UNIVERSITA' DEGLI STUDI DI NAPOLI
"FEDERICO II"

FACOLTA' DI INGEGNERIA
DIPARTIMENTO DI INFORMATICA E
SISTEMISTICA

TESI DI LAUREA IN
SISTEMI DI ELABORAZIONE

PROGETTO DI UN PROTOTIPO PER LA FIRMA DIGITALE DI DOCUMENTI CON VALIDITA' GIURIDICA

RELATORI
Ch.mo Prof. Ing. Antonino Mazzeo
Ch.mo Prof. Ing. Nicola Mazzocca

CANDIDATO
Lelio della Pietra
Matr. 41/664

Anno Accademico 1999/2000

INDICE

INTRODUZIONE

Questo lavoro di tesi nasce e si inquadra in un progetto di ricerca dell'Università degli Studi di Napoli "Federico II" finalizzato alla realizzazione di alcuni sistemi informatici, previsti dalle più recenti leggi, che consentano di automatizzare, e quindi snellire considerevolmente, tutta una serie di pratiche burocratiche che attualmente sono svolte esclusivamente con materiale cartaceo, con tutto lo svantaggio che ne deriva, sia dal punto di vista della velocità di trasmissione delle informazioni (e, quindi, della relativa facilità di accesso), che da quello, non meno importante, del volume occupato fisicamente dai fogli che contengono le informazioni stesse.

Ci si trova di fronte a quella che sarà una rivoluzione nella Pubblica Amministrazione; procedure che si possono definire "storiche" e ormai profondamente acquisite nelle abitudini quotidiane, come le due trattate in questa tesi (la firma e, come esempio di applicazione, la gestione del protocollo), verranno completamente stravolte e l'intera struttura sarà affidata a macchine: il cambiamento sarà, quindi, non solo strettamente tecnico, ma anche culturale, dovendo, ad esempio, il cittadino doversi fidare di apparecchiature, come le *smartcard*, che avranno facoltà di apporre firme per suo conto.

Attualmente, la firma apposta (sotto certe condizioni) in calce ad un documento cartaceo vuole dimostrare la volontà del soggetto firmante di sottoscrivere il documento stesso; tutta una

serie di tecniche grafologiche garantiscono che, in caso di controversia, sia possibile verificare che una determinata firma appartenga ad un determinato soggetto (e che quindi questi non possa ripudiarla). Un siffatto sistema (insieme a timbri, sigilli e punzoni, per i quali vale un discorso molto simile) ha contraddistinto la burocrazia praticamente dall'invenzione della scrittura fino ai nostri giorni.

In un'epoca in cui si mira alla quasi totale eliminazione del materiale cartaceo a vantaggio dell'informazione automatica, comincia a farsi sempre più pressante la necessità di dover "firmare" (intendendo con questo termine la manifestazione tangibile della volontà di sottoscrivere) i documenti direttamente nella loro forma digitale, senza dover stampare il documento e conservarlo in un archivio non elettronico, con tutti gli ovvi svantaggi che ne derivano. Una siffatta procedura prende il nome di Firma Digitale.

Poiché dal punto di vista tecnico tutto ciò è già realtà (al momento gli ostacoli sono dovuti a rallentamenti burocratici), sarà quindi possibile, in un futuro così prossimo da poter essere considerato già presente, generare un documento direttamente al computer, sottoscriverlo e inviarlo utilizzando le attuali reti di telecomunicazioni senza che un solo carattere venga stampato su un foglio di carta. Non solo: si riesce a dimostrare (ma questo non è argomento di questa tesi) che contraffare una Firma Digitale risulta operazione estremamente più ardua (se non addirittura impossibile) rispetto alla contraffazione di una firma tradizionale.

Scopo di questa tesi è quello di realizzare un'architettura che implementi una Firma Digitale conforme alla Legge italiana a partire da un dispositivo di firma anch'esso già a norma, garantendo la massima portabilità e facilità d'utilizzo. La struttura del lavoro è quindi la seguente:

- nel **Capitolo I** sono richiamati i fondamentali concetti e tecnologie che sono alla base della Firma Digitale e del Protocollo Informatico;
- nel **Capitolo II** vengono analizzati gli standard attualmente utilizzati e la legislazione vigente in materia;
- il **Capitolo III** descrive, partendo da un software già esistente, la necessità di una nuova architettura, di cui sono quindi illustrati il progetto e lo sviluppo;
- nel **Capitolo IV** è invece mostrato, quale esempio applicativo, come la Firma Digitale possa essere utilizzata nella gestione di un registro di Protocollo Informatico;
- nell'**Appendice A** è stato inserito il manuale d'uso (compilazione – e, quindi, installazione – e lista dei metodi) della classe Java di Firma Digitale;
- l'**Appendice B** è il corrispondente dell'Appendice A per il porting in C della classe di cui sopra;
- l'**Appendice C** contiene il manuale d'uso dell'utility TestCard, necessaria per visualizzare il contenuto di una smartcard;
- l'**Appendice D** è costituita da una porzione di codice sorgente necessaria per realizzare l'esempio del Capitolo IV

ed estremamente utile per impiegare in futuro l'architettura stessa;

- infine, nell'**Appendice E** è stata inserita la procedura per l'installazione sulla smartcard di un certificato[1].

Va considerato parte integrante di questa tesi l'allegato CD-ROM, contenente tutti i sorgenti e le documentazioni cui si fa riferimento.

[1] In realtà quest'appendice descrive una procedura che non fa parte di questo lavoro – che considera la smartcard con il certificato preinstallato un "input" –, ma si è ritenuto opportuno inserirla a seguito delle notevoli difficoltà ad installare il certificato sulla carta riscontrate da chi si è servito di quest'architettura. Contiene, tra l'altro, qualche utile consiglio per evitare blocchi della smartcard dovuti al crash di qualche applicazione.

CAPITOLO I
Concetti introduttivi

1.1 Fondamenti di crittografia

Si definisce *crittologia* quella scienza che studia tutto ciò che riguarda e gravita attorno alla cifratura dei messaggi al fine di renderli inintellegibili a chi non ne sia il legittimo destinatario. Si divide in due branche: la *crittografia*, che è la disciplina volta alla cifratura in senso stretto dei messaggi, e la *crittoanalisi*, che è, invece, volta ad individuare maliziosamente (ossia da parte di un eventuale soggetto che non sia il reale destinatario) un metodo per decifrare messaggi crittografati; vale la pena di notare che la crittoanalisi può essere utilizzata anche "a fin di bene", ossia per verificare la robustezza di un algoritmo crittografico.

Generalmente alla crittografia si richiede la risoluzione dei seguenti problemi:

- *riservatezza* o *segretezza* delle comunicazioni, ossia garanzia che solo il legittimo destinatario possa avere accesso alle informazioni;

- *integrità* delle informazioni stesse, ossia verifica della loro non alterazione durante il tragitto compiuto per giungere a destinazione;
- *autenticazione* della comunicazione, ossia certezza che il mittente e/o il destinatario siano effettivamente coloro che sostengono di essere.

In seguito allo studio di metodologie e tecniche per conseguire i suddetti obiettivi, il Legislatore vuole fornire uno strumento che consenta l'utilizzo della crittografia al fine di renderla del tutto equivalente (se non, dal punto di vista della sicurezza, addirittura migliore) alla firma "cartacea" oggi utilizzata per garantire la volontà di un soggetto di sottoscrivere un atto pubblico o privato:

- il *non ripudio*, che consiste nel riconoscimento giuridico di una sottoscrizione crittografica (che diverrà quindi la Firma Digitale), in modo che non si possa negarne l'apposizione avvenuta in precedenza.

Il funzionamento dei meccanismi crittografici si basa su due componenti distinte: un *algoritmo* ed una *chiave*. L'algoritmo è da considerarsi pubblico (esclusi, ovviamente, quelli coperti da segreto militare); il fatto che sia pubblico non pregiudica la sicurezza del crittosistema, anzi, si può affermare che quanto più un algoritmo è diffuso, tanto più è stato sviscerato al fine di scovarne punti deboli che ne consentano violazioni (o *crack*: è così

definita l'operazione di crittoanalisi con cui si decifra il messaggio senza avere la chiave o con cui si riesce a scoprire la chiave dato il messaggio).

Si comprende, quindi, come al centro della sicurezza del sistema vi sia la chiave, che è l'elemento del sistema che va tenuto (almeno in parte) segreto e che varia da soggetto a soggetto: quanto più è ampio lo *spazio di chiave* (ossia l'insieme delle chiavi possibili, pari alla potenza di due corrispondente alla lunghezza in bit della chiave stessa), tanto più risulta difficile violare l'algoritmo con *attacchi di forza bruta* (o *brute force attack*: viene così definito il tentativo di decrittare un messaggio provando tutte le possibili chiavi disponibili qualora un algoritmo non presenti "falle" che consentano un lavoro di crittoanalisi).

Si esamineranno ora le principali metodologie sia di crittografia che accessorie che permetteranno in seguito di definire e formalizzare meglio quel concetto di "sottoscrizione crittografica" già citato in precedenza e al quale è stato dato il nome di Firma Digitale.

1.2 Crittografia a chiave simmetrica

Sebbene la crittografia a chiave simmetrica non sia direttamente utilizzata nella firma digitale, la sua introduzione risulta interessante al fine di evidenziare i vantaggi che derivano invece dall' uso della crittografia a chiave asimmetrica.

