabile*. Il proprietario dev'essere messo in condizione di utilizzarla in luoghi diversi.

Si pensi alla firma digitale: ognuno vorrebbe poter apporre la propria firma in qualunque luogo, come avviene per quella manoscritta, senza correre alcun rischio aggiuntivo rispetto all'esecuzione di una tale operazione in un ambiente "protetto".

Si pensi ancora all'autenticazione di utenti per garantire un

controllo degli accessi a determinati locali: adottando uno schema a chiavi pubbliche sarebbe necessario che l'utente dimostri di possedere la propria chiave privata "in loco", all'ingresso di ognuno dei locali protetti da un tale sistema. Ecco allora che l'utente necessita di un *mezzo di trasporto sicuro* della propria chiave privata di accesso.

Si vedrà che un tale dispositivo, affinché risulti realmente sicuro, deve essere dotato di un'*intelligenza* di qualche tipo, in modo da rendere impossibile la lettura della chiave in esso contenuta da parte di "estranei" che dovessero eventualmente venirne in possesso.

## 7.2 Le possibili alternative

Si consideri inizialmente l'ipotesi in cui ognuno memorizzi la chiave privata in chiaro in un file all'interno del proprio Personal Computer.

In tal modo chiunque avesse accesso al PC in questione potrebbe leggere la chiave privata o utilizzarla direttamente sostituendosi al legittimo proprietario. Allora l'unico modo per garantire la segretezza di una tale chiave sarebbe un controllo sicuro dell'accesso all'elaboratore, realizzabile mediante attivazione della password di protezione dell'elaboratore stesso, e/o chiusura a chiave del locale in cui esso è custodito.

Tale soluzione sicuramente non si adatta ai casi in cui più

persone lavorano su di uno stesso elaboratore. Inoltre risulta efficace solamente se i dati protetti dalla chiave privata hanno poca importanza per altre persone. Bisogna infatti considerare che dei dati possono ritenersi al sicuro solamente quando, per tutti gli altri utenti, il costo necessario per violare il sistema di sicurezza che li protegge è molto superiore al valore dei dati stessi. Infatti, nel caso in considerazione, un ipotetico attaccante potrebbe essere disposto a scassinare il locale ove è custodito l'elaboratore, smontare il disco rigido, collegarlo ad un altro elaboratore e leggerne il contenuto.

Naturalmente per i due casi esaminati sarebbe sufficiente memorizzare la chiave non in chiaro, ma cifrata, ad esempio, con una chiave simmetrica ottenuta da una "passphrase", che verrebbe richiesta dagli applicativi ad ogni tentativo di utilizzo della chiave stessa.

In tal modo, anche in caso di lettura del file di chiave, solo il possessore della "passphrase", cioè il proprietario, potrebbe decifrarlo. Ma ecco che allora la sicurezza della chiave privata verrebbe a coincidere con la sicurezza della "passphrase" stessa: conoscere quest'ultima sarebbe equivalente a conoscere la chiave privata. Risulta allora evidente che l'utilizzo di sofisticate tecniche crittografiche diverrebbe inutile: la "passphrase" dovrebbe essere sufficientemente mnemonica da poter essere ricordata facilmente dal legittimo proprietario. Ma con altrettanta facilità un estraneo potrebbe

allora indovinarla, magari sfruttando le proprie conoscenze su cose o fatti personali dell'interessato.

L'unica possibilità che rimane, naturalmente, è quella di non memorizzare affatto la chiave sull'elaboratore destinato ad usarla, ma mantenerla su un qualche altro tipo di dispositivo trasportabile. In questo modo ognuno potrebbe portarsi dietro la propria chiave privata, e permetterne di volta in volta la lettura agli applicativi che ne hanno bisogno.

Una soluzione economica potrebbe essere costituita da un dischetto magnetico (affidabile). Naturalmente, per evitare che chiunque, impossessandosi del dischetto, possa "impersonare" il proprietario della chiave, il contenuto dovrebbe essere cifrato con una chiave ottenuta da una qualche "passphrase".

Una tale soluzione funzionerebbe perfettamente se ci si potesse sempre e comunque "fidare" dei dispositivi in cui si inserisce il dischetto per l'utilizzo della chiave. Purtroppo ciò non sempre è vero. Infatti a volte non ci si può fidare neanche del *proprio* Home Computer, come è dimostrato dalle innumerevoli volte in cui ci si scontra con virus e simili.

Bisogna considerare che la chiave memorizzata su un supporto magnetico comunque va trasferita ad un elaboratore, prima del suo utilizzo, e tale elaboratore potrebbe contenere programmi "maliziosi"

in grado di memorizzare e/o trasmettere ad altri la chiave privata durante il suo utilizzo da parte del proprietario ignaro di tutto.

A questo punto risulta evidente che l'unica soluzione per difendersi da casi del genere è quella in cui il dispositivo di memorizzazione non rivela affatto la chiave contenutavi, ed è quindi esso stesso capace di utilizzarla per gli scopi prefissati.

Un tale dispositivo, ovviamente, non può essere certamente un supporto “passivo”, ma deve essere dotato di una capacità di elaborazione sufficiente all'esecuzione di complessi algoritmi crittografici.

Naturalmente un computer portatile potrebbe essere la soluzione più immediata, ma si vedrà nella prossima sezione che la tecnologia offre oggi soluzioni più vantaggiose: le smartcard.

## 7.3 Smartcard

Il termine "smartcard" viene comunemente utilizzato, oggi, per indicare tutti quei dispositivi costituiti da un circuito integrato incorporato in una scheda di plastica simile a quella delle carte telefoniche o delle carte di credito. Lo standard ISO preferisce adottare il termine più specifico **Integrated Circuit Card (ICC)**, cioè scheda a circuito integrato.

Tali dispositivi, come illustrato in Figura 7.1, possono essere interfacciati con un elaboratore elettronico mediante una periferica,

comunemente nota come “lettore di smartcard”, che nello standard ISO viene indicato con il termine **Interface Device (IFD)**, cioè dispositivo d'interfaccia.

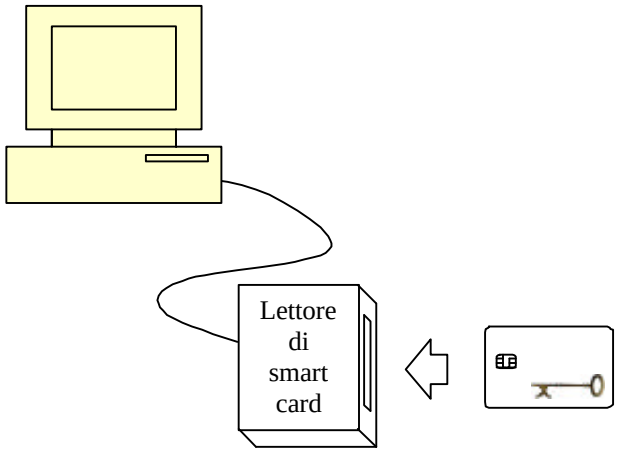


*Figura 7.1. Rappresentazione schematica di un lettore di smartcard collegato ad un PC*

A seconda della modalità di comunicazione con il mondo esterno, distinguiamo:

- smartcard con contatti (le più comuni)
- smartcard senza contatti (*contactless smartcard*)

*Contactless smartcard*

Una scheda senza contatti è un dispositivo in grado di scambiare dati con un opportuno lettore che si trovi ad essa “sufficientemente” vicino. Non è necessario inserire fisicamente la scheda all'interno di alcuno slot, come invece avviene per le card a contatti (Figura 7.2). Ciò svincola completamente la scheda dall'avere una forma ben

definita, e permette la realizzazione di smartcard “embedded” in oggetti che, a prima vista, presentano funzionalità del tutto diverse, come ad esempio una chiave (nel senso classico del termine).

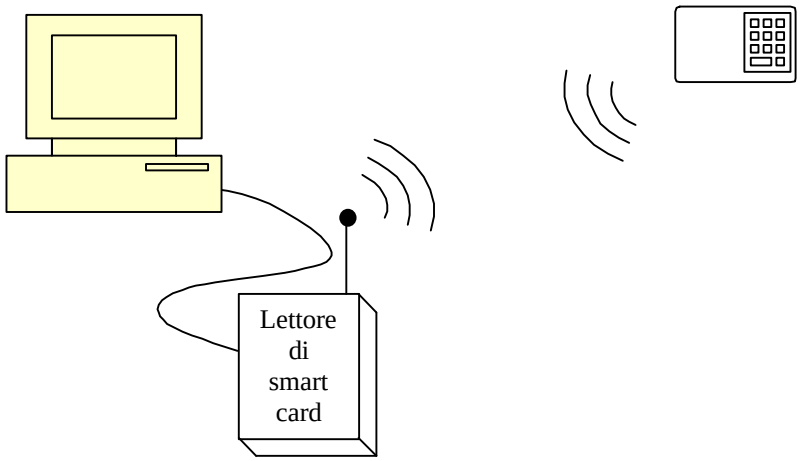


*Figura 7.2. Rappresentazione schematica di una smart card contactless che comunica con un elaboratore.*

Questo approccio, che a prima vista potrebbe sembrare “eccessivo”, permette di incrementare notevolmente il livello di sicurezza di dispositivi a serratura, da sempre realizzati meccanicamente.

Infatti, classicamente, uno dei principali problemi di tali dispositivi è l'esistenza della possibilità di “clonare” perfettamente la chiave.

L “aggiunta delle funzionalità di sicurezza messe a disposizione dai complessi algoritmi crittografici esistenti oggi innalza il livello di sicurezza di tali dispositivi, rendendo pressoché impossibile la falsificazione della chiave.

Questa tecnologia può essere utile, ad esempio, nella

realizzazione di meccanismi di serratura per auto, ove, oltre alla corrispondenza meccanica fra chiave e serratura, è necessario anche che la chiave porti a termine (con successo) un vero e proprio protocollo di autenticazione con la serratura stessa, affinché si abbia l'apertura.

Nelle smartcard contactless lo scambio dei dati con il lettore avviene mediante l'utilizzo di onde elettromagnetiche. Entrambi i dispositivi devono essere in grado sia di trasmettere che di ricevere segnali sulle opportune lunghezze d'onda, ciò che rende l'elettronica interna di una tale card più complessa rispetto a quella di una card a contatti.

Uno dei problemi principali delle contactless è la sorgente di alimentazione. Alcune schede posseggono, a bordo, una batteria interna di alimentazione, ciò che ne vincola notevolmente l'autonomia, oltre a porre vincoli incredibilmente stretti sull'assorbimento (in corrente) della circuiteria. Infatti maggiore è l'assorbimento, minore sarà l'autonomia della scheda.

L’approccio alternativo è quello di utilizzare direttamente le radiazioni elettromagnetiche per l'alimentazione della scheda. Infatti, sfruttando il fenomeno degli accoppiamenti induttivi, è possibile utilizzare onde elettromagnetiche a bassa frequenza per trasferire dal lettore alla scheda l'energia sufficiente all'operatività dei circuiti elettrici a bordo. Tale soluzione elimina il problema dell'autonomia,

ma non quello del piccolo assorbimento, perché la potenza elettrica trasferibile con questa tecnica rimane comunque modesta.

Alcune smartcard senza contatti, le cosiddette **super smartcard**, eliminano addirittura la necessità, per l'utente, di avvicinarsi fisicamente al lettore, avendo incorporata una mini-tastiera ed un mini-display LCD, che permettono una perfetta interazione con l'utilizzatore.

*Caratteristiche fisiche delle smartcard a contatti*

Una smartcard a contatti, invece, è una scheda di plastica di dimensioni (Figura 7.3) 85.6x53.97x0.76mm, come specificato dagli standard ISO.