Il principio della crittografia a chiave simmetrica è estremamente semplice: dato un messaggio M ed una chiave K e detto S il messaggio crittografato, si ha:

$S = F_{cr}(M,K)$ ed, analogamente, $M = F_{decr}(S,K)$

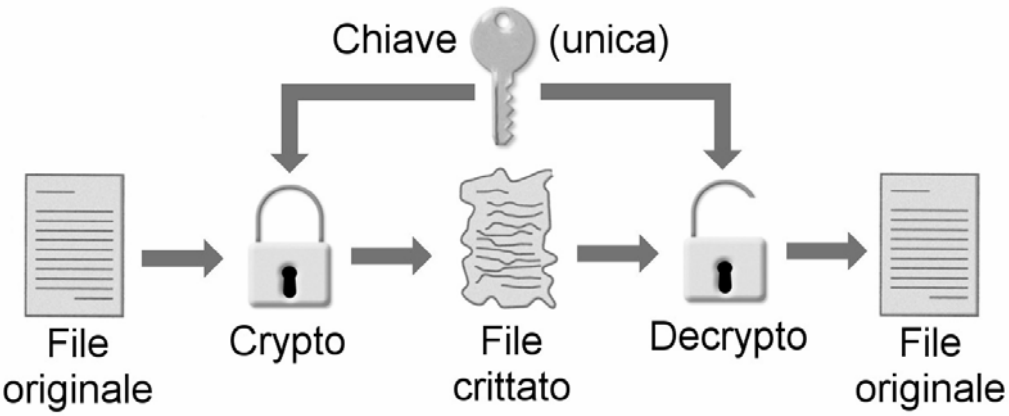

Figura 1.2: Crittografia a chiave simmetrica

Sostanzialmente, esistono due funzioni diverse, una per la crittazione ed una per la decrittazione (anche se per alcuni algoritmi le funzioni sono uguali – si pensi alla funzione di

crittografia più elementare, l'OR esclusivo: XOR(K,XOR(K,M)) = M). La caratteristica di questa tecnica è che la chiave è unica, ossia chi decritta deve avere la stessa chiave di chi critta: questo è chiaramente un punto debole dal punto di vista della sicurezza, in quanto diventa estremamente delicata la procedura di scambio delle chiavi (un eventuale *sniffer* – ascoltatore abusivo – sulla rete non avrebbe grosse difficoltà ad intercettare la chiave e a servirsene). Inoltre, è sempre possibile che una delle due parti distribuisca la chiave all'insaputa dell'altra o, peggio ancora, qualora la chiave sia distribuita fra più di due soggetti, che una delle parti generi fraudolentemente un messaggio firmandosi come una delle controparti, cosa possibile in quanto la chiave è la stessa per tutti. Al fine di evitare quest'ultimo problema (che è poi quello che interessa più direttamente) sarebbe necessario che ciascuna persona conservasse una coppia di chiavi per ogni corrispondente con cui è in contatto: se, ad esempio, i soggetti in causa fossero cinque, ciascuno dovrebbe avere quattro chiavi, il che vuol dire che nel sistema (già di per sé estremamente piccolo) dovrebbero circolare dieci chiavi (più in generale, detto N il numero di utenti, il numero di chiavi nel sistema sarebbe pari alla somma degli interi da 1 a N-1). Infine, sebbene tale tecnica abbia il vantaggio di richiedere un ridotto costo computazionale, è altrettanto vero che la stessa riduzione di costo computazionale rispetto ad altri algoritmi viene ereditata anche da parte di chi tenta di violare il canale mediante tecniche di brute force attack.

E' quindi inutile sottolineare che, per gli scopi di questa tesi, la crittografia a chiave simmetrica risulta inservibile. Tuttavia, vale

la pena di far notare che, date le caratteristiche viste sopra, essa viene ancora utilizzata con discreta frequenza laddove siano necessarie sessioni "*one shot*", ossia per le quali ogni chiave viene utilizzata una sola volta e per un ridottissimo periodo di tempo (ad es. un'autenticazione).

1.3 Crittografia a chiave asimmetrica

La vera svolta nella crittografia avviene con il passaggio da algoritmi a chiave simmetrica ad algoritmi a chiave asimmetrica. L'idea che è alla base dei crittosistemi a chiave asimmetrica è che la chiave di crittazione sia diversa da quella di decrittazione: così facendo è possibile distribuire la propria chiave di decrittazione (che prende quindi il nome di *chiave pubblica*) e mantenere segreta la chiave di crittazione (la *chiave privata*), il che risolve tutta una serie di problemi tipici della crittografia a chiave simmetrica:

- lo scambio delle chiavi non è più critico, anzi nella maggior parte dei casi le chiavi sono da considerarsi pubbliche e quindi non esiste più alcun problema in merito (per i rari casi in cui lo scambio debba rimanere segreto esiste un'applicazione – l'algoritmo di Diffie-

Hellman, che verrà analizzato in seguito – che risolve egregiamente il problema);

- il problema dell'autenticità del mittente viene immediatamente risolto, in quanto solo il titolare di quella chiave privata potrà aver generato il messaggio crittografato con la relativa chiave pubblica;
- analogamente si risolve il problema della riservatezza: poiché l'algoritmo è simmetrico dal punto di vista delle chiavi (ciò che viene crittato con la chiave privata va decrittato con quella pubblica, ma anche viceversa), è sufficiente crittografare un messaggio con la chiave pubblica affinché solo il titolare della corrispondente chiave privata possa leggerlo;
- infine, ogni soggetto dovrà detenere una sola coppia di chiavi (la propria): le chiavi pubbliche verranno iscritte in appositi registri, pubblicamente consultabili, dai quali potranno essere scaricate al momento opportuno (il discorso su questi registri sarà ripreso in seguito in maggiore dettaglio).

Il problema della crittografia a chiave asimmetrica venne formulato per la prima volta da W. Diffie e M. Hellman nel 1976; dette, infatti, P ed S le funzioni di de/crittazione applicate, rispettivamente, alle chiavi pubblica e privata ed M il messaggio, le quattro condizioni minime da soddisfare sono le seguenti:

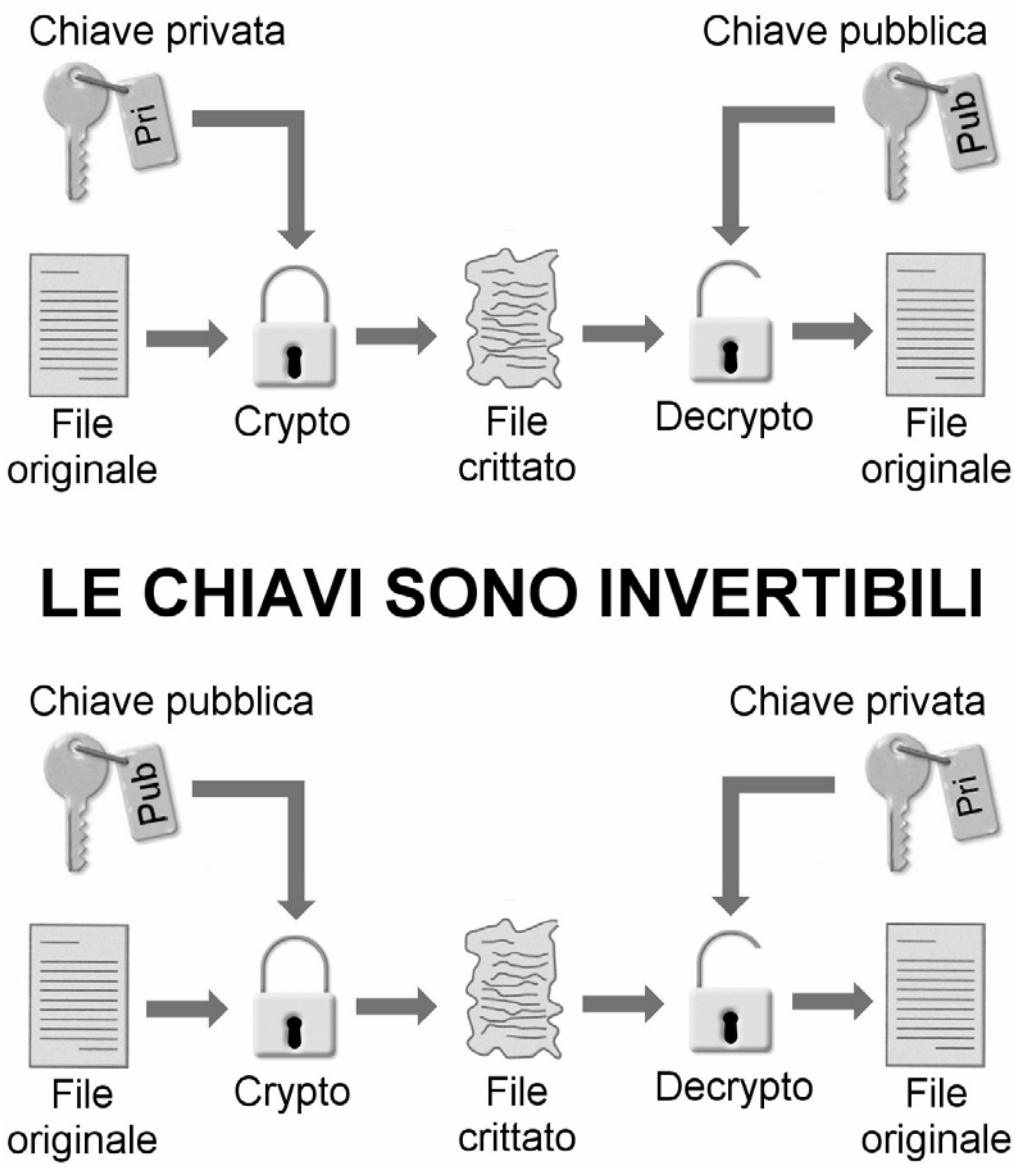

Figura 1.3: Crittografia a chiave asimmetrica

1) S(P(M)) = M per qualsiasi messaggio M;
2) tutte le coppie (S, P) sono distinte;
3) derivare S da P deve essere altrettanto difficile che leggere M;
4) sia S che P devono essere relativamente semplici da ottenere.

La prima regola garantisce la coerenza di un qualsiasi sistema crittografico, le seconde due ne garantiscono la sicurezza, l'ultima la fattibilità.

Nonostante fossero stati i primi a formalizzare il problema, Diffie ed Hellman non riuscirono a trovare un sistema in grado di risolverlo così come formulato; un metodo siffatto fu invece scoperto subito dopo da R. Rivest, A. Shamir e L. Adleman: il loro schema, divenuto famoso come il *sistema di cifratura a chiavi pubbliche RSA*, è basato sull'applicazione di algoritmi aritmetici a grandi valori interi ([PKCS#1]).

Nell'RSA la chiave P è in realtà una coppia di interi (N, p), mentre la chiave S è una coppia (N, s) con s segreto; questi numeri devono essere estremamente grandi e la lunghezza di N deve essere all'incirca il doppio di quella di s e p. I metodi utilizzati per cifrare e decifrare (la funzione, infatti, è unica) sono semplici: innanzi tutto si scompone il messaggio in numeri minori di N (i *blocchi*); per cifrare un blocco si calcola $C = P(M) = M^P \bmod N$ e, analogamente, per decifrare un blocco si calcola $M = S(C) = C^S \bmod N$. Benché operare con numeri estremamente grandi possa sembrare macchinoso, il fatto di essere interessati solamente al resto della divisione per N significa evitare che i numeri stessi esplodano: infatti $A^B \bmod N = ((((A*A)$

mod N) *A) mod N) ... mod N), il che vuol dire troncare ogni operazione esattamente alla lunghezza di N[1].

A questo punto la condizione 4) è soddisfatta, e risulta semplice forzare la 2), ma è ancora necessario assicurare che sia possibile scegliere N, p ed s in modo da soddisfare le condizioni 1) e 3). Si riesce a dimostrare che, scelti tre numeri primi casuali (s, x, y) di grosse dimensioni e tali che $s > x$ e $s > y$, ponendo $N = xy$ e p tale che $ps \bmod (x-1)(y-1) = 1$, risulta $M^{PS} \bmod N = M$ per ogni messaggio M (condizione 1)) e, inoltre, che derivare p da s risulta estremamente difficoltoso (condizione 3)).

Vale la pena di notare che, non essendo possibile una verifica in tempi ragionevoli sul fatto che x, y e s siano primi, si introduce il concetto di *numeri relativamente primi*, ossia di numeri per i quali è "quasi certa" la condizione di indivisibilità se non per uno e per se stessi; altre condizioni matematiche possono essere poste sui numeri di partenza al fine di evitare la generazione di quelle che in gergo vengono definite *chiavi deboli*.

Infine, una nota su un problema di sicurezza: si è già detto che gli algoritmi di crittografia asimmetrica operano per

[1] Sostanzialmente l'algoritmo si basa sul fatto, immediatamente verificabile, che il risultato di una moltiplicazione – quale è, tutto sommato, l'elevamento a potenza – comincia ad ottenersi dalle cifre meno significative ("da destra") e, quindi, l'operazione può essere interrotta appena si scavalcano i confini del modulo; per ottenere l'operazione inversa – la radice o il logaritmo, il cui risultato comincia ad ottenersi dalle cifre più significative ("da sinistra") – bisogna prima elaborare tutta quella parte di numero che non solo non ci serve, ma che è stata precedentemente distrutta dall'operazione di modulo.

blocchi, la cui dimensione è generalmente pari a quella della chiave; la crittografazione ripetuta di file di dimensioni inferiori al blocco (come, ad esempio, una *fingerprint*, che verrà illustrata più avanti) può, talvolta, facilitare il processo di crittoanalisi: per evitare questo problema generalmente si effettua il padding del blocco corto con dei dati aggiuntivi, magari derivati dal blocco stesso (ad esempio un CRC) e aggiungendo così un ulteriore livello di verifica.

1.3.1 L'algoritmo di Diffie-Hellman

Un'interessante tecnica correlata alla crittografia è l'algoritmo di Diffie-Hellman, che consente di generare una chiave di sessione in ambiente distribuito ed è pertanto utilizzato per risolvere il problema dello scambio segreto delle chiavi fra due soggetti ([PKCS#3]).

L'algoritmo è estremamente semplice e sfrutta le stesse proprietà dell'elevamento a potenza e del modulo utilizzate nella crittografia a chiave asimmetrica: in particolare, si basa sulla proprietà delle potenze $(A^B)^C = (A^C)^B$, che si riesce a dimostrare valida anche in presenza di un modulo, $(A^B \bmod D)^C = (A^C \bmod D)^B$.

Supponendo che due soggetti A e B vogliano generare una chiave di sessione segreta, si procede come segue:

1) A inizia lo scambio generando due numeri casuali p e q e comunicandoli a B;
2) A sceglie quindi un numero casuale XA e calcola il risultato dell'equazione $Y_A = Q^{XA} \bmod p$;
3) analogamente B sceglie un numero casuale XB e calcola il risultato dell'equazione $Y_B = Q^{XB} \bmod p$;
4) A e B si scambiano quindi Y_A e Y_B;
5) A calcola $Z_A = Y_B{}^{XA} \bmod p$;
6) B calcola $Z_B = Y_A{}^{XB} \bmod p$.

In base alle proprietà esposte sopra, si evince facilmente che $Z = Z_A = Z_B = q^{XA*XB} \bmod p$ e, quindi, i due soggetti dispongono di una chiave di sessione comune.

Per quanto fin qui detto, l'algoritmo di Diffie-Hellman non garantisce la sicurezza da un eventuale sniffer presente in rete: per aggirare questo problema l'intera sessione viene incapsulata in un livello di crittografia asimmetrica.

1.4 Le funzioni di *hash*

A questo punto la soluzione più semplice che si possa pensare per pervenire ad un sistema di Firma Digitale è quella di prendere il messaggio in blocco e di crittografarlo

con la chiave pubblica del mittente. Purtroppo tale tecnica non sempre è utilizzabile, in quanto il crittosistema può soffrire di notevoli limitazioni sulla potenza di calcolo (tale problema verrà discusso nel paragrafo relativo alle *smartcard*). Si rende pertanto necessaria una tecnica che, a partire da un messaggio, generi una sequenza di numeri (impronta o *fingerprint* o *digest*) molto più corta del messaggio stesso e che possa essere considerata (parafrasando la stessa definizione utilizzata in precedenza sui numeri primi per la crittografia) *relativamente univoca*, nel senso che dovrà essere estremamente difficile trovare un altro messaggio, specie se sensato, che generi la medesima sequenza. Le funzioni di *hash*, che già trovano altre applicazioni in informatica (come ad esempio l'indicizzazione di una tabella), risolvono egregiamente il problema; restano, tuttavia, da effettuare alcune puntualizzazioni che, per un'applicazione meno critica di una firma digitale, vengono normalmente trascurate.

Formalizzando il problema, una funzione di *hash* deve godere delle seguenti proprietà:

1) deve essere coerente: a messaggio uguale deve corrispondere uguale hash;
2) deve essere (o quanto meno apparire) casuale, per impedire l'interpretazione accidentale del messaggio originale;

3) deve essere (relativamente) univoca, ossia la probabilità che due messaggi generino il medesimo hash deve essere virtualmente nulla;
4) deve essere non invertibile: non deve essere possibile risalire al messaggio originale dalla sua fingerprint;
5) deve, infine, essere equiprobabile: ognuna delle possibili sequenze binarie che costituiscono l'hash deve avere la stessa probabilità di essere generata delle altre.

Come si può notare, nei tipici casi di utilizzo delle funzioni hash la sola proprietà necessaria è la 1), sebbene la 5) sia raccomandabile per motivi di efficienza. Nel caso di firma digitale tutte e cinque le proprietà diventano indispensabili, a tal punto che la 5) è addirittura imposta per Legge.

Attualmente sono vari gli algoritmi di hash utilizzati; tra i più comuni si segnalano:

- quelli della serie "Message Digest": gli ormai obsoleti MD2 e MD4 e il più recente MD5; quest'ultimo, in particolare, elabora il messaggio a blocchi di 512 bit per generare una fingerprint di 128 bit ([RFC1321]);
- il "Secure Hash Algorythm 1" (o SHA-1): derivato da MD4, elabora il messaggio a blocchi di 512 bit e genera una fingerprint di 160 bit ([SHA1]);
- il RIPEMD-160: elaborato da un gruppo di lavoro finanziato dall'Unione Europea (il RIPE – Race Integrity Primitives Evaluation), nasce come ideale sostituto di

MD5 e SHA-1, rispetto ai quali promette maggiore sicurezza; elabora il messaggio a blocchi di 256 bit e genera una fingerprint di 160 bit (ne esistono anche versioni a 128, 256 e 320 bit, ma in questi casi viene chiaramente specificato che all'aumentare della lunghezza dell'hash non aumenta il livello di sicurezza ottenuto) ([RIPEMD]).

Si riporta un confronto in termini prestazionali delle varie funzioni di hash:

Algoritmo	Cicli	Mbit/s	Performance relativa
MD4	241	191.2	1.00
MD5	337	136.7	0.72
SHA-1	837	55.1	0.29
RIPEMD-160	1013	45.5	0.24

Tabella 1.4: prestazioni delle varie funzioni di hash ottenute con compilatore ottimizzante su Pentium 90 in modalità nativa (non V86)

Val giusto la pena di notare che, sebbene MD5 sia al momento l'algoritmo di hash più diffuso, la Legge Italiana impone l'uso di SHA-1 o RIPEMD-160 (che è, al contrario, il meno supportato al momento della scrittura di questa tesi).

1.5 La Public Key Infrastructure (PKI)

Nel parlare della crittografia a chiave asimmetrica se ne sono comprese le innumerevoli potenzialità, date dal fatto che le chiavi pubbliche di ciascun soggetto possono essere custodite presso archivi accessibili a tutti. Appare evidente che, per riferirsi ad un determinato soggetto, la chiave pubblica deve essere accompagnata da tutta una serie di dati aggiuntivi che colleghino la chiave al suo legittimo titolare.

Restano quindi aperti due problemi:

a) definire cosa allegare nel messaggio insieme alla fingerprint e definirne un formato;
b) trovare un sistema in base al quale sia possibile garantire in maniera legalmente inoppugnabile che quella determinata chiave appartenga a quel determinato soggetto (e non ad altri) e che pertanto i documenti firmati con quella chiave risultino *non ripudiabili*.