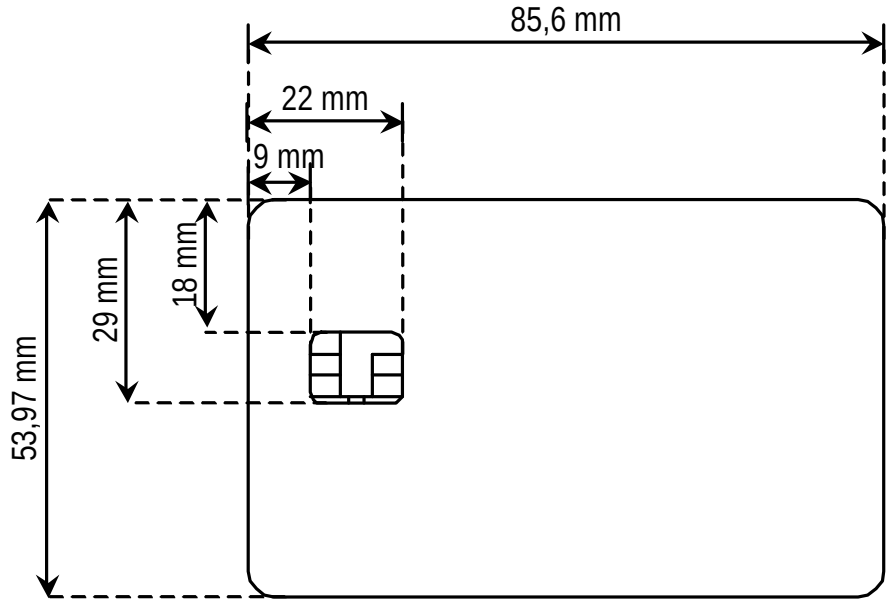


*Figura 7.3. Dimensioni di una smartcard e posizionamento dei contatti secondo lo standard ISO-7816.*

Il posizionamento dei contatti rispetta lo standard ISO 7816-2, che prevede otto contatti disposti come in Figura 7.4, dei quali soltanto sei sono attualmente utilizzati, mentre due contatti sono stati aggiunti in previsione di un utilizzo futuro (RFU[48]).

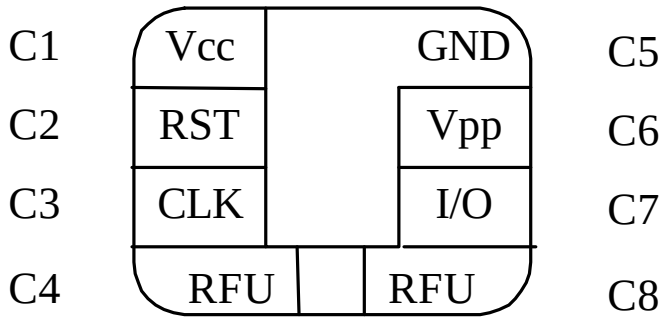


*Figura 7.4. Disposizione dei contatti di una smartcard secondo lo standard ISO 7816-2.*

Le smartcard a contatti sono le più comuni, e ad esempio sono attualmente utilizzate in diverse parti d'Europa come schede telefoniche prepagate.

La maggior parte di esse contiene un singolo chip, che necessita di una tensione di alimentazione di 5V (nel futuro sarà di 3V), e di un segnale di clock alla frequenza di 3.5795 MHz o 4.9152 MHz[49].

Quindi 2 dei 6 contatti vengono utilizzati per l'alimentazione (Vcc e GND), uno per il segnale di clock (CLK), poi abbiamo un contatto per il *Reset* (RST) della scheda, ed uno per l'input/output seriale dei

[48] Reserved for Future Usage.
[49] Tali valori per le frequenze di clock sono stati scelti considerando la disponibilità di oscillatori a basso costo in grado di fornire segnali adeguati, infatti la prima delle due frequenze coincide con la "sottoportante" prevista per lo standard televisivo americano NTSC.

dati. Le schede che contengono memorie di tipo EPROM[50] utilizzano anche un sesto contatto (Vpp) per l'applicazione della tensione di programmazione della memoria.

Al di sotto dei contatti della scheda, ben protetto dall'ambiente esterno, si trova il *core* della smartcard (Figura 7.5), un microchip dalle dimensioni ridottissime in grado di comunicare con il mondo esterno mediante i contatti sovrastanti, di immagazzinare dati in modo permanente e di effettuare elaborazioni anche molto complesse.

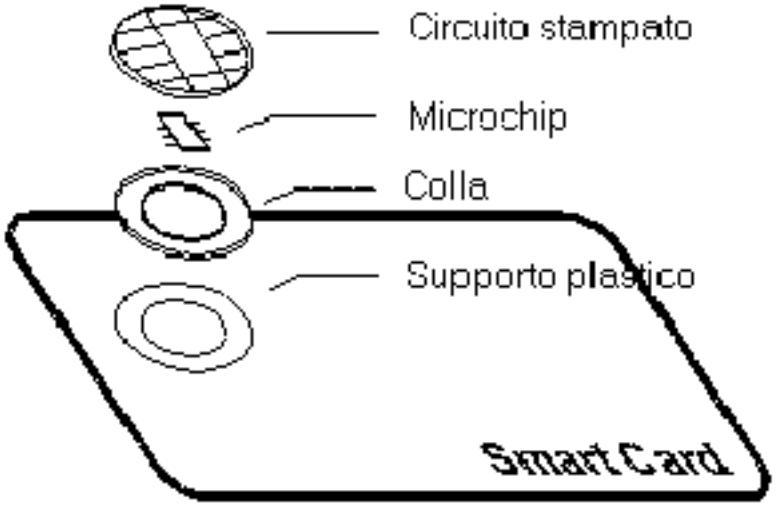


*Figura 7.5. Rappresentazione schematica dei componenti di una smartcard.*

[50] Electrically Programmable Read Only Memory: sono memorie di tipo persistente in grado di essere programmate elettricamente mediante l'applicazione di una speciale tensione. In assenza di questa, la memoria si comporta da ROM (Read Only Memory), permettendo solamente la lettura del suo contenuto.

*Architettura interna di una smartcard*

Ciò che possiamo trovare all'interno di una smartcard è, in definitiva, un intero microcalcolatore, comprensivo di:

- memoria volatile
- memoria persistente
- processore
- coprocessore crittografico

*Memoria*

Come già detto, l'utilizzo principale delle smartcard è quello della memorizzazione persistente ed altamente portabile di chiavi private, quindi un componente fondamentale è costituito dal modulo di memoria.

All'interno del chip possono trovarsi differenti tipi di memorie:

- ROM
- EEPROM
- RAM

L’utilizzo di memorie PROM all'interno di smartcard non è indicato a causa delle alte tensioni e quindi correnti necessarie alla loro programmazione. Le memorie EPROM, invece, essendo riprogrammabili mediante esposizione ai raggi ultravioletti, non

vengono utilizzate per l'impossibilità di predisporre il circuito a bordo della scheda con l'opportuna finestrella a quarzo.

Le memorie persistenti più usate nelle smartcard sono quindi quelle di tipo EEPROM, che risultano cancellabili elettricamente, e possono essere riscritte da 10.000 ad 1 milione di volte.

*RAM e processore*

La memoria di tipo RAM, volatile, è utile soltanto in presenza di un'unità di elaborazione centrale (CPU) sulla smartcard, e mantiene tutti i dati temporanei che sono necessari all'esecuzione del programma da parte della CPU.

Comunemente si utilizza il termine smartcard per indicare sia:

1. schede con sola memoria: permettono la memorizzazione ed il successivo recupero di dati
2. schede con memoria e un po' di logica di sicurezza: permettono la memorizzazione ed il recupero di dati soltanto dopo la verifica di un codice di accesso
3. schede con memoria e CPU: permettono di far girare una vera e propria applicazione all'interno della scheda

Per le schede della terza categoria, il tipo di applicazione che può girare varia da caso a caso. L'applicazione di base è un semplice "file-manager" per l'organizzazione di dati in file in modo simile a

quanto avviene con i supporti magnetici, e con funzionalità primitive di autenticazione e/o protezione da scrittura.

Il tipo di applicazione che si interfaccia con le PKI, invece, è capace di eseguire algoritmi crittografici direttamente nella scheda, operando su dati forniti man mano dal mondo esterno, ed implementa sofisticati meccanismi di autenticazione. Un'applicazione di questo tipo è solitamente progettata per non rivelare mai al mondo esterno la chiave privata immagazzinata nella scheda.

Esistono smartcard in grado di eseguire solamente algoritmi simmetrici, come il DES, che sono abbastanza veloci da poter essere implementati "in software" utilizzando le operazioni che una CPU "general purpose" mette a disposizione. Quando, invece, si richiedono operazioni di crittografia a chiave pubblica, è quasi obbligatorio l'utilizzo di un **coprocessore crittografico** che affianca la CPU principale nell'esecuzione di algoritmi asimmetrici, riducendo di molto il tempo di elaborazione.

Quanto detto vale principalmente per l'algoritmo RSA. Quindi alcuni produttori, per mantenere un prezzo economico e quindi competitivo, scelgono di incorporare nelle schede o soltanto algoritmi simmetrici, o anche algoritmi asimmetrici del tipo **a curve ellittiche** (EC[51]). Tali algoritmi hanno un tempo di elaborazione molto più contenuto rispetto all'RSA, ciò che li rende particolarmente adatti a

questo tipo di applicazioni. Ciò dipende dal fatto che, a parità di livello di sicurezza raggiunto[52], una chiave EC può essere molto più corta di una chiave RSA. Naturalmente ciò potrebbe dipendere solamente dal fatto che l'algoritmo EC è relativamente più “giovane” di RSA, quindi le tecniche di crittoanalisi non hanno ancora avuto il tempo di affinarsi come per RSA. Un ulteriore *rischio* nell'utilizzo di algoritmi EC lo si corre nel campo dell'*interoperabilità*. Infatti lo standard RSA è sicuramente il più diffuso e largamente utilizzato, in tutto il mondo, per applicazioni sicure, a cominciare dallo standard X.509 per i certificati.

In definitiva, per applicazioni *critiche* la scelta sicuramente migliore consiste nell'utilizzare schede con RSA a bordo, mentre in assenza di requisiti di sicurezza di altissimo livello è sufficiente l'adozione di schede più economiche con a bordo l'algoritmo EC.

Bisogna osservare che l'incorporazione di una CPU in una smartcard non è comoda solamente per applicazioni crittografiche, ma può venire in aiuto in *tutte* le applicazioni, a causa dell'estrema flessibilità che ne risulta. Infatti, in smartcard senza CPU, il comportamento della scheda viene definito in fase di fabbricazione del chip, e i tempi necessari per la produzione di un chip con una ROM diversa possono essere dell'ordine di alcuni mesi. Per contro, nelle schede a microprocessore, sono sufficienti pochi minuti per

[51] Ellipic Curve

caricare sulla scheda un nuovo programma ed effettuare quindi un aggiornamento.

Per avere un'idea della complessità che il microcalcolatore smartcard ha raggiunto oggi, si consideri che già esistono smartcard relativamente “economiche” con una JVM[53] incorporata, o con un interprete Basic incorporato.

## 7.4 Funzionamento a basso livello di una smartcard

Una smartcard diventa operativa subito dopo la ricezione del segnale di Reset, inoltrato dal dispositivo d'interfaccia sull'apposito contatto. La scheda risponde al reset con una sequenza di byte che contiene varie informazioni sulla tipologia di protocolli di trasmissione supportati e la velocità di trasferimento ottimale.