Il primo problema viene risolto con i *certificati*: un certificato è una struttura dati che contiene, al proprio interno, sia la chiave pubblica che i dati del suo titolare. Pertanto, in un'operazione di firma digitale, viene allegato sempre, insieme alla fingerprint, anche copia del certificato di chi sta firmando il documento. Il formato di certificato più diffuso è l'X.509, che sarà trattato nel capitolo successivo.

Il secondo problema viene, a sua volta, risolto con la firma digitale stessa: l'idea di fondo è che nulla impedisce che un certificato possa a sua volta essere firmato. Pertanto un qualsiasi soggetto può presentarsi presso un Pubblico Ufficiale con il proprio certificato: una volta accertata l'identità del titolare, il Pubblico Ufficiale firma a sua volta il certificato, garantendo così la legalità dell'operazione.

Purtroppo, in questo caso, la soluzione pratica risulta un po' più complicata di quella squisitamente teorica: questo perché è necessario, innanzi tutto, un soggetto che si occupi della pubblicazione dei certificati e, non potendo pretendere che un Pubblico Ufficiale si occupi della gestione di un server, è evidente che questo soggetto debba essere una terza parte. Non è nemmeno proponibile l'idea di delegare all'ente preposto alla pubblicazione dei certificati il compito di identificare il soggetto, in quanto tale ente dovrebbe avere una presenza capillare che non giustificherebbe i costi di gestione; è, inoltre, da notare il particolare che i dati anagrafici sono soggetti a meno variazioni rispetto ad un certificato (che può essere revocato e ricreato un'infinità di volte) e che, quindi, le due strutture devono avere procedure aziendali e flussi informativi decisamente diversi.

Ci si orienta, pertanto, verso una divisione dei compiti fra chi verifica l'identità del soggetto (la *Registration Authority* o *RA*), con una struttura che possa anche fornire una capillare

assistenza agli utenti, e chi emette e pubblica il certificato (la *Certification Authority* o *CA*), che è sostanzialmente un'entità centralizzata che deve invece garantire standard tecnici e di sicurezza di massima qualità; tra i requisiti essenziali di una CA si possono infatti segnalare:

- il garantire la massima *disponibilità* delle informazioni: il sistema deve essere quanto più possibile *fault-tolerant*;
- il garantire la massima *sicurezza*: nessuno deve accedere a risorse cui non è autorizzato ad accedere (questo punto e il precedente sono inglobati nel *piano di sicurezza* della CA);
- la necessità di formulare una *politica di certificazione*, ossia un insieme di regole che indichino il campo di applicabilità di ogni certificato (non tutti i certificati vengono rilasciati per tutti gli scopi);
- l'obbligo di pubblicare un corretto ed esaustivo *manuale operativo*, che descriva esattamente tutte le procedure relative alla gestione dei certificati stessi;
- infine, ma non per questo meno importante, una CA deve garantire la massima *interoperabilità* con tutti i possibili client, il che implica automaticamente che la CA deve essere sempre costantemente aggiornata sulle ultime tecnologie in fatto di certificazione.

Appare, quindi, ancor più evidente che il compito di CA deve essere delegato ad una società con solide basi tecniche;

inoltre, una tale struttura è molto più orientata, per motivi di sicurezza, al concetto di centralità rispetto che a quello di capillarità. Ecco, quindi, giustificata ancora una volta la politica di divisione RA/CA, che tende ad affidare ad Enti Locali il compito dell'accertamento dell'identità e a società di importanza nazionale il compito di rilascio e gestione dei certificati.

Resta a questo punto in sospeso il problema di chi garantisce che una CA sia effettivamente quello che sostiene di essere; come ormai si sarà capito, la procedura di certificazione è iterabile: una CA può essere certificata o da un organismo pubblico preposto a questo compito (che sarà a sua volta una CA) o da un'altra CA di livello superiore che sia legalmente abilitata a certificare la CA in questione (quest'operazione è in realtà abbastanza rara: si verifica però con una certa frequenza quando una CA a livello nazionale certifica una propria sede locale). Pertanto, il certificato della CA (che dovrà essere reso pubblico) risulta a sua volta firmato da un'altra CA.

Naturalmente a monte di tutto ciò ci sarà una CA che non sarà certificata da nessun'altra: tale CA, che sarà ovviamente la CA pubblica per eccellenza, disporrà di un certificato "*self-signed*"; in Italia tale ruolo è ricoperto dall'Autorità per l'Informatica nella Pubblica Amministrazione (AIPA), che, però, alla data di scrittura di questo lavoro di tesi, non ha

ancora pubblicato il proprio certificato. Il certificato a monte di ogni catena di certificazione prende il nome di *certificato radice* (*root certificate*) e, analogamente, la CA a monte di ogni catena prende il nome di *CA radice* (o *root CA*); purtroppo, per ragioni politiche che vanno al di là di mere considerazioni tecniche, probabilmente non ci sarà mai una vera CA radice valida per tutte le nazioni[2] e quindi si verranno a creare una moltitudine di root CA, una per ogni nazione (addirittura alcuni paesi, come gli Stati Uniti, non hanno in programma una CA governativa da cui si originino tutte le CA autorizzate, ma autorizzano semplicemente le singole CA senza firmarne il relativo certificato).

Una siffatta infrastruttura prende il nome di *Infrastruttura a Chiave Pubblica* (o *PKI*) ed è formalmente definita in [X.509] come segue: "*L'insieme di hardware, software, persone, politiche e procedure necessario per creare, gestire, conservare, distribuire e revocare certificati, che si basa sulla crittografia a chiave pubblica*".

Formalmente, un'infrastruttura PKI consta di cinque tipi di componenti ([POLK]):

- le CA, che rilasciano i certificati;
- le ORA (Organizational Registration Authority), che si fanno garanti dei collegamenti fra chiavi pubbliche, identità dei titolari dei certificati e altri attributi

[2] E' sufficiente vedere cosa sta accadendo da un anno a questa parte nell'ICANN (che gestisce il Domain Name System, che è quindi una risorsa dal punto di vista legale molto meno importante di una Certification Authority) per rendersi conto che purtroppo una *root CA* a livello mondiale non la vedremo mai.

(rientrano in questa categoria sia le RA che le Authorization Authority che verranno discusse in seguito);
- i titolari dei certificati, che possono firmare i documenti;
- tutto il software necessario per apporre e convalidare firme digitali;
- i registri per la conservazione dei certificati (incluse le CRL, descritte più avanti).

Nella catena di certificazioni così come enunciata c'è da sottolineare un problema di sicurezza. Si supponga di ricevere un file firmato con un certificato utente rilasciato dalla CA XYZ, al cui interno ci siano le informazioni necessarie per recuperare il certificato di XYZ. Supponendo che XYZ sia stata certificata direttamente da AIPA, si potrebbe pensare che all'interno del certificato di XYZ ci siano le informazioni per ricavare il certificato dell'AIPA, rendendo così l'intera procedura automatizzabile. Ciò porta però ad una possibile violazione di sicurezza. Si supponga, infatti, di voler falsificare una firma digitale: si potrebbe generare un falso certificato AIPA e mettere su una copia fraudolenta del server AIPA; a partire da questo si può generare un falso certificato XYZ, firmato col certificato AIPA contraffatto, contenente, all'interno, l'indirizzo del clone del server AIPA; analogamente, basterebbe creare un server XYZ fasullo e generare, quindi, un falso certificato rilasciato da

XYZ. Così facendo l'utente che verifica la firma (che risale la catena delle CA dal basso) si troverebbe a recuperare il certificato di XYZ non dal vero server, ma da quello fraudolentemente creato (perché così indicato nel certificato utente) e, analogamente, a partire da questo arriverebbe non sul vero server di AIPA, ma sul server manomesso.

Ciò porta ad una semplice considerazione: nella catena di certificati, il certificato radice (in questo caso quello dell'AIPA) non va ottenuto mediante la trafila certificato-certificato, ma deve essere scaricato a parte, con un metodo diverso, ed installato separatamente. Così facendo, in caso di catena di certificazione manomessa, si perverrà prima o poi al certificato radice che, essendo preinstallato e non dovendo essere scaricato, non verificherà più i certificati sottostanti, segnalando così il tentativo di frode.

Vale, infine, la pena di notare che spesso in una macchina sono installati più certificati: sia i vari certificati radice validi nazione per nazione, ma anche, per motivi di efficienza, i certificati delle principali CA (una specie di cache, che, però, per i motivi di sicurezza descritti sopra è spesso gestita manualmente); così facendo si evita di dover instaurare una (o più) connessioni con le CA ogni volta che si deve verificare l'autenticità di un certificato. Vanno altresì installati (per ovvie ragioni) i certificati radice delle CA "ad uso interno"

(come la CA del CEDA per l'Università degli Studi di Napoli "Federico II").

1.5.1 Operazioni basilari sui certificati

Si analizzeranno ora le principali operazioni di un'infrastruttura PKI:

- *registrazione*: è il processo preliminare mediante il quale un soggetto si notifica presso una RA (o, dove non prevista, direttamente presso la CA);
- *inizializzazione*: è il momento in cui l'utente o il client ottengono i valori necessari per iniziare le comunicazioni con l'infrastruttura PKI (generalmente mediante uno scambio Diffie-Hellman);
- *certificazione*: è il processo mediante il quale la CA rilascia un certificato per la chiave pubblica del soggetto;
- *aggiornamento*: è così definita l'obbligatoria generazione di una nuova coppia di chiavi una volta scaduta la precedente (ogni certificato ha, infatti, un proprio periodo di validità);
- *verifica di un certificato*: l'operazione mediante la quale chi riceve il certificato verifica la firma digitale appostavi dall'ente certificatore, realizzata recuperando il certificato del certificatore e verificandolo a sua volta, fino a giungere alla root CA (che, come già detto, ha il certificato preinstallato in macchina). Si noti che, per motivi di efficienza, talvolta si usa allegare in una firma digitale non

solo il certificato utente, ma anche quello della CA che lo ha firmato fino ad arrivare ad un livello immediatamente inferiore a quello della CA radice;

- *revoca di un certificato*: l'operazione mediante la quale un certificato viene annullato, o per scadenza o perché venuti a mancare i presupposti per la sua validità (manomissione, cambio di identità giuridica del soggetto, ecc.).