Dopo questo “preambolo”, lettore e scheda possono scambiarsi dati sul contatto seriale di I/O secondo quanto concordato.

*Il segnale di Reset*

Lo standard ISO prevede 3 modalità di reset:

1. reset interno (causato internamente dal programma in esecuzione)
2. reset attivo basso

[52] Cioè a parità di tempo di elaborazione necessario per un attacco crittoanalitico.
[53] Java Virtual Machine: è un interprete per il Java ByteCode ottenuto dalla compilazione di un programma Java).

3. reset sincrono attivo alto (utilizzato principalmente nelle schede a sola memoria)

La maggior parte delle smartcard a microprocessore utilizza il reset attivo basso, in cui la CPU rimane nello stato di reset per tutto il tempo in cui il segnale di reset rimane basso, per trasferire il controllo all'*entry-point* del programma in ROM non appena il segnale di reset torna alto.

Lo standard ISO prevede una precisa sequenza per l'attivazione e la disattivazione della scheda, al fine di minimizzare la probabilità di danneggiamento durante queste due delicatissime fasi. Per quanto riguarda l'attivazione, il dispositivo d'interfaccia deve:

1. mantenere il segnale RST basso
2. applicare la tensione di alimentazione Vcc
3. mettere il contatto di I/O in modalità ricezione
4. mettere il contatto Vpp in "idle-mode"[54]
5. applicare il segnale di clock su CLK
6. alzare il segnale RST

Per la disattivazione, invece, deve:

1. abbassare il segnale RST

[54] Cioè applicare, se previsto, al contatto $V_{pp}$ la tensione necessaria per il "normale" nunzionamento della scheda. Alcune schede richiedono l'applicazione di una tensione speciale in fase di programmazione permanente della memoria interna.

2. mantenere il segnale CLK basso

3. "scollegare" la tensione Vpp

4. portare il segnale I/O al livello basso

5. disattivare la tensione di alimentazione Vcc

*Risposta al reset (ATR)*

Dopo l'applicazione del segnale di reset, la scheda risponde con una sequenza di caratteri nota come Answer To Reset (ATR), lunga non più di 33 caratteri, e contenente le seguenti informazioni:

- un carattere iniziale (TS)
- un carattere di formato (TO)
- una serie di caratteri di interfaccia
- i caratteri “storici” (historical characters)
- un carattere di controllo (TCK)

Il carattere iniziale consiste in un pattern per la sincronizzazione di bit sulla linea, utile anche per la determinazione automatica della velocità di trasmissione della scheda.

Il carattere di formato fornisce informazioni necessarie all'interpretazione dei rimanenti caratteri della sequenza ATR, fra cui la presenza o meno di caratteri di interfaccia ed il numero di caratteri storici.

I caratteri d'interfaccia forniscono informazioni su:

- disponibilità di protocolli di comunicazione
- parametri per la corretta determinazione di tensioni e correnti di programmazione della EPROM eventualmente presente
- parametri per la corretta determinazione della frequenza di trasmissione

I caratteri “storici” possono essere utilizzati per fornire informazioni sul ciclo di vita della smartcard, ma il loro utilizzo non è ancora stato del tutto standardizzato.

*I protocolli di trasmissione T0 e T1*

Lo standard ISO prevede, come già detto, una singola linea di comunicazione seriale fra scheda e dispositivo d'interfaccia. Questo significa che la direzione dei dati, su tale linea, varia nel tempo, permettendo sia l'input che l'output.

Il protocollo di trasmissione adottato dalla maggior parte delle schede è un protocollo di tipo "half duplex" asincrono. Esistono due protocolli fondamentali: T0 e T1.

Entrambi, per la trasmissione di un singolo carattere, utilizzano al livello più basso un protocollo seriale con le seguenti caratteristiche:

- 1 bit di start
- 8 bit di dati

- parità pari (presente solo nel T0)
- 2 bit di stop

### *IL PROTOCOLLO T0*

È un protocollo orientato al carattere (byte-oriented), in cui l'unità elementare trasmessa è il byte, con un meccanismo “elementare” di rilevamento e correzione d'errore, basato sul bit di parità presente in ogni carattere trasmesso.

Questo era l'unico protocollo disponibile fino al 1992, anno in cui l’ISO standardizzò il protocollo T1. Per questo motivo, oltre che per la sua maggiore semplicità, è rimasto il più utilizzato all'interno delle smartcard.

Questo protocollo prevede che il lettore si comporti da “master” e la smartcard da “slave”. Questo significa che è sempre il lettore a prendere l'iniziativa, inoltrando sulla linea di I/O una **trama di comando**. Ad essa la smartcard risponde con un **byte di risposta** *(ACK)*. Successivamente si ha il trasferimento di un **blocco di dati**, o dal lettore alla card o viceversa, ed infine la smartcard termina con due **byte di stato** *(SW1 e SW2)*.

La trama di comando consiste di 5 caratteri (Figura 7.6):

- un carattere per la classe dell'istruzione (CLA)
- un carattere per il codice dell'istruzione (INS)

- due caratteri per gli eventuali parametri dell'istruzione (P1 e P2)
- un carattere per la lunghezza di un eventuale blocco di dati seguente (P3)

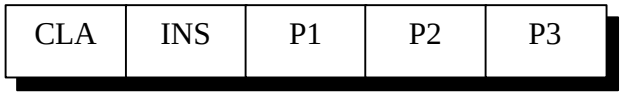


*Figura 7.6. Contenuto della trama di comando*

Il byte di risposta, normalmente, consiste nella ripetizione del codice d'istruzione.

Successivamente al byte di risposta, possono aversi due situazioni, come schematizzato in Figura 7.7:

1. l'istruzione specificata prevede un trasferimento di dati *verso* la scheda: in tal caso il lettore inoltrerà sulla linea di I/O un blocco di dati
2. l'istruzione specificata prevede un trasferimento di dati *dalla* scheda: in tal caso sarà la smartcard ad inoltrare sulla linea di I/O un blocco di dati

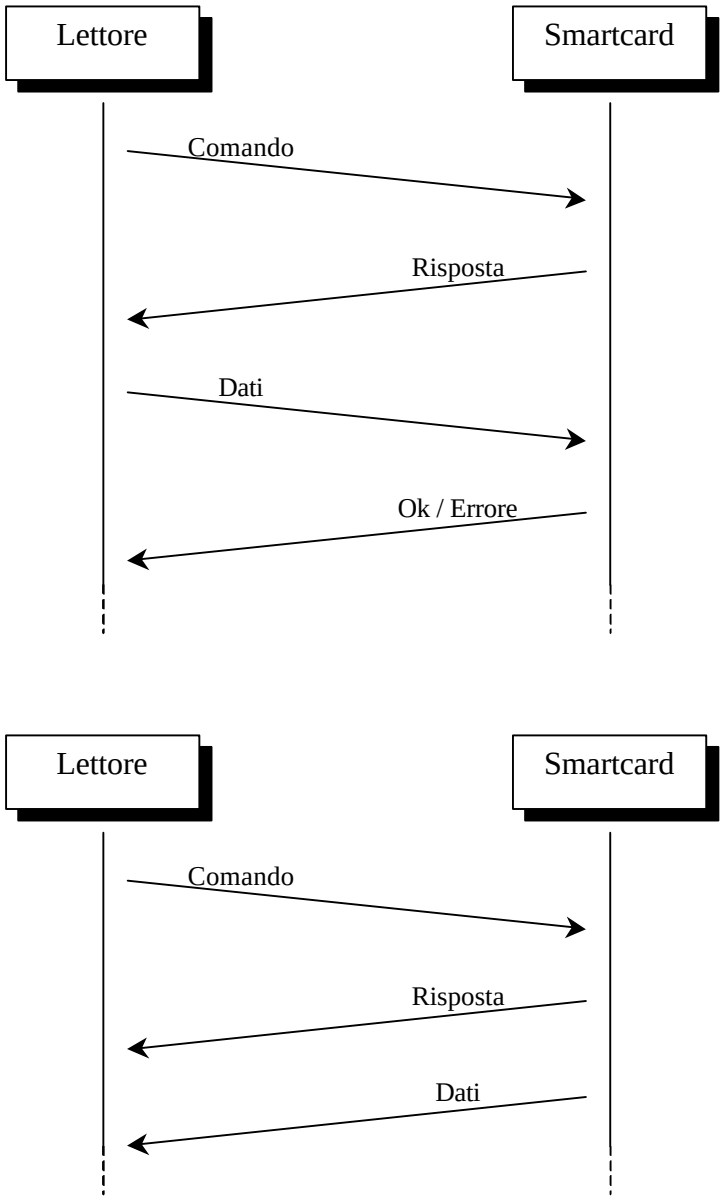


*Figura 7.7. Coppia comando-risposta*

I due byte di stato specificano se l'operazione ha avuto successo (in tal caso risulta SW1,SW2 = 0x90,0x00) o meno, e codificano un eventuale codice d'errore.

Per quanto riguarda eventuali situazioni di errore nella

trasmissione, il protocollo prevede che, dopo la trasmissione di ogni carattere, nel caso in cui rilevi un errore di parità, il ricevente forzi un livello basso sulla linea di I/O entro il termine del primo stop-bit. In questo modo viene richiesta la ritrasmissione dell'intera trama di carattere.

### *IL PROTOCOLLO T1*

È un protocollo *block-oriented* in cui l'unità di base trasmessa può essere una sequenza di bytes di lunghezza variabile. Inoltre il protocollo prevede un meccanismo più avanzato di *error-recovery*, nonché un meccanismo per il *controllo di flusso* ed un ulteriore meccanismo per la gestione di *canali multipli*. Ne risulta chiaramente un protocollo di comunicazione di più alto livello rispetto al T0.

Facendo un paragone con lo stack di protocolli previsto nel modello ISO/OSI, T0 si colloca al livello fisico (Physical Layer), mentre T1 al livello Data-Link. La situazione è illustrata in Figura 7.8.

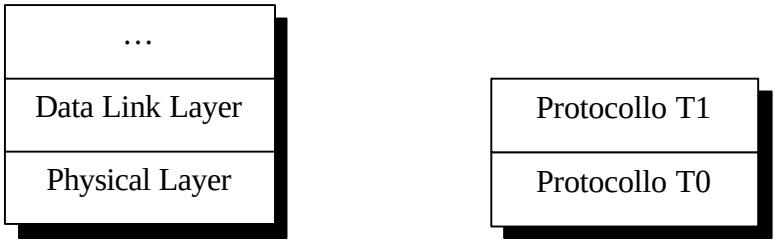


*Figura 7.8. Collocazione dei protocolli di comunicazione T0 e T1 secondo il modello ISO/OSI.*

Il protocollo T1 lavora utilizzando, al di sotto, un protocollo simile

al T0, con controllo di parità assente, e conseguentemente nessun bit di parità. Infatti il meccanismo di gestione degli errori è basato sull'intero blocco di bytes trasmesso di volta in volta, al termine del quale un campo di epilogo (EDC) contiene un codice per la correzione degli eventuali errori.

Ogni blocco (*block-frame)* contiene i seguenti campi (Figura 7.9):

- *prologo*, costituito a sua volta da:
  - 1 byte di indirizzo di nodo (NAD)
  - 1 byte di controllo per il protocollo (PCB)
  - 1 byte per la lunghezza dei dati (LEN)
- da o a 254 bytes di dati (INF)
- 1 o 2 bytes di epilogo (EDC)

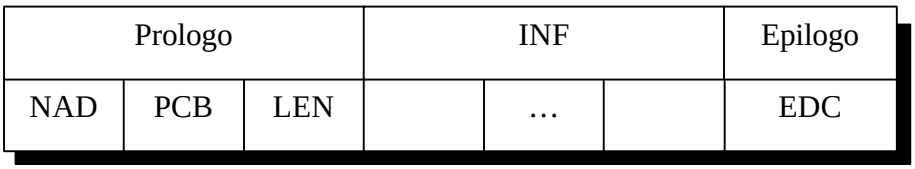


*Figura 7.9. Struttura della trama di base del protocollo T*

Il byte NAD permette l'implementazione di più canali di comunicazione logici, mediante un meccanismo di “multiplexing”. Nel byte NAD vi sono 3 bits per l'identificazione dell'indirizzo sorgente, e 3 bits per l'indirizzo di destinazione. In questo modo si possono avere fino ad 8 canali “virtuali” di comunicazione fra dispositivo d'interfaccia

e smartcard.

Il byte PCB permette di distinguere fra 3 tipi di trama:

- trama informativa, cioè contenente dati
- trama di controllo “pronto a ricevere” (Receive Ready)
- trama di controllo “di supervisione” (Supervisory Block).

Tale byte permette inoltre di effettuare un controllo di flusso primitivo, mediante un singolo bit di numero di sequenza. Infine un bit per il concatenamento di trame permette la trasmissione, mediante più trame informative consecutive, di un unico blocco di dati di lunghezza superiore a quella massima prevista in una singola trama.

Il campo LEN codifica la lunghezza del campo dati. I valori permessi vanno da 0x00 a 0xFE, limitando così a 254 il massimo numero di bytes di dati trasferibili con una singola trama informativa.

Il protocollo T1, come già detto, nasconde tale limite utilizzando la concatenazione di più trame informative per la trasmissione di blocchi di dati di qualsiasi lunghezza.

Il campo di epilogo EDC può contenere o un codice di controllo longitudinale (LRC), che occupa 1 byte, o un codice di controllo ciclico (CRC), che invece ne occupa 2. La selezione del meccanismo di error-detection è fatta a livello di caratteri d'interfaccia, nella trama di risposta al reset (ATR).

Un'ultima caratteristica del protocollo T1 è l'eliminazione del rapporto *master-slave* fra lettore e smartcard: in esso sia l'uno che l'altro possono, in ogni istante, iniziare la comunicazione.

Quanto appena esposto non lascia dubbi in merito alla superiorità del protocollo T1 rispetto al T0. Infatti l'industria delle smartcard si muove verso una sempre maggiore diffusione del primo, inizialmente ostacolata solamente dalla limitatezza delle risorse a disposizione su di una scheda. Infatti è evidente come l'implementazione del protocollo T1 da parte di una smartcard richieda risorse, in termini sia di memoria che di elaborazione, superiori a quelle richieste dal protocollo T0. D'altronde, come dimostra l'avvento delle card di ultima generazione, ciò non costituisce più un problema per i produttori.

## 7.5 Campi applicativi

Dopo aver esposto come è fatta una smartcard, cosa vi è al suo interno ed in che modo comunica col mondo esterno, si può analizzare quali siano le possibilità offerte da un tale dispositivo in termini di funzionalità di alto livello.

*“Smartcard” con sola memoria*

Le smartcard più semplici, come già detto, sono quelle che hanno solamente funzionalità di memorizzazione di dati.

Alcune schede si limitano a comportarsi come dei semplici moduli di memoria, e posseggono quindi molto poco di “intelligente” al loro interno. Tali schede, spesso, implementano un protocollo di comunicazione specifico, più semplice di quelli standardizzati esaminati nella sezione precedente. Le funzionalità richieste, in questo caso, possono essere implementate direttamente con della “logica” di controllo.

Altre schede, invece, arrivano ad implementare un vero e proprio file-system, permettendo la separazione di sequenze di bytes in diversi files, organizzati in una struttura gerarchica ad albero.

Nel primo caso la scheda si comporta semplicemente da “interfaccia”, traducendo le sequenze di comando ricevute dalla linea seriale in operazioni da effettuare sul modulo EEPROM interno. A volte tali schede hanno anche un meccanismo di protezione del contenuto da accessi non autorizzati basato su un codice di accesso (PIN). All'accensione della scheda, in tal caso, l'unica operazione effettuabile è l'autenticazione, da effettuarsi trasmettendo il PIN. Qualsiasi altra operazione genera un codice di errore. Una volta effettuata l'autenticazione, è possibile effettuare le normali operazioni di lettura e modifica del contenuto della memoria, nonché un'operazione speciale che permette di modificare il valore del PIN stesso. A volte è possibile avere due codici di protezione, uno per la lettura ed uno, diverso, per la scrittura.

Come illustrato in Figura 7.10, il comportamento di una scheda di questo tipo è del tipo “macchina a stati”, ove ad ogni stato è associato un determinato livello di protezione.

Gli stati indicati in figura hanno il seguente significato:

- **NoAut** è lo stato in cui si porta la scheda subito dopo l'accensione, in cui non è permesso eseguire alcuna operazione di lettura o scrittura della memoria
- **ReadOnly** è lo stato in cui è possibile soltanto leggere il contenuto della memoria
- **ReadWrite** è lo stato in cui è possibile sia leggere che modificare il contenuto della memoria
- **Reading** è lo stato in cui la scheda sta trasmettendo al lettore il contenuto di una parte della memoria
- **Writing** è lo stato in cui la scheda accetta dal lettore una sequenza di bytes con cui modifica il contenuto di una parte della memoria
- **Reset** è l'evento indicante l'applicazione della sequenza di reset
- **Read PIN** è l'evento indicante la trasmissione, da parte del lettore, del codice di protezione da lettura
- **Write Pin** è l'evento indicante la trasmissione, da parte del

lettore, del codice di protezione da scrittura

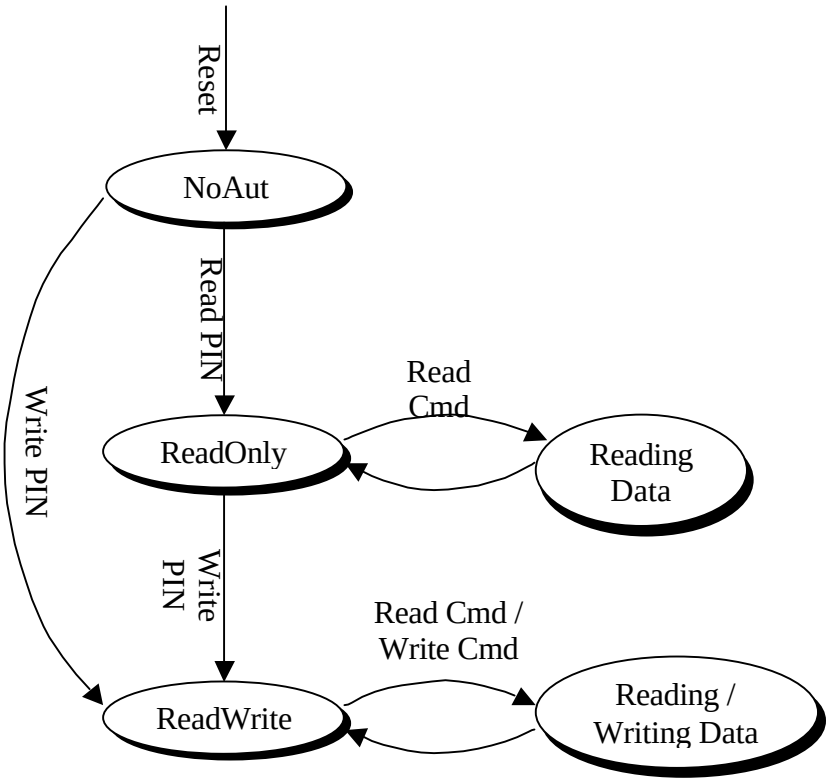


*Figura 7.10. Diagramma di stato approssimativo di una smartcard a sola memoria.*

Le smartcard con sola memoria più evolute sono quelle che supportano la **segmentazione** della memoria. Tale meccanismo risulta utile nel caso di più applicazioni che condividono la stessa scheda, o di un'unica applicazione che necessita di separare i propri dati in più segmenti.

*Smartcard per la memorizzazione di valori*

Sono smartcard a sola memoria predisposte per la memorizzazione di valori, cioè utili come “schede prepagate”, ed

hanno meccanismi di protezione molto particolari. Infatti tali schede vengono utilizzate, di norma, per precaricare, al loro interno, una certa quantità di denaro, che l'utente può spendere usufruendo di un particolare servizio.

In questo caso ciò che più conta è che non si possa, inserendo la card all'interno di un apposito lettore, aumentare il credito a disposizione. Ciò viene effettuato utilizzando, all'interno, un tipo di memoria che memorizza in ogni cella un contatore, sul quale le uniche operazioni effettuabili sono quelle di *decremento* (non in modulo, ovviamente) e di *lettura*.

Una scheda di questo tipo viene venduta con le celle di memoria già precaricate. Durante l'utilizzo del servizio il lettore provvede all'invio del comando di decremento alle varie celle. Una volta che tutte le celle hanno raggiunto il valore zero, la scheda diventa completamente inutile e può essere buttata via. Alcune schede di questo tipo supportano, comunque, un meccanismo di *ricarica* che permette di riportare i vari contatori al valore positivo iniziale.

Le schede a memorizzazione di valori sono attualmente utilizzate in tutto il mondo come schede telefoniche prepagate.

*Schede per la memorizzazione di files*

Queste schede sono quelle che implementano, al loro interno, uno schema di memorizzazione e recupero dei dati simile a quello

del filesystem di un computer. Naturalmente all'interno della smartcard non è presente alcun dispositivo magnetico di memorizzazione, e tutti i dati vengono comunque immagazzinati in un modulo di memoria di tipo EEPROM. Per ottenere ciò è necessario che la smartcard possegga un vero e proprio sistema operativo, in grado di svincolare le applicazioni dalla struttura monodimensionale della memoria fisica sottostante.

Di solito queste schede supportano almeno meccanismi elementari di protezione da scrittura dei singoli file, mentre presentano un unico codice di autenticazione globale per l'accesso all'intero filesystem. Le più sofisticate, comunque, sono in grado di mantenere, per ogni singolo file, una lista delle autenticazioni necessarie all'esecuzione di ciascuna operazione.

L'interfaccia logica del filesystem a bordo è stato standardizzato nel documento ISO 7816-3, e prevede la memorizzazione di dati in sequenze logicamente distinte, dette **file**. Ogni file è individuato da un *identificatore* numerico a 2 bytes. I file sono logicamente organizzati in una struttura gerarchica ad albero. Esistono diversi tipi di file (Figura 7.11):

- un **file elementare** (EF[55]) contiene dati che la scheda non interpreta in alcun modo

[55] Elementary File.

- un **file dedicato** (DF[56]) è un file rappresentante un directory, quindi il suo contenuto è interamente gestito dal file-system a bordo della smartcard e specifica quali EF o ulteriori DF sono presenti all'interno del directory stesso
- il **master file** è il DF alla radice della struttura ad albero, ed ha un identificatore numerico pari a *3F00* esadecimale.