Sulla revoca è necessario qualche altro approfondimento: si noti che, dovendo la CA garantire la verificabilità delle firme antecedenti la revoca anche in tempi successivi alla revoca stessa, si dovrà tener traccia di tutti i certificati emessi, anche se revocati; pertanto la CA dovrà suddividere i certificati in due liste: quella dei certificati attivi e quella dei certificati revocati (la *Certificate Revocation List* o *CRL*).

Un interessante problema è quello di chi debba essere autorizzato a richiedere la revoca di un certificato; mentre per i privati cittadini è il titolare stesso del certificato che detiene la facoltà di richiederne la revoca (più eventualmente l'anagrafe, in caso di decesso), nel caso di società o iscrizione ad albi professionali, non è ben chiara la posizione di colui che possa richiedere la revoca di un certificato. Si pensi, ad esempio, all'amministratore delegato di una società: il suo mandato può decadere per dimissioni (e in questo caso sarebbe il legittimo titolare a chiedere la revoca del certificato), per sfiducia del consiglio di amministrazione o per provvedimento esecutivo di magistratura o altri enti pubblici. In questi casi la procedura di revoca diventa terribilmente macchinosa e richiede il ricorso ai

canali "tradizionali" con le relative lungaggini, con l'effetto immediato che il titolare del certificato può continuare a firmare documenti per tutto il periodo che intercorre tra la sua formale decadenza ad ordine del CdA o di un giudice e la materiale revoca del certificato. E' questo uno dei problemi ancora irrisolti della Firma Digitale.

1.6 La firma digitale

Si è a questo punto elencato il background delle tecnologie necessarie per poter finalmente parlare concretamente di Firma Digitale (anche se in questa sede verranno esaminati i soli concetti base, rinviando ai capitoli successivi per ulteriori dettagli legali e tecnici).

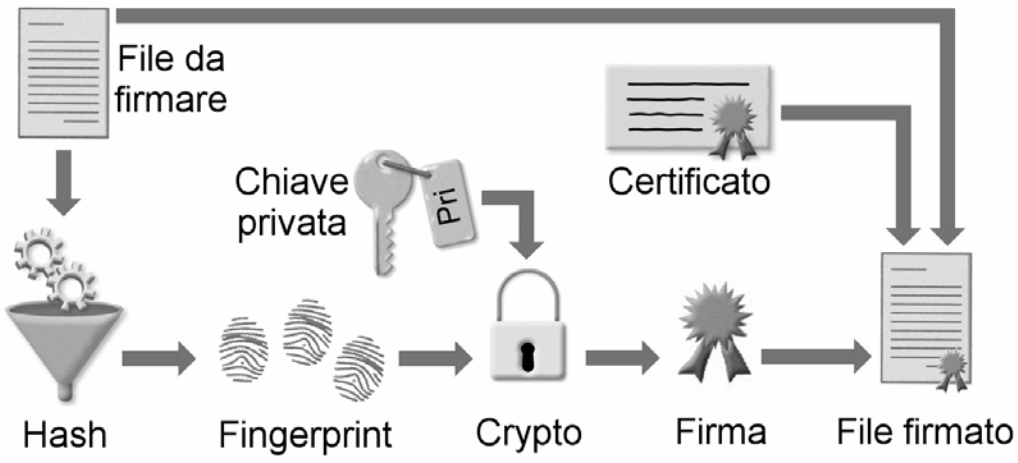

Figura 1.6.1: Firma digitale

Si può quindi descrivere il procedimento utilizzato per apporre una firma digitale ad un documento (o gruppo di documenti):

1) il documento viene generato e salvato in un formato possibilmente pubblico (laddove il formato non è imposto per Legge questa è una regola dettata più dal buon senso che da norme specifiche: la firma digitale è, infatti, applicabile a qualsiasi tipo di file binario);
2) viene generata una fingerprint del documento mediante un algoritmo di hash quanto più possibile diffuso (e, se necessario, a norma di Legge[3]);
3) la fingerprint viene crittografata con la chiave privata del soggetto e corredata con il suo certificato, più, eventualmente, (per motivi di efficienza) quello della CA;
4) il tutto viene quindo inserito in un *package file*: non ci sono particolari limitazioni su questo punto, salvo il fatto che, qualora i file da firmare siano più d'uno (e in questo caso si prevede una fingerprint unica per l'intero blocco) essi devono essere estratti nello stesso ordine in cui sono stati inseriti (la fingerprint di A+B non è la stessa di B+A) e, pertanto, sono esclusi tutti quei programmi di packaging che non rispettano questa condizione.

[3] Nulla vieta infatti l'utilizzo di algoritmi, tecniche o chiavi non a norma di Legge per le certificazioni interne; l'importante è che, qualora un documento dovesse uscire dall'organizzazione, ci sia un responsabile (come ad es. il Responsabile di Protocollo) con una firma valida a norma di Legge che lo controfirmi.

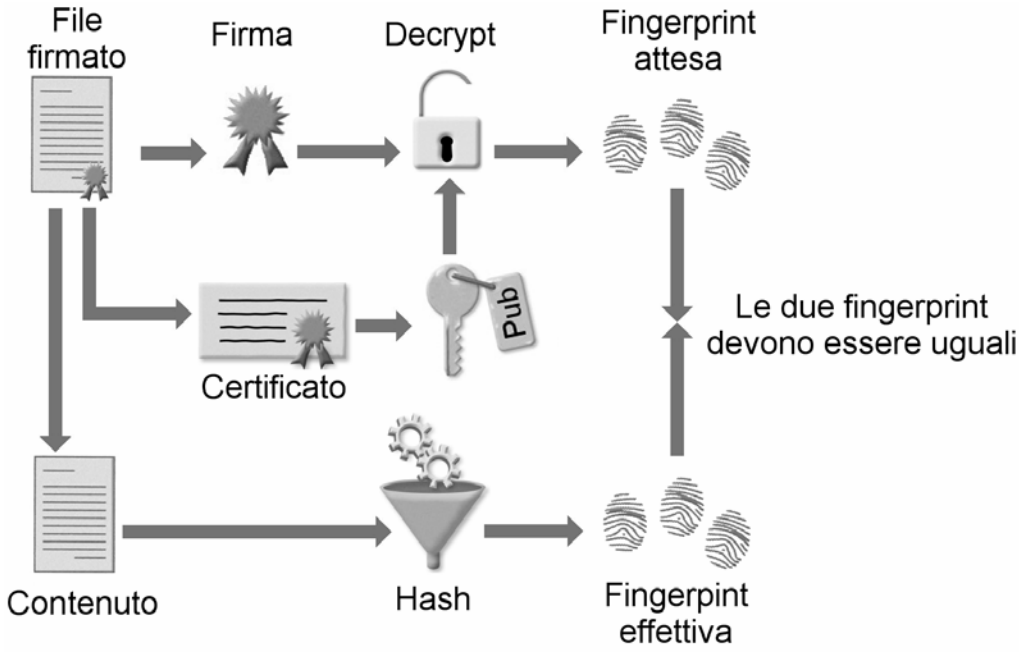

Figura 1.6.2: Verifica di una Firma Digitale

Il processo di verifica della firma da parte del destinatario è chiaramente simile al precedente:

1) si identifica l'algoritmo di hash utilizzato;
2) i documenti vengono estratti dal package e si calcola la fingerprint dell'intero blocco;
3) si estrae il certificato utente e lo si verifica;
4) si decritta la fingerprint inserita nella firma con la chiave pubblica del certificato e si verifica con quella ottenuta: se nulla è stato alterato, le due fingerprint devono risultare identiche.

1.7 Dispositivi di firma

Finora si è genericamente parlato di algoritmi di crittografia e relative chiavi, supponendo che tutte le operazioni venissero svolte dallo stesso calcolatore utilizzato per produrre i documenti. Basta però considerare che la Firma Digitale, rispetto a quella cartacea, soffre del problema di un possibile furto della chiave privata per rendersi conto che il fatto di detenerla su di un calcolatore risulta decisamente poco sicuro.

Per questa ragione la Legge impone che la firma debba essere generata da un dispositivo apposito; tale dispositivo dovrà generare al proprio interno la coppia di chiavi e non permettere in alcun modo che la chiave privata esca dal dispositivo stesso. Ne deriva, pertanto, che un tale dispositivo dovrà essere in grado di applicare al suo interno gli algoritmi di crittografia più diffusi, risultando, quindi, decisamente più complesso di una semplice memoria.

Al momento la tecnologia principe per queste applicazioni è quella delle *smartcard*, costituite da un supporto di plastica sul quale è inciso un apposito microchip. Il microchip può essere del tipo ASIC (Application Specific Integred Circuit, come avviene ad esempio per le carte realizzate da InCard), nel qual caso verrà realizzato appositamente per la Firma Digitale, o un microcontroller (un chip programmabile

contenente un microprocessore e una certa quantità di RAM/ROM/EEPROM, oltre magari ad un sistema operativo – come avviene, ad esempio, nel caso delle carte Schlumberger), caso in cui la smartcard verrà realizzata da terze parti e solo programmata (chiaramente con un procedimento one-shot) da chi realizza il dispositivo di firma.

Le smartcard vengono realizzate in due versioni: normale e *contactless*. Le prime prevedono che fra la carta e il dispositivo di lettura ci sia un contatto meccanico, che fornisca, tra l'altro, l'alimentazione; le seconde, invece, operano (e ricevono energia) in radiofrequenza, senza quindi contatto e necessità d'inserimento in un apposito lettore. Per la Firma Digitale, salvo rarissime applicazioni, una più economica smartcard tradizionale è più che sufficiente.