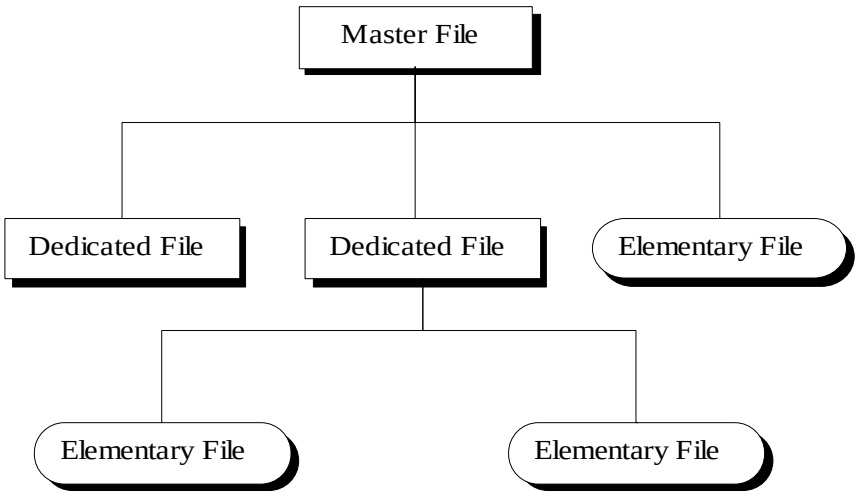


*Figura 7.11. Rappresentazione schematica del contenuto della memoria di una smartcard con filesystem a bordo.*

[56] Dedicated File.

# Capitolo 8. Realizzazione del progetto

## 8.1 Presentazione

Nel capitolo precedente sono state presentate le smartcard come soluzione efficace, sicura e comoda per la memorizzazione, il trasporto e l'utilizzo delle chiavi private di un utente. Oggi questa tecnologia si sta diffondendo sempre più, anche se in Italia siamo un po' indietro rispetto ad altri Paesi europei, dove già da anni si usano le smart card come carte telefoniche, patenti di guida, tessere sanitarie, denaro elettronico prepagato, ecc…

Pur esistendo moltissime applicazioni, purtroppo ancora non è stato fatto molto a livello di integrazione fra architetture a chiave pubblica e smartcard per la fornitura di un servizio di firma digitale sicuro. Infatti questo tipo di tecnologia, finora, è stato utilizzato quasi esclusivamente nell'ambito del commercio elettronico o delle carte di credito, o simili. Il problema di queste soluzioni è che ad ogni società commerciale interessa esclusivamente di fornire al cliente una

soluzione sicura per il pagamento dei *propri* servizi, e nient'altro. Per ottenere ciò non è assolutamente necessaria un'intera PKI, ma è sufficiente una soluzione molto meno complessa, spesso addirittura basata sulla "semplice" crittografia simmetrica, dove l'utilizzo delle smartcard diventa più un fatto di *moda* che altro. Inoltre non bisogna dimenticare che, ancora oggi, ogni applicazione commerciale cerca di differenziarsi dalle altre, di allontanarsi dagli standard, nella illusione che la segretezza di algoritmi, formati e protocolli garantiscano la sicurezza dei loro dati[57].

Naturalmente questa situazione non potrà durare a lungo: la comodità di trasporto delle smartcard è tale fin quando è sufficiente averne una o due con sé. Sembra che in un futuro imminente si dovrà avere una smartcard per i telefoni pubblici[58], una per l'accesso alla biblioteca comunale, una per l'accesso ai servizi universitari, una per l'accesso alla postazione di lavoro, una per il pagamento al supermercato (o una diversa per ogni catena di supermercati), ecc… Chiaramente, in mancanza di una cooperazione, di un'interoperabilità fra le varie applicazioni commerciali, la situazione rischia di diventare inaccettabile. Ciò che bisogna fare è dare all'utente un *unico strumento per l'identificazione personale, il pagamento*, *l'autenticazione di documenti*.

[57] A tale proposito, si veda il capitolo "Fondamenti di crittografia".
[58] O addirittura potremmo trovarci con più smartcard solo per telefonare, data la liberalizzazione della Telefonia Pubblica che sta avvenendo in Italia in questo periodo.

In questo scenario si inserisce il progetto da me realizzato: l'integrazione di un'infrastruttura a chiave pubblica con la tecnologia delle smartcard, per ottenere i servizi di firma digitale e autenticazione, nell'ambito del software "Open Source". Come sottolineato nel capitolo precedente, l'apposizione di firme digitali mediante smartcard è il sistema che offre più garanzie per la chiave privata dell'utente. Se pensiamo ai cambiamenti legali che il nostro Paese sta attraversando nel campo delle firme digitali[59], si capisce come questo lavoro rappresenti uno sforzo perfettamente in linea con i tempi.

Lo sviluppo di uno strumento "Open Source" per il rilascio di certificati elettronici ed il loro utilizzo mediante smartcard costituisce un ulteriore passo verso una sempre maggiore integrazione di questa tecnologia con gli applicativi.

## 8.2 Architettura del sistema

Il software da me sviluppato si integra con strumenti software "aperti" per la realizzazione, sotto il sistema operativo Linux, di una PKI a struttura gerarchica, per l'erogazione dei servizi di firma digitale e autenticazione con supporto per smartcard.

L'architettura generale del sistema è illustrata in Figura 8.1.

[59] Si veda il capitolo Aspetti legali della "Firma Digitale".

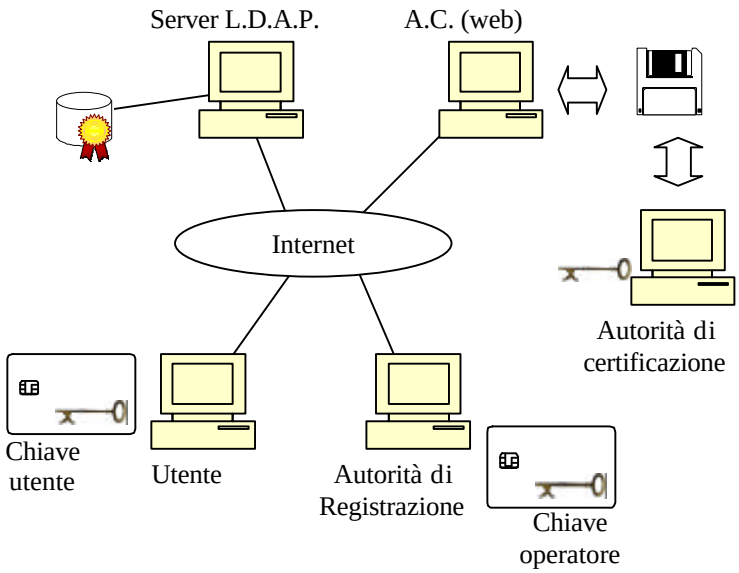


*Figura 8.1. Architettura generale del sistema.*

Possiamo distinguere le seguenti componenti software:

- **Autorità di certificazione**: costituisce l'AC di più basso livello, nell'ambito della struttura gerarchica della PKI
- **A.C. (web):** costituisce il punto di arrivo, mediante rete, delle richieste di certificazione; la loro sottoscrizione digitale, comunque, avviene soltanto sulla AC vera e propria, risiedente su di un calcolatore sconnesso per motivi di sicurezza; per questo è necessario procedere al trasferimento, mediante supporto removibile, prima delle richieste di certificazione dalla AC (web) alla AC, e poi dei certificati firmati in direzione inversa
- **Server LDAP:** immagazzina una copia di tutti i certificati

emessi dalla AC, in modo che chiunque possa reperire il certificato di un utente, mediante una ricerca basata sul DN

- **Autorità di Registrazione**: accetta (via web) le richieste di certificazione degli utenti; l'operatore della AR, dopo aver verificato le credenziali degli utenti che si recano fisicamente presso l'AR stessa, procede alla sottoscrizione digitale, mediante smartcard, della richiesta, inoltrandola verso la AC
- **Utente**: si collega al server dell'AR mediante Internet, su di un canale protetto (SSL); genera la propria coppia di chiavi, dopodiché inoltra una richiesta di certificazione insieme ai suoi dati e alla sua chiave pubblica; in questa fase la smartcard dell'utente diviene *l'unico dispositivo a conoscenza della chiave privata*; successivamente l'utente dovrà recarsi fisicamente presso la sede della AR a lui più vicina, per la verifica delle sue credenziali (mostrerà un documento valido per il riconoscimento);
- in un secondo momento l'utente potrà scaricare, dal server LDAP, il proprio certificato di chiave pubblica, che potrà memorizzare subito nella smartcard

Il flusso dei dati, nella fase di creazione del certificato dell'utente, è illustrato in figura:

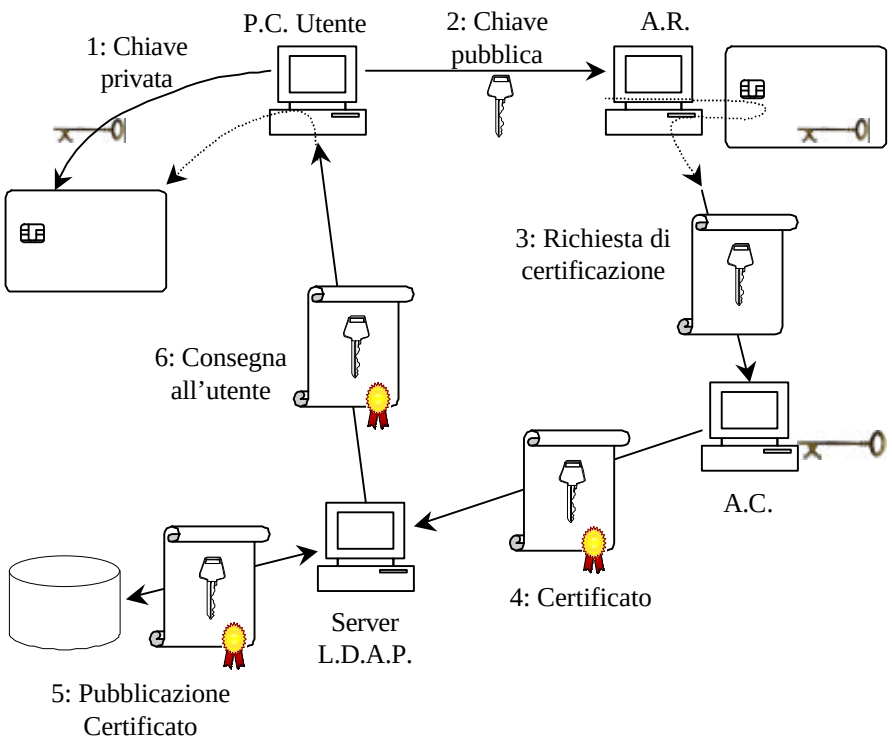


*Figura 8.2. Flusso dei dati durante la fase di generazione di un certificato per l'utente, e sua memorizzazione su smartcard.*

## 8.3 Funzionalità

Non mi soffermerò sulla specifica delle funzionalità dettagliate dei vari componenti, in quanto esse rispecchiano quanto detto in generale nel capitolo "Infrastrutture a chiave pubblica" a proposito delle architetture a struttura ad albero. Esaminerò, invece, soltanto quelle funzionalità che hanno richiesto un intervento specifico per l'interazione con le smartcard.