Le dimensioni delle smartcard, la posizione del relativo microchip, la piedinatura, le tensioni di alimentazione e gli elementi basilari del protocollo di colloquio con il lettore sono fissate da un apposito standard (ISO 7816). Lo standard definisce solo la parte seriale del collegamento fra smartcard e lettore (parlando in termini di pila ISO-OSI, diremmo che ne definisce il livello 1): ciò consente ad ogni produttore di realizzare smartcard con le più svariate funzioni, in quanto il formato del pacchetto di comunicazione è lasciato al produttore stesso. Per quanto riguarda questa tesi, è importante sottolineare che, indipendentemente dal

dispositivo, esistono alcuni standard per l'accesso ad alto livello alle funzioni della smartcard; in particolare, per le smartcard ad uso crittografico (talvolta definite anche *cryptocard*) gli standard più diffusi sono PKCS#11 e PKCS#15.

Vista la delicatezza dei dati trattati, è da notare che, durante la realizzazione delle cryptocard ad uso Firma Digitale, particolare attenzione viene posta nel tentare di impedire violazioni "fisiche" della carta, come ad esempio lo scoperchiamento del microchip al fine di poterne leggere il contenuto al microscopio elettronico[4]; tali carte vengono definite *tampered proof cards* (letteralmente: carte a prova di scoperchiamento). Tale protezione viene ottenuta immergendo il chip in particolari materiali che, al contatto con l'aria, con la luce o con le radiazioni emesse da un microscopio elettronico, mandino in cortocircuito il gate fluttuante della EEPROM, cancellandone immediatamente il contenuto.

Per quanto possano risultare pratiche, economiche e facilmente trasportabili, purtroppo le smartcard non sono esenti da difetti. I principali sono i seguenti:

- la smartcard è facile da trafugare: per aggiungere un ulteriore livello di protezione vi si associa normalmente

[4] Alcuni hacker sostengono addirittura di essere riusciti a leggere il contenuto della EEPROM di una smartcard basandosi sulle fluttuazioni della potenza richiesta in alimentazione (!). Sebbene la cosa non sia mai stata pubblicamente dimostrata (e, a detta di molti, abbia più il sapore di una favola che non una vera rilevanza), i vari costruttori hanno inserito all'interno delle smartcard un circuito che provvede a mantenere costante la potenza assorbita deviando su una resistenza la porzione di corrente non utilizzata.

un codice identificativo (PIN) al quale vengono solitamente applicate le classiche misure di sicurezza impiegate, ad esempio, nei Bancomat (digitando più di N volte il PIN errato la carta si blocca). Addirittura, alcuni produttori (tra i quali si cita CompEd) realizzano smartcard che sono abbinate anche a sensori di dati biometrici, come, ad esempio, l'impronta digitale;

- la dissipazione di calore effettuata dal centimetro quadrato di metallo posto sul chip è assolutamente insufficiente: per questa ragione i chip montati sulle smartcard sono affetti da grosse limitazioni prestazionali;
- i contatti del lettore tendono ad ossidarsi e a sporcare o graffiare la smartcard;
- infine, la smartcard contiene una EEPROM, che è un dispositivo che, specie dopo parecchi cicli di lettura/scrittura, rischia di perdere qualche colpo; tale problema viene, però, minimizzato dalla Legge, che impone il costante aggiornamento della lunghezza delle chiavi, il che porta inevitabilmente ad una frequente sostituzione della smartcard stessa.

1.8 Certificati di attributo

Fino ad ora si è parlato di firma digitale associando la firma alle singole persone. Già nel paragrafo relativo alle operazioni sui certificati è stato illustrato un tipico problema riguardante la revoca dei certificati, qualora il certificato sia rilasciato ad una persona giuridica (es. amministratore di società).

E' allo studio la possibilità di introdurre, oltre alla RA e alla CA, una terza autorità: l'*Autorità di Autorizzazione* (o *Authorization Authority* o *AA*). Tale autorità avrà lo scopo di certificare non più l'identità della persona, ma la sua autorizzazione a compiere determinate operazioni in base alla sua personalità giuridica.

Ad esempio, un professore di Ingegneria potrebbe apporre tre diverse firme: una come privato cittadino, una come Ingegnere iscritto all'Albo e una terza come docente; senza l'introduzione di estensioni relative alle mansioni, sarebbero necessari tre diversi certificati. La soluzione proposta per risolvere il problema è la creazione di un nuovo certificato (da allegarsi al precedente), noto come *Certificato di Attributo* (o *Attribute Certificate* o *AC*): così facendo è infatti possibile rilasciare un solo certificato identificativo per la persona, delegando all'AC la decrizione delle autorizzazioni rilasciate al titolare (un po' come, del resto, avviene nella realtà: ogni

persona ha una sola carta d'identità e un documento o timbro attestante l'iscrizione ad un albo professionale o le mansioni all'interno di un'azienda). Ovviamente, l'AC risulta gestito con le stesse identiche tecniche del certificato normale e risulta, quindi, firmato dall'AA.

Le motivazioni che sono alla base dell'introduzione di una terza autorità di certificazione sono simili a quelle che hanno portato alla distinzione fra RA e CA:

- le informazioni presenti nell'AC possono variare più velocemente rispetto a quelle contenute nel certificato normale;
- si preferisce non sobbarcare la CA anche del compito di gestire le informazioni relative alle autorizzazioni, che possono essere meglio gestite da un'autorità appositamente preposta, magari in diretto contatto con tutti quegli organi giuridico-amministrativi che possono modificare la personalità giuridica di un soggetto (Tribunali, Camere di Commercio, ecc.).

Sebbene risultino ormai completamente standardizzati, i certificati di attributo sono al momento inutilizzati. La principale causa di tale mancato utilizzo è che, a tutt'oggi, non esiste ancora né un software di certification authority né un client che li supporti.

CAPITOLO II
Gli standard e la legislazione sulla Firma Digitale

2.1 Gli standard di codifica: X.208 e X.209

Nell'introdurre gli standard che attualmente riguardano la Firma Digitale si inizierà partendo "dal basso", ossia dagli standard più semplici per poi arrivare a quelli più complessi. Dovendo trattare strutture dati la cui prima caratteristica è quella di risultare indipendenti dalla piattaforma, si comincerà, pertanto, dal documento relativo alla rappresentazione delle strutture stesse, che è poi quello cui fanno riferimento tutti i successivi.

Tale standard è, in realtà, scomposto in due documenti diversi: il primo contiene l'insieme di regole necessario per formulare tipi e strutture dati secondo un linguaggio formale e non ambiguo, mentre il secondo si occupa di introdurre una forma di codifica in ottetti (byte) che risulti indipendente dal processore e dal sistema operativo utilizzato. Avendo però entrambi una rilevanza marginale nel problema trattato, se ne darà qui un rapido excursus, limitandosi alle sole parti di interesse.

X.208 definisce l'*Abstract Syntax Notation One*, spesso abbreviata in *ASN.1* ([X.208], ma una sintesi limitata agli argomenti di interesse in questa tesi si può trovare in [KALISKI]). Tale notazione sintattica consente di esprimere strutture dati anche estremamente complesse a partire da pochi semplici tipi fondamentali: l'intero, la stringa di bit, la stringa di ottetti, la stringa di caratteri (in varie versioni, a seconda del set di caratteri usato), il valore nullo e il tempo UTC[5].

[5] Una curiosità sul tipo UTCTime: nel tempo UTC (Universal Coordinated Time) propriamente detto è prevista la possibilità che, durante l'anno solare, due minuti (e precisamente l'ultimo minuto del 30 Giugno e l'ultimo minuto

In aggiunta ai tipi fondamentali, sono introdotti due tipi strutturati: la sequenza ordinata (in due varianti, quella con uno o più elementi e quella che prevede anche la possibilità di sequenza vuota) e il set, o sequenza non ordinata (con le medesime due varianti).

Vengono infine definiti due tipi di aggregatori: il *CHOICE*, che prevede la possibilità di unire una o più alternative e l'*ANY*, che indica un tipo generico arbitrario o scelto da una particolare lista.

A partire da tali tipi base l'utente può costruire una sua personale notazione dati creandone di propri; tale creazione può essere realizzata implicitamente, ossia semplicemente rinominando un tipo base, o esplicitamente, includendo il tipo base in una sequenza o un set e completandolo con altre informazioni.

Tutto ciò è possibile in seguito ad un serie di regole di formulazione sintattica che, visto lo scarso interesse che tale forma riveste in questo lavoro, non verrà descritta in questa sede.

Risulta invece interessante la descrizione dello standard X.209, che introduce le regole di codifica in ottetti dell'X.208 e che prende il nome di *Basic Encoding Rules* o *BER* ([X.209], ma anche qui risulta utile la sintesi di [KALISKI]).

BER assegna innanzi tutto un valore numerico a sei bit a ciascuno dei tipi base e strutturati introdotti in ASN.1; i due bit rimanenti sono utilizzati per definire se l'oggetto in questione è da considerarsi universale, specifico per l'applicazione, per il contesto o privato. Essendo tutti i tipi a lunghezza variabile è necessario codificare per ciascun oggetto la sua dimensione e ciò può essere fatto secondo due forme:

- la codifica a lunghezza definita, che fa seguire al byte corrispondente al tipo uno o più byte contenenti la lunghezza stessa (precisamente uno solo se la

del 31 Dicembre) possano avere durata leggermente differente per compensare il ritardo o l'anticipo accumulato dal tempo sulla Terra, come previsto dalla teoria della relatività; pertanto, nel tempo UTC il range dei secondi non è semplicemente 0-59, ma possono esserci casi di minuti con 60, 61 o 62 secondi. ASN.1 non prevede che i secondi possano eccedere 59: pertanto, quello che qui viene definito UTC è semplicemente un abuso di linguaggio per definire un tempo in fuso orario GMT.

lunghezza è minore di 128 byte e 1+L/256 – dove L è la lunghezza – altrimenti);

- la codifica a lunghezza non definita: in questo caso la lunghezza viene posta a zero e il blocco di informazioni viene chiuso con un tag apposito.

Vengono, infine, stabilite le codifiche di tutti i tipi base (ad esempio, gli interi sono in formato MSB-first).

Resta però aperto il problema di come distinguere i tipi definiti dall'utente in maniera esplicita (quelli definiti in maniera implicita vengono riconosciuti semplicemente dalla loro posizione nella struttura che li contiene). Tale problema viene risolto introducendo un nuovo tipo, l'identificatore.

Ad ogni società che ne fa richiesta, l'ISO assegna un codice identificativo BER[6]; a partire da tale codice potranno essere costruiti tutti i tipi definiti dalla società stessa semplicemente postponendo al codice una sequenza il cui formato è a cura di chi definisce il tipo. Giusto per citare un caso di diretto interesse, RSA genera gli identificativi postponendo al proprio codice il numero di standard PKCS in questione e quindi un progressivo (seguito eventualmente da un "subtype"): ad esempio, l'algoritmo di crittografia RSA, definito in PKCS#1 come primo oggetto, è: {Codice RSA} {PKCS#1} {Oggetto 1} = 2A864886F70D 01 01.

Formato pertanto l'identificativo, ogni oggetto creato dall'utente verrà costruito come sequenza o set of il cui primo elemento sarà l'identificativo stesso, seguito quindi dai dati.

Si può notare immediatamente che la codifica BER non è univoca: ogni oggetto può, ad esempio, essere codificato con lunghezza definita o indefinita. Siccome in certi casi questa ambiguità è da evitare, è stato definito un sottoinsieme di BER, le *Distinguished Encoding Rules* o *DER*, che stabilisce un metodo univoco per la codifica dei dati[7].

[6] Il codice è a sua volta strutturato, contenendo il numero della struttura rilasciante il codice, la posizione del richiedente nella struttura e il codice della nazione di appartenenza.

[7] Sostanzialmente le DER sono identiche alle BER, salvo l'escludere la codifica a lunghezza indefinita e l'imporre sempre la codifica a lunghezza definita di dimensione minore.

2.2 I certificati: X.509

Come già accennato nel capitolo precedente, il certificato è quell'oggetto che collega la chiave con il soggetto che la detiene legalmente; è, quindi, un oggetto di primaria importanza e, come tale, va a sua volta firmato da chi lo rilascia (la CA). I certificati (non solo quelli a chiave pubblica, come si vedrà alla fine di questo paragrafo) sono descritti nel documento [X.509]. Si descrive ora la struttura ASN.1 di un certificato a chiave pubblica, servendosi anche della figura 2.2, che cerca di sintetizzare il tutto graficamente:

```
Certificate ::= SIGNED {
  SEQUENCE {
    version [0] Version DEFAULT v1,
    serialNumber CertificateSerialNumber,
    signature AlgorithmIdentifier,
    issuer Name,
    validity Validity,
    subject Name,
    subjectPublicKeyInfo SubjectPublicKeyInfo,
    issuerUniqueIdentifier [1] IMPLICIT UniqueIdentifier
          OPTIONAL[V2],
    subjectUniqueIdentifier [2] IMPLICIT UniqueIdentifier
          OPTIONAL[V2],
    extensions [3] Extensions OPTIONAL[V3]}

NOTA: [V2] = da versione 2 in poi, [V3] = da versione 3 in poi
```

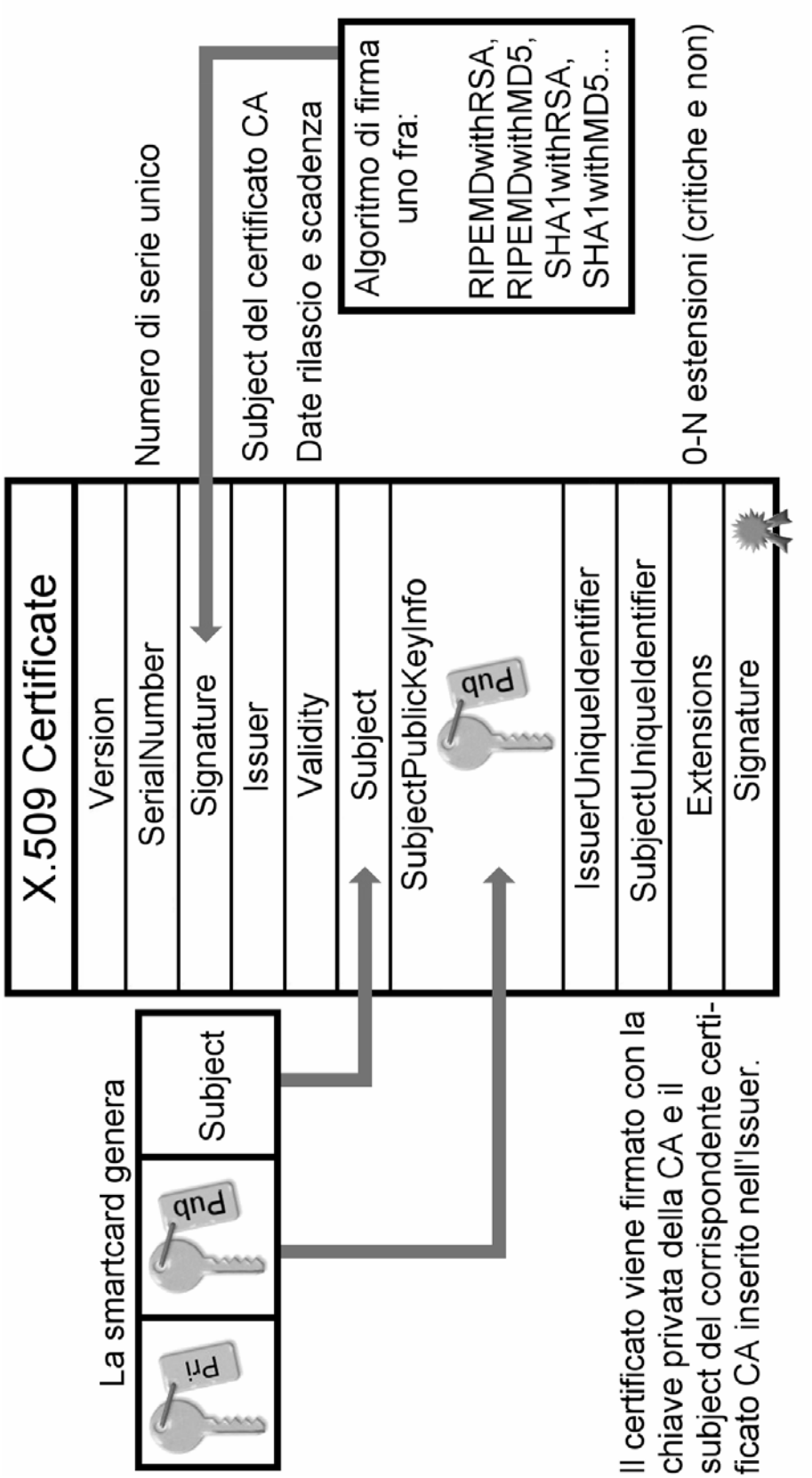

Figura 2.2: Certificato X.509

Si analizzano, quindi, i campi del certificato:

- *version*: è un intero indicante la versione del certificato (attualmente v3);

- *serialNumber* è un intero assegnato dalla CA ad ogni certificato in modo che non ve ne siano due uguali;
- *signature* contiene l'identificativo dell'algoritmo utilizzato per generare la signature; tale algoritmo è "composito", nel senso che non si tratta semplicemente di un algoritmo di hash (come, ad es. SHA-1) o di uno di crittografia (es. RSA), ma di un identificativo appositamente assegnato a una combinazione hash+crittografia (tipico è sha-1WithRSAEncryption); in realtà il campo può contenere non solo l'algoritmo utilizzato per firmare il certificato in questione, ma anche una lista di algoritmi supportati dalla CA;
- *issuer* è il *subject* del certificato con cui la CA firma il certificato in questione; generalmente tale subject è però costituito da una sequenza di dati (per lo più stringhe) che identificano la CA che ha emesso il certificato, per cui può essere anche considerato un identificativo della CA stessa;
- *validity* è l'intervallo temporale di validità del certificato;
- *subject* è il subject che associa il certificato alla coppia di chiavi (la procedura prevede, infatti, prima la generazione della coppia di chiavi e quindi del certificato, con lo stesso subject delle chiavi);
- *subjectPublicKeyInfo* è il campo che contiene fisicamente la chiave pubblica;
- *issuerUniqueIdentifier* contiene un intero (generalmente un numero di serie) che consente di identificare univocamente l'issuer in caso di ambiguità (ad es. due CA con lo stesso nome);
- *subjectUniqueIdentifier*, come nel campo precedente, contiene un numero di serie per identificare univocamente la coppia di chiavi in caso di omonimia;
- *extension*, novità introdotta nella v3 di X.509, contiene le possibili estensioni del certificato, che, data la loro importanza, meritano un approfondimento a parte.

Le estensioni sono così definite:

```
Extension ::= SEQUENCE {
  extnId EXTENSION.&id ({ExtensionSet}),
```

```
critical BOOLEAN DEFAULT FALSE,
extnValue OCTET STRING }
```

Le estensioni sono campi aggiuntivi al certificato non sempre previsti dallo standard; non vengono inserite in un ordine specifico e quindi, analogamente a quanto visto nel caso di BER, dispongono di un identificatore che consenta, in caso di estensione non nota al programma che riceve il certificato, di ignorarla e passare oltre. Fondamentale, in questo caso, diventa però il flag *critical*: con tale flag settato l'intero certificato dovrà essere considerato non valido se l'applicazione non è in grado di gestire l'estensione in questione (cosa che non avverrebbe, invece, col flag impostato a false).

Le estensioni si dividono nei seguenti gruppi principali:

- *key and policy information extensions*, che contengono informazioni sullo scopo per il quale la chiave o il certificato possono essere usati ed eventuali informazioni accessorie che consentano di distinguere univocamente parti del certificato, della chiave o del percorso di certificazione qualora ci siano ambiguità;
- *subject and issuer information extensions*, che consentono di definire nomi alternativi (alias) per subject e issuer del certificato, oltre ad alcuni attributi aggiuntivi;
- *certification path constraints extensions*, che contengono informazioni sul percorso di certificazione del certificato stesso e, in particolare, se il titolare del certificato è una CA; in quest'ultimo caso, possono essere specificati i vincoli imposti a nome e policy dei certificati rilasciati (ad esempio si può stabilire che il subject di tutti i certificati rilasciati da ABC cominci con "ABC Certificate no. ").