*PC dell'utente*

A livello utente, OpenCA prevedeva la possibilità di

- richiedere al sistema il rilascio di un certificato; la chiave privata veniva generata, memorizzata e gestita dal browser
- installare il nuovo certificato direttamente nel database dei certificati di Communicator[60]
- utilizzare la chiave privata per firmare o cifrare messaggi di e-mail, e firmare messaggi di e-news, dall'interno del browser stesso

A queste ho aggiunto la possibilità di:

1. generare la coppia di chiavi, sempre su richiesta del browser, ma in un modulo *esterno al browser stesso*
2. immediata memorizzazione della chiave privata direttamente sulla smartcard e distruzione della copia in memoria volatile; dopo questa fase il valore di tale chiave non è più reperibile da alcuno
3. installare il certificato rilasciato da OpenCA, al termine del "processo di certificazione", direttamente su smartcard, utilizzando comodamente il browser
4. utilizzare la chiave privata così memorizzata sulla propria smartcard per la sottoscrizione di messaggi di e-mail e di e-

news, e per la cifratura di messaggi di e-mail direttamente dal browser

5. utilizzare la smartcard per la sottoscrizione digitale di documenti qualsiasi, utilizzando un programma a riga di comando

6. utilizzare la chiave privata sulla propria smartcard per l'autenticazione mediante terminale: grazie al modulo PAM da me sviluppato, l'utente potrà usare un certificato rilasciato da OpenCA per autenticarsi su di un sistema; tutto ciò che l'amministratore del sistema deve fare è installare il modulo PAM e configurarlo correttamente

*Autorità di Registrazione*

A questo livello, OpenCA permetteva la sottoscrizione delle richieste di certificazione utilizzando il browser Netscape, oltre che come interfaccia grafica per l'operatore, anche per l'apposizione vera e propria della firma.

Ho ritenuto opportuno aggiungere le seguenti funzionalità:

1. possibilità di firmare elettronicamente le richieste di certificazione *mediante smartcard*, utilizzando la chiave privata dell'operatore ivi memorizzata

2. possibilità di *autenticazione dell'operatore* all'accensione del

[60] La procedura avviene via Internet, mediante l'ausilio del HTML Tag "KEYGEN".

sistema mediante smartcard

## 8.4 Implementazione

Per la realizzazione delle funzionalità appena elencate, è stato necessario sviluppare le seguenti componenti software:

- un **modulo PKCS-11** per l'integrazione delle smartcard con il software "Netscape™ Communicator"
- un **modulo PAM** per l'integrazione dei meccanismi di autenticazione smartcard-based con le più comuni applicazioni del sistema operativo Linux
- una **shell** a riga di comando che permette l'interazione dell'utente con la smartcard, ed in particolare il preciso controllo di tutti gli attributi di sicurezza della scheda
- un programma a riga di comando per la *sottoscrizione digitale* di documenti qualsiasi mediante smartcard.

L'architettura di queste componenti, esemplificata in figura, verrà esaminata con maggior dettaglio nei paragrafi seguenti.

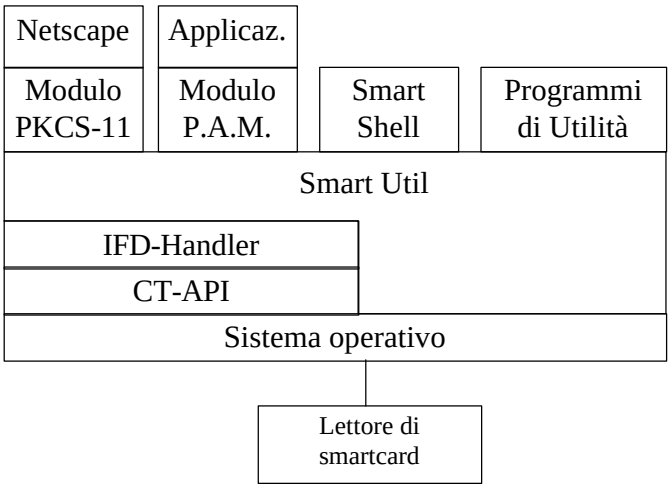


*Figura 8.3. Architettura delle varie componenti software.*

*Modulo PKCS-11*

Lo standard PKCS-11 specifica[61] "*un'interfaccia di programmazione (API) che permetta alle applicazioni di utilizzare dispositivi che memorizzano informazioni crittografiche ed eseguono funzioni crittografiche*". Tale interfaccia permette alle applicazioni di interagire con dispositivi crittografici indipendentemente dalla loro tecnologia[62].

Netscape Communicator ha scelto di adottare questo standard per la comunicazione con dispositivi crittografici, permettendo l'installazione di "*moduli crittografici*", cioè librerie dinamiche che "esportano" un'interfaccia compatibile con lo standard.

[61] RSA Laboratories, "*PKCS #11 v2.10: Cryptographic Token Interface Standard*", December 1999.
[62] Come caso limite, un "dispositivo crittografico" può addirittura essere realizzato completamente in software.

Il modulo PKCS-11 da me sviluppato va appunto installato nel Communicator come “cryptograhic module”, dopodiché può essere comodamente utilizzato dall’utente per firmare e-mail ed e-news. Dato che sulla AR OpenCA utilizza Netscape per la sottoscrizione delle richieste di certificazione (sotto forma di “form” HTML), è sufficiente installare il modulo all’interno del browser della AR per ottenere la sottoscrizione mediante smartcard da parte dell’operatore.

Per la stesura del modulo, mi sono avvalso di *gpkcs11*, una libreria che semplifica appunto lo sviluppo di moduli PKCS-11 preoccupandosi di gestire “in automatico” un certo numero di problematiche, come ad esempio la condivisione della libreria fra più applicazioni., o le operazioni di il “log” durante la fase di debug, ecc…

Durante la stesura del modulo PKCS-11 è risultata molto utile la libreria “*libcrypto.so*”, messa a disposizione dal progetto OpenSSL, per la gestione dei vari formati standard per chiavi e certificati e per l’esecuzione dei protocolli crittografici lato PC.

In figura è rappresentata l’architettura interna del modulo PKCS11, e le relazioni fra le varie sottocomponenti.

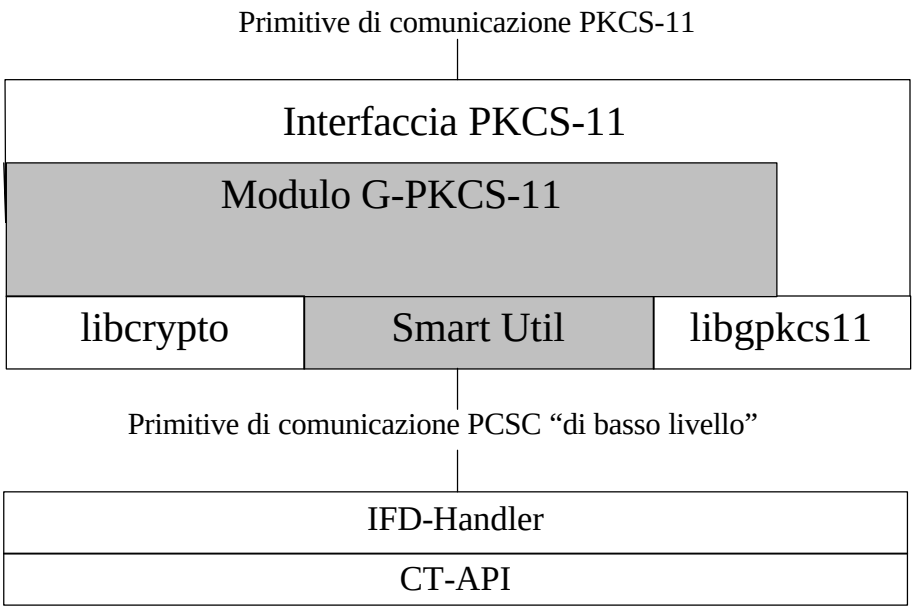


*Figura 8.4. Architettura interna del modulo PKCS-11.*

Il modulo PKCS-11 “si innesta” all’interno della struttura predisposta dal tool “gpkcs11”, che prevede l’interfacciamento verso l’esterno mediante un modulo d’interfaccia comune (il blocco più in alto, nella figura), mentre il codice vero e proprio (indicato in grigio) va scritto rispettando un’interfaccia interna, non visibile dall’esterno. Il modulo può utilizzare i servizi di ausilio messi a disposizione da gpkcs11 stesso, mediante la libreria “*libgpkcs11.so*”. Nel mio caso è stato necessario ricorrere anche alla “*libcrypto.so*” messa a disposizione da OpenSSL.

La figura evidenzia, inoltre, come nella stesura del modulo ho isolato le funzionalità di interfaccia verso la smartcard in un’unica libreria, da me chiamata “Smart Util”. Questa libreria contiene tutte le chiamate necessarie per l’interazione con la scheda, quindi, di fatto,

costituisce un driver (di basso livello) per la smartcard “Cyberflex Access 16K”, della “Schlumberger”. Questo componente è il “core” di tutte le altre componenti software da me implementate, ed è stata sempre utilizzata per la comunicazione con la scheda.

*Modulo P.A.M.*

Il modulo P.A.M.[63] che ho sviluppato permette l’identificazione dell’operatore, da parte del calcolatore della AR, mediante un protocollo di autenticazione basato su smartcard.

Il modulo è utilizzabile comodamente su di un qualsiasi calcolatore per autenticare gli utenti in base ai certificati rilasciati da OpenCA. Una volta installato, non richiede alcuna modifica delle applicazioni esistenti nel sistema, purché esse utilizzino la libreria PAM per l’autenticazione. Quindi si può utilizzare, ad esempio, il programma “login”, con la differenza che, invece della coppia “username” e “password”, l’utente dovrà fornire la coppia “Distinguished Name” e “Smartcard PIN”.

Il vantaggio nell’utilizzare le PKI nell’autenticazione risiede nella possibilità di concedere l’accesso ad un utente semplcemente rilasciandogli un certificato di accesso. Non è necessario creare preliminarmente alcun “*account*” per l’utente in questione, sulla macchina cui egli dovrà accedere, ma è sufficiente predisporre, su

[63] Pluggable Authentication Module.

quest'ultima, soltanto il certificato di "root" di OpenCA, insieme al modulo PAM, opportunamente configurato. Quest'operazione va effettuata una volta sola, su ognuna delle macchine cui gli utenti potranno accedere.

L'architettura interna del modulo PAM è esemplificata in figura.

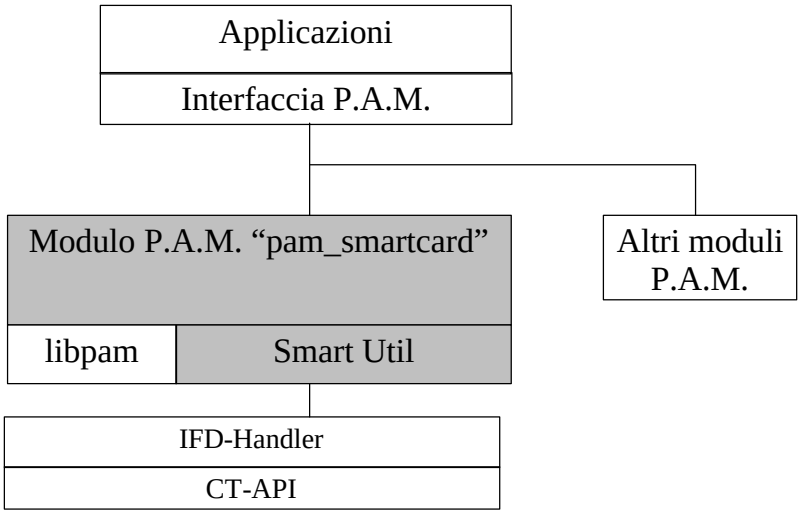


*Figura 8.5. Architettura interna del modulo P.A.M. "pam_smartcard".*

Come cerca di evidenziare la figura, l'architettura dei Pluggable Authentication Module prevede un'interfaccia comune per tutti i moduli, utilizzabile dalle applicazioni, che rimangono ignare di quale sia il meccanismo effettivo con cui si autentica l'utente.

Sarà l'amministratore del sistema a prendersi cura della configurazione delle applicazioni, predisponendo l'utilizzo dell'uno o dell'altro modulo, a seconda delle applicazioni.

Per quanto riguarda la struttura interna del modulo, quindi, esso

interagisce con le applicazioni mediante una libreria d'interfaccia ("Interfaccia P.A.M.", in figura). Lo sviluppatore ha a disposizione, nella stesura del modulo, la libreria "*libpam.so*" che fornisce alcune funzionalità ausiliarie necessarie ad una corretta implementazione dello schema di autenticazione previsto dal modello.

Il modulo da me sviluppato interagisce anche con l'IFD-Handler, il software di basso livello[64] per la comunicazione con le smartcard, indipendentemente dal modello di lettore a disposizione. Naturalmente, per la verifica del certificato dell'utente, ho dovuto ricorrere anche ad OpenSSL.

*Smart Shell*

Questa shell a riga di comando, a disposizione dell'utente, costituisce uno strumento indispensabile per il controllo e la supervisione di tutto ciò che avviene all'interno della scheda. Con essa l'utente può:

- navigare nel filesystem della smartcard
- creare, leggere, modificare, sovrascrivere file nella scheda
- copiare file dalla smartcard al PC e viceversa
- verificare che gli attributi di sicurezza dei vari file, all'interno della smartcard, siano propriamente configurati

[64] Sviluppato all'interno del progetto MUSCLE.

- autenticarsi alla smartcard, secondo le varie modalità previste dalla scheda stessa, dalla semplice verifica del PIN, ai protocolli di autenticazione a chiave simmetrica con DES, fino a quelli a chiave pubblica con RSA

Inoltre tale strumento permette agli sviluppatori l'automazione delle operazioni da svolgere con la smartcard, mediante la stesura di opportuni "script", che pilotino la shell con opportuni comandi. Il principio è ispirato al funzionamento di OpenSSL, che, oltre a predisporre una libreria utilizzabile dal "C/C++", ha messo a disposizione un tool a riga di comando che permette l'automazione di tutte le operazioni mediante script[65].

In Figura 8.3 è illustrata l'architettura della shell, peraltro molto semplice. Viene utilizzata la libreria "*Smart Util*" per l'interazione con la smartcard, mentre la "*libcrypto.so*" viene in aiuto nelle fasi di lettura dei file di certificato o di chiave nei vari formati, nonché durante i protocolli di autenticazione con la scheda, per le operazioni crittografiche lato PC.

*Programmi "Sign_SC" e "Verify_SC"*

Il primo di questi programmi è stato ottenuto come modifica del programma "Sign" distribuito con OpenCA, al quale è stata aggiunta

[65] Questo meccanismo, fra l'altro, ha permesso la scrittura della maggior parte del codice di OpenCA, basato su script CGI che utilizzano il tool a riga di comando di OpenSSL per l'effettuazione delle operazioni crittografiche.

la possibilità di apposizione della firma digitale mediante smartcard. Il programma, cioè, mantiene ancora le vecchie funzionalità di specifica della chiave privata in un file, ma in più, con l'opzione "*-sc*", cerca il certificato di chiave pubblica dell'utente dalla smartcard, e utilizza la scheda per l'apposizione successiva della firma digitale ad un file qualsiasi indicato sulla riga di comando.

La firma calcolata viene memorizzata in un file separato, secondo il formato PKCS-7, che, fra l'altro, include nella firma anche il certificato di chiave pubblica stesso. In questo modo si semplifica l'operazione di verifica della firma, in quanto, avendo già a disposizione il certificato del firmatario, non è necessario collegarsi per scaricarlo dal Server LDAP.

Il programma "`VerifySC`" permette di verificare la validità della firma apposta, avendo (chiaramente) a disposizione il file originale cui la firma si riferisce. Il programma utilizza direttamente OpenSSL per tutte le operazioni di verifica, comprese quelle di individuazione del "percorso di certificazione" che porta dall'AC locale al verificante, fino all'AC locale al firmatario.

# Capitolo 9. Guida all'uso e all'installazione del software

## 9.1 Requisiti di sistema

Il software gira in ambiente Linux e richiede l'installazione preliminare di Netscape Communicator, del server web Apache, di OpenCA e di OpenSSL versione 0.4, per la cui installazione si rimanda alla documentazione predisposta dai rispettivi distributori. È inoltre necessario avere a disposizione un lettore di smartcard ed una smartcard del tipo "Cyberflex Access 16K", prodotta dalla Schlumberger. Se si utilizza il lettore di smartcard "CHIPDRIVE" della "Towitoko" il relativo driver può essere trovato nel pacchetto, all'interno della directory "`lib/`", altrimenti è necessario scaricarsi il driver corretto dal sito del progetto MUSCLE[66].

## 9.2 Installazione

Dopo la decompressione dell'archivio, i file si presenteranno

[66] Indirizzo http://www.smartcardlinux.com

suddivisi in varie directory:

- `gpkcs11-0.6`: contiene i sorgenti del modulo PKCS-11 da installare all'interno di Netscape Communicator
- `pam_modules`: contiene i sorgenti del modulo PAM
- `smart`: contiene i sorgenti della libreria "`smart_util`", della shell interattiva "`smartsh`" e dei programmi di utilità "`sign_sc`" e "`verify_sc`"
- `include`: contiene i file include necessari per la ricompilazione di tutti i componenti software
- `bin`: contiene gli eseguibili delle diverse utility a riga di comando, eseguibili direttamente dall'interno di una shell, compilati per architettura Intel-386
- `lib`: contiene le varie librerie da installare, compilate per architettura Intel-386
- `etc`: contiene alcuni file di configurazione di esempio
- `doc`: contiene questa guida di installazione ed altri eventuali file di documentazione

## 9.3 Modulo PKCS-11

Per l'utilizzo di questo modulo è necessario installare il pacchetto gpkcs11 come modulo crittografico per Netscape. L'operazione è

effettuabile dalla finestra di dialogo “Security” di Netscape, scegliendo “Cryptographic Modules” nella tree-view di sinistra: comparirà l’elenco dei moduli crittografici installati. Premere il pulsante “Add New”, quindi immettere nel campo “Name” un nome qualsiasi, come ad esempio “`sc-token`”, e nel campo “Path” il percorso completo del file “`libgpkcs11.so`”, che si trova inizialmente all’interno del directory “`lib`” dell’archivio.

A questo punto è necessario configurare il modulo gpkcs11 per l’inclusione del modulo “`sctoken`” al suo interno. A tal fine, copiare il file “`etc/gpkcs11.rc`” nella directory “`/etc/`” di sistema; quindi aprirlo con un editor di testi e, nella sezione “`[SCTOKEN]`*”:*

- immettere il percorso completo del file “`libsctoken.so`”, che si trova inizialmente nel directory “`etc/`” dell’archivio;
- configurare la porta cui è collegato il lettore di smartcard, modificando la riga “`COM=`”

A questo punto il modulo è installato. È necessario riavviare il browser, quindi, dalla finestra di dialogo “Security”, abilitare il modulo crittografico gpkcs11 all’esecuzione delle operazioni crittografiche RSA. Se tutto è avvenuto correttamente, il browser dovrebbe segnalare la presenza o meno della smartcard (“*token*”) all’interno dello *slot* (il lettore).

*Utilizzo del modulo PKCS-11 con OpenCA*

Per generare e memorizzare una coppia di chiavi sulla smartcard è necessario prima predisporre la smartcard mediante il programma "`prepare_sc`", che si trova inizialmente nel directory "`bin/`" dell'archivio. A questo punto è sufficiente collegarsi con Netscape al server di OpenCA precedentemente predisposto per il collegamento degli utenti. L'operazione va effettuata dopo aver opportunamente Quindi, dopo aver selezionato il link "New certificate", si seguano le istruzioni che compaiono a video. Al momento della generazione della coppia di chiavi, il browser chiederà all'utente di scegliere il modulo crittografico con cui effettuare l'operazione. Bisognerà selezionare il nome con cui si è registrato il modulo gpkcs11 all'interno del browser ("`sc-token`"), dopodiché verrà generata una nuova coppia di chiavi per l'utente. La chiave privata verrà memorizzata direttamente all'interno della smartcard, e d'ora in poi nessuno sarà più in grado di leggerla o modificarla.

L'utente, ricevuta da OpenCA l'e-mail attestante l'avvenuta certificazione, si collegherà sulla pagina di OpenCA e, seguendo il link "Get requested certificate", importerà il nuovo certificato direttamente all'interno della smartcard. Se tutto ha funzionato correttamente, nella finestra di dialogo "Security", sotto la voce "Certificates➔Yours", compare il nuovo certificato, preceduto da "`sc-token:`", indicando all'utente che il certificato, in realtà, si trova

sulla smartcard. Nel caso in cui il certificato non compaia subito come descritto, provare ad effettuare un restart del browser.

Per l'utilizzo sicuro della chiave privata installata, anche su altri computer, è necessario che l'utente, al termine della procedura appena descritta, esca dal browser, ed esegua il programma "`lock_sc`" (directory "`bin/`" dell'archivio). Dopo tale operazione non sarà più possibile leggere, modificare o cancellare la chiave privata dalla smartcard, a meno che non si effettui il login come "`AUTH1`", cioè effettuando un protocollo di autenticazione crittografico con la scheda, che inizialmente utilizza la chiave memorizzata dalla fabbrica. È possibile effettuare una tale operazione dalla "`smartsh`", descritta nel seguito. Con l'utilizzo della shell è inoltre possibile sia cambiare il valore della chiave crittografica di protezione citata, sia precludersi completamente questa *backdoor* per il "riciclo" della smartcard.

## 9.4 Modulo PAM

*Guida per l'amministratore di sistema*

Questo modulo è costituito dalla libreria "`pam_smartcard.so`" (directory "`lib/`" dell'archivio), che può essere messa insieme a tutte le altre librerie PAM presenti nel sistema (directory "`/lib/security/`"), oppure in qualsiasi altro posto. L'importante è

specificare, nel file di configurazione dell'applicazione su cui si vuol abilitare l'autenticazione con smartcard, il percorso completo che individua la libreria nel filesystem. Il file "`etc/mylogin`" costituisce un file di configurazione di esempio per l'applicazione "`bin/mylogin`" che, a scopo di test, controlla il corretto funzionamento del protocollo di autenticazione, stampando un messaggio a video indicante l'avvenuta autenticazione dell'utente o meno. Tale file va copiato nella directory "`/etc/pam.d/`", come specificato nella documentazione di sistema relativa ai moduli PAM.

La configurazione del modulo PAM avviene mediante le seguenti opzioni, specificabili nel file di configurazione delle singole applicazioni (si tenga presente che, nello specificare le opzioni, il segno di uguale non deve avere spazi a sinistra o a destra):

- `device=[com1 | com2 | ...]`

  opzione *obbligatoria*: specifica la porta cui il lettore di smartcard è collegato;

  *esempio*: `device=com2`

- `wait=<seconds>`

  opzione *facoltativa*: specifica, in caso di assenza iniziale della smartcard dallo slot, il tempo massimo (in secondi) di attesa da parte del modulo; in assenza di questa opzione il software non effettua alcuna attesa

*esempio*: `wait=10`

- `root_cert=<path>`

  opzione *obbligatoria*: specifica il percorso completo, nel filesystem dell'elaboratore, del file contenente il certificato di "root" di OpenCA in formato PEM

  *esempio*: `root_cert=/etc/cacert.pem`

- `cert_id=<hex-number>`

  opzione *obbligatoria*: specifica il file elementare, all'interno della directory principale della smartcard (`/3f00`), del file contenente il certificato di chiave pubblica di autenticazione dell'utente; per la compatibilità con il modulo PKCS-11 è necessario specificare il valore `2000`, altrimenti si può specificare un altro file (il nome del file va comunque digitato come 4 cifre esadecimali)

- `dn=<filename>`

  opzione *facoltativa*: specifica il percorso completo che individua il file, nel filesystem del computer, contenente i "domini" di utenti cui è concesso inserire il DN in forma abbreviata; ogni dominio è specificato su di una singola riga, ed è costituito a partire dal DN degli utenti, eliminando il campo Common Name.

*Guida per l'utente.*

Un utente che abbia ricevuto una smartcard per l'autenticazione ad un host, contenente una chiave privata di autenticazione ed un certificato di chiave pubblica, deve *prima* inserire la smartcard all'interno del lettore e *poi* lanciare l'applicazione protetta. A questo punto il sistema richiederà l'inserimento del proprio "Distinguished Name" e, subito dopo, del PIN della smartcard. Per quanto riguarda il DN, è sufficiente inserire il proprio Common Name se i campi restanti del DN specificano un "dominio" fra quelli dichiarati dall'amministratore del sistema. In caso contrario, è necessario inserire l'intero DN.

## 9.5 Smartsh: la shell interattiva

Questo strumento permette di gestire praticamente tutte le funzionalità della smartcard "CyberflexAccess 16K":

- protocolli crittografici di autenticazione, sia basati su DES che su RSA
- navigazione nel filesystem della scheda, con la possibilità di leggere, modificare, creare e cancellare files elementari, directory e file di chiave; possibilità di trasferire interi file da e verso la smartcard
- visione e modifica degli attributi di sicurezza dei singoli file e directory (*matrice dei permessi*)

- gestione “ad alto livello” dei file di chiave, con possibilità di trasferimento di una chiave DES o RSA in formato PEM direttamente sulla smartcard con semplici operazioni.

Non mi soffermerò sulla spiegazione dettagliata dei singoli comandi, in quanto battendo `<help>` dalla riga di comando è possibile averne una lista esaustiva, con una spiegazione sufficiente a comprenderne l’immediato utilizzo. Inoltre, in caso di errori, la shell segnala chiaramente se c’è stato un errore di sintassi da parte dell’utente o comunque fornisce messaggi di errore significativi.

## 9.6 Programmi “Sign_sc” e “verify_sc”

*Sintassi: “`sign_sc <argomenti>`”*

Gli argomenti supportati sono:

- `-in <filename>`

  opzione *obbligatoria*: specifica il nome del file da firmare

- `-out <filename>`

  opzione *obbligatoria*: specifica il nome del file che conterrà la firma digitale

- `-cert <filename>`

  opzione *facoltativa*: specifica il nome del file contenente il certificato di chiave pubblica da includere con la firma;

quest'opzione è utile solo se non si usa una smartcard per apporre la firma, nel qual caso si dovrà specificare anche l'opzione "`-keyfile`"

- `-keyfile <filename>`

  opzione *facoltativa:* specifica il nome del file contenente la chiave privata di firma da usare; quest'opzione è utile solo se non si usa una smartcard per apporre la firma, e richiede anche l'uso dell'opzione "`-cert`"

- `-sc [com1 | com2 | ...]`

  opzione *facoltativa*: specifica che si utilizza una smartcard per l'apposizione della firma; il certificato di chiave pubblica viene recuperato dalla smartcard, a meno che non sia presente l'opzione "`-cert`"; la smartcard effettua fisicamente la firma usando la chiave privata al suo interno

- `-key [password | PIN]`

  opzione *facoltativa*: specifica la password che protegge la chiave pubblica, nel caso in cui questa sia memorizzata in cifra, all'interno del file specificato da "`-keyfile`"; nel caso di firma mediante smartcard, specifica il PIN di protezione della scheda

- `-h | --help`

opzione *facoltativa*: stampa un messaggio di aiuto con l'elenco delle opzioni disponibili

*Sintassi: "`verify_sc <sigfile> <argomenti>`"*

`<sigfile>` specifica il file contenente la firma, in formato PKCS7, così come viene prodotta dal comando " `sign_sc`".

Gli argomenti supportati sono:

- `-h`

  opzione *facoltativa*: stampa un messaggio di aiuto con l'elenco delle opzioni disponibili

- `-d datafile`

  opzione *obbligatoria*: specifica il file contenente la firma digitale da verificare

- `-cf certfile`

  opzione *obbligatoria*: specifica il file contenente il certificato di "root" di OpenCA

- `-cd certsdir`

  opzione *facoltativa*: specifica un directory contenente i certificati delle AC che eventualmente bisogna attraversare nel "percorso di certificazione" dalla radice dell'albero fino all'AC che ha emesso il certificato per il firmatario; in questo

directory i certificati devono avere dei nomi speciali[67], formati da un hash del rispettivo DN, e possono essere configurati correttamente mediante l'utilizzo di un Makefile[68]; se il certificato dell'utente è stato emesso dalla AC di cui si è specificato il certificato in "`-cf`", questa opzione non è necessaria.

[67] Oppure devono essere accessibili mediante link simbolici con nomi speciali.
[68] Si rimanda alla documentazione di OpenSSL per i dettagli sulla configurazione del processo di verifica di un certificato.

# Bibliografia

Prof. Dr. Hartmut Pohl, "*Guidelines for the use of Names and Keys in a Global TTP Infrastructure*", ISIS Institute for Information Security at Essen, Germany (May 1997).

Bruce Schneier, "*Applied Cryptography Second Edition : protocols, algorithms, and source code in C*", John Wiley & Sons, Inc.

"*Legal and Regulatory Issues for the European Trusted Services Infrastructure*", ETS.

E. Giannantonio, "*Manuale di diritto dell'informatica*", Padova, 1994.

"*Proposta di Direttiva del Parlamento Europeo e del Consiglio relativa a regole comuni sulle firme elettroniche*", Bruxelles, 13 Maggio 1998.

RSA Laboratories, "*PKCS #11 v2.10: Cryptographic Token Interface Standard*", December 1999

Ake Nilson, "*European Trusted Services (ETS) – Result of 1995 TTPS Projects*", April 1997

Andrew Colleran, "*Standardisation Issues for the European*

*Trusted Services*", European Trusted Services (ETS), May 1997

Brian Gladman, Brian Randell, "*A Proposal for Action in Europe for on International Information Security Standards*", European Security Standards Reference Implementation Initiative (ESSRII), May 1997

"*Regole tecniche per la formazione, la trasmissione, la conservazione, la duplicazione, la riproduzione e la validazione, anche temporale, dei documenti informatici ai sensi dell'articolo 3, comma 1, del Decreto del Presidente della Repubblica, 10 Novembre 1997, n.513*"

"*Proposta di Direttiva del Parlamento Europeo e del Consiglio relativa a regole comuni sulle firme elettroniche", Bruxelles, 13 Maggio 1998*

*"Decreto del Presidente della Repubblica del 10 Novembre 1997, n.513*"

Autorità per l'Informatica nella Pubblica Amministrazione, "*Nuove regole tecniche per l'archiviazione ottica – bozza di articolato*"

Mario Terranova, "*Firma digitale: tecnologie e standard*", Autorità per l'Informatica nella Pubblica Amministrazione (AIPA)

C. Fornaro, M. Gavelli, A. Lioy, F. Maino, "*Esperienze nello sviluppo di un'infrastruttura di rete a chiave pubblica*", Politecnico di Torino, Dipartimento di Automatica e Informatica

A. Lioy, F. Maino, M. Mezzalama, "*Secure document management and distribution in an open network environment*",

Politecnico di Torino, Dipartimento di Automatica e Informatica, Torino, Italy

CERT-IT, "*La posizione del CERT-IT a proposito della bozza di legge «Atti e documenti in forma elettronica»*"

Corrado Giustozzi, "*La bozza del regolamento tecnico: un passo indietro?*", Settembre 1998

Commissione delle comunità europee, "*Proposta di direttiva del parlamento europeo e del consiglio relativa a regole comuni sulle firme elettroniche*", Bruxelles – May 1998

Dr. Despina Polemi, "*Biometric techniques: review and evaluation of biometric techniques for identification and authentication, including an appraisal of the areas where they are most applicable – Final report*", Institute of Communication and Computer Systems – National Technical University of Athens, April 1997

Philip Zimmermen, "*Manuale d'uso del PGP™*"

D. W. Chadwick, A. J. Young, N. Kapidzic, "*Merging and Extending the PGP and PEM Trust Models – The ICE-TEL Trust Model*"

The Open Group, "*Architecture for Public-Key Infrastructure (APKI), Draft 1*", May 1997

Burton S.Kaliski Jr., "*An Overview of the PKCS Standards – An RSA Laboratories Technical Note*", November 1993

Ellison et al., "*Simple Public Key Infrastructure Certificate Theory*

*(Internet Draft)*", October 1998

RSA Laboratories, "*PKCS #11 v2.10: Cryptographic Token Interface Standard*", from "*RSA Data Security Inc. Public Key Cryptography Standards (PKCS)*", December 1999

Certicom Corp., "*Certicom Smart Card Developer's Guide*", 1997-1998

Sean Turner, Alfred Arsenault, "*Internet X.509 Public Key Infrastructure PKIX Roadmap*", September 1998

Tim Howes, S. Boeyen, P. Richard, "*Internet X.509 Public Key Infrastructure LDAPv2 Schema*", September 1998

Burton S. Kalinski Jr, "*A Layman's Guide to a Subset of ASN.1, BER, and DER*", RSA Data Security Inc.

Lutz Behnke, "*GPKCS-11 A Cryptographic Token Interface Standard Implementation, Edition 0.1 for gpkcs11, Version 0.5.0*"

Incard S.P.A., "*Incard Multiapplication Platform (IMP) Overview*"

Stephan Bausson, "*ISO7816 asynchronous smartcard information*"

"*ISO/IEC 7816 Part 4 : Interindustry command for interchange*"

Schlumberger, "*Cyberflex Access Developer's Series – Programmer's Guide*", September 1999, Version 4.0

Schlumberger, "*Cyberflex Access Software Developer's Kit 2 –*

*Release Notes*", November 12th 1999

Schlumberger, "*Cyberflex™ Access Java™ Programmable Smart Card*", December 1998

MUSCLE – Movement for Use of Smart Card in a Linux Environment, "*Smart Card Tutorial*"

Dr. David B. Everett, "*Smart Card Technology: Introduction To Smart Cards*"

Ross Anderson, Markus Kuhn, "*Low Cost Attacks on Tamper Resistant Devices*"

Alan O. Freier, Philip Karlton, Paul C. Kocher, "*The SSL Protocol – Version 3.0*", Internet-draft

"*Interoperability Specification for ICCs and Personal Computer Systems*", Bull CPS, Gemplus SA, HP Company, IBM Corp., …

Carl Ellison, Bruce Schneier, "*Ten Risks of PKI: What You're not Being Told about Public Key Infrastructure*"

"*ICC Terminal Specification for Payment Systems*", May 1998

Netscape Communications Corporation, "*Netscape Extensions for User Key Generation for Communicator 4.0 Version*"

Netscape Communications Corporation, "*Netscape Certificate Download Specification for Communicator 4.0 Version*"

Netscape Communications Corporation, "*Implementing PKCS*

*#11 for the Netscape Security Library*"

Netscape Communications Corporation, "*PKCS #11 F.A.Q.*"

Netscape Communications Corporation, "*Signing Software with Netscape Signing Tool*"

David Chaum, "*Prepaid Smart Card Techniques: A Brief Introduction and Comparison*", Copyright© 1994 by DigiCash