Alla luce di quanto visto nel capitolo precedente, è scontato aggiungere che il certificato risulta terminato da una stringa di ottetti contenente la sua stessa firma.

Infine, [X.509] definisce anche tutta una serie di strutture accessorie ai certificati; tra queste vanno citate, anche se non analizzate in questa sede, la *CRL* (*Certificate*

Revocation List – o, più precisamente, la *Certificate List*, di cui la CRL è una specializzazione –, che è, sostanzialmente, una lista di numeri di serie di certificati e delle relative date di revoca), e l'*AttributeCertificate* (la cui definizione ricorda quella di un certificato normale, ma il cui corpo è costituito da una lista di attributi il cui funzionamento ricalca quello delle estensioni, delle quali il certificato di attributo è comunque corredato).

2.3 La codifica dei dati firmati: PKCS#7

Al termine di ogni pacchetto di Firma Digitale deve essere contenuta, com'è ovvio, la firma stessa; come già detto nel capitolo precedente, la Legge impone che la firma debba contenere anche il certificato di chi sta firmando, pertanto è necessaria una struttura un po' più complessa della semplice sequenza di ottetti costituenti il digest crittografato e che, al fine di evitare manipolazioni, sia a sua volta firmata. Si rende quindi necessaria la definizione di un oggetto che contenga almeno un certificato e che risulti intrinsecamente firmato.

Lo standard che introduce una siffatta struttura è il Public Key Cryptographic Standard number 7 (in breve PKCS#7, descritto in [PKCS#7]), rilasciato, nella sua prima versione, da RSA Data Security alla fine degli anni '80 (l'attuale versione, la 1.5, è datata Novembre 1993).

Tale standard introduce quattro tipi di blocchi di dati:

- *Data*, che corrisponde a dei semplici dati senza alcun trattamento aggiuntivo;
- *SignedData*, ossia un blocco dati con relativa firma (e certificato/i);
- *EnvelopedData*, costituito da un blocco di dati crittografato;
- *SignedAndEnvelopedData*, che riunisce i due casi precedenti con un blocco di dati sia firmato che crittografato.

Dei quattro tipi, quello di interesse per questo lavoro è il secondo (SignedData), pertanto lo si analizzerà in dettaglio.

Il tipo SignedData nasce per contenere:

- un blocco di dati da firmare;
- un blocco di attributi da firmare;
- un blocco di attributi che non è necessario firmare;
- dei certificati X.509, tra cui quello con cui è firmata la struttura;
- delle CRL;
- tutti i dati relativi alla firma di se stesso.

Eccezion fatta per l'ultimo, ciascuno dei dati sopra elencati è opzionale, perciò è possibile realizzare strutture PKCS#7 mancanti di alcune parti. In particolare si vedrà tra poco che esistono forme degeneri di PKCS#7 la cui importanza è fondamentale nell'ambito della Firma Digitale.

Si analizza, pertanto, la definizione ASN.1 completa del tipo SignedData, servendosi anche della Figura 2.3.

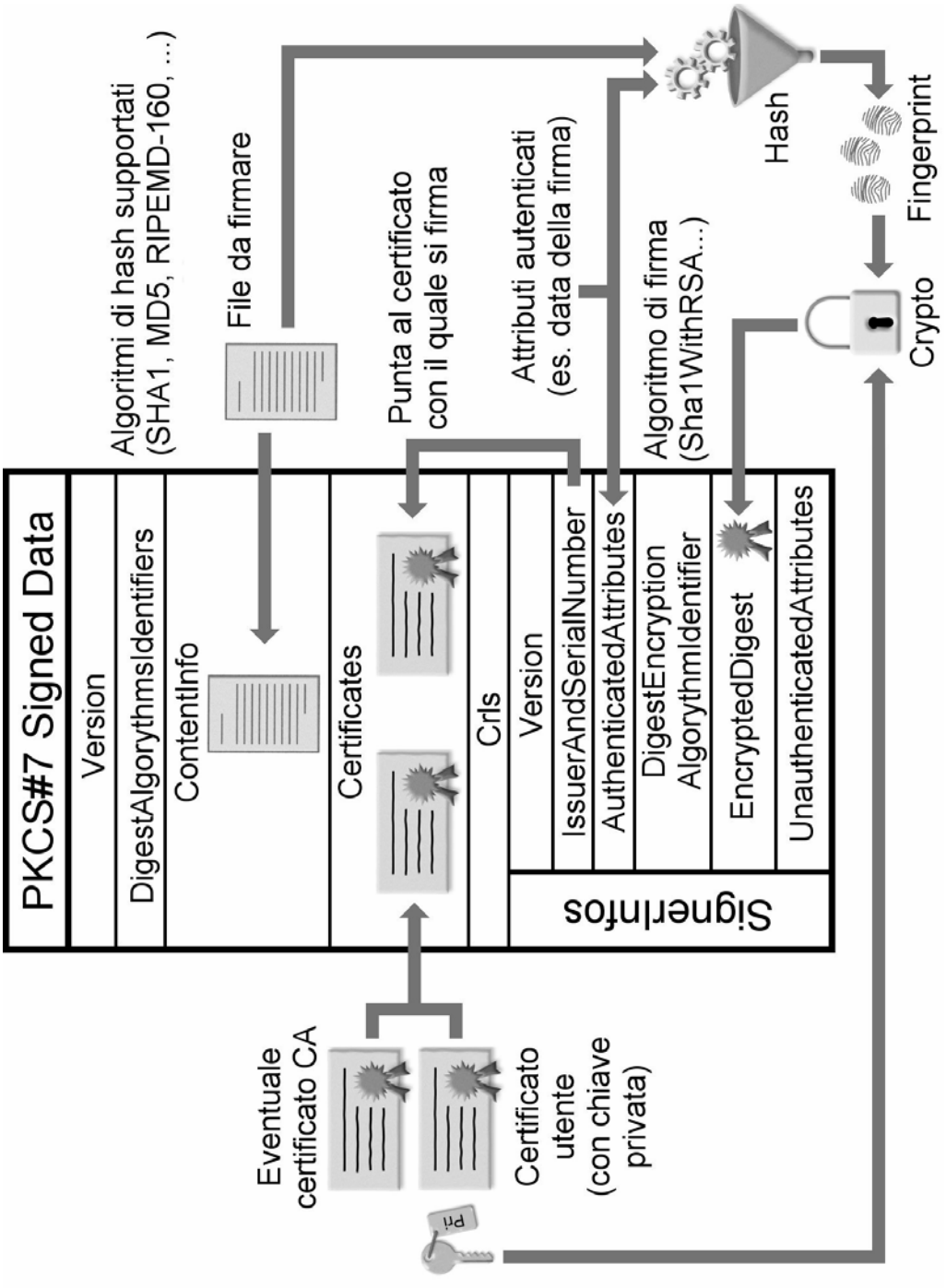

Figura 2.3: Creazione di un file PKCS#7

```
SignedData ::= SEQUENCE {
  version Version,
  digestAlgorithms DigestAlgorithmIdentifiers,
```

```
  contentInfo ContentInfo,
  certificates
    [0] IMPLICIT ExtendedCertificatesAndCertificates OPTIONAL,
  crls
    [1] IMPLICIT CertificateRevocationLists OPTIONAL, signerInfos
          SignerInfos }
```

Come si può vedere, la sequenza comincia con il numero di versione e l'identificativo degli algoritmi di firma supportati. La sezione ContentInfo è, invece, quella costituita dal blocco di dati che si vuole firmare; seguono quindi zero o più certificati e altrettante liste di revoca. Il tutto termina con uno o più blocchi SignerInfos che contengono ciascuno una firma[8] e la cui struttura è analizzata di seguito:

```
SignerInfo ::= SEQUENCE {
  version Version,
  issuerAndSerialNumber IssuerAndSerialNumber,
  digestAlgorithm DigestAlgorithmIdentifier,
  authenticatedAttributes
    [0] IMPLICIT Attributes OPTIONAL, digestEncryptionAlgorithm
    DigestEncryptionAlgorithmIdentifier, encryptedDigest
          EncryptedDigest,
  UnauthenticatedAttributes
    [1] IMPLICIT Attributes OPTIONAL }
```

Il blocco comincia con il numero di versione, l'Issuer e il numero di serie del certificato con cui è apposta la firma (potrebbero infatti esservene più d'uno o addirittura nessuno, nel qual caso risulta fondamentale individuare qual è quello con cui la firma è stata apposta). Viene quindi specificato quale algoritmo, tra quelli elencati nel campo digestAlgorithms visto sopra, è stato utilizzato per generare la

[8] Un documento può, infatti, essere firmato simultaneamente da più soggetti.

fingerprint; segue la lista degli attributi autenticati, inserita come sequenza di oggetti il cui tipo non è specificato.

A questo punto la firma di contentInfo + authenticatedAttributes[9] viene crittografata con l'algoritmo specificato in digestEncryptionAlgorithms e inserita come stringa di ottetti in encryptedDigest. Seguono, infine, gli attributi non autenticati, inseriti con la stessa sintassi di quelli autenticati.

Esistono forme particolari di strutture PKCS#7, che nello standard stesso vengono definite "degeneri"; se ne segnalano due:

- una prima forma è quella che prevede che i campi digestAlgorithms, contentInfo, signerInfo ed, eventualmente, crls vengano omessi, e che viene utilizzata per distribuire i certificati;
- una seconda forma è quella che è priva del solo signerInfo, che viene quindi utilizzata per autenticare soli attributi: come si vedrà in seguito, è questa una delle forme utilizzate nell'architettura di firma presentata in questa tesi.

Con l'introduzione di PKCS#7 si ha quindi a disposizione una struttura estremamente versatile, che consente di maneggiare simultaneamente i dati da firmare, degli attributi che li accompagnano e la firma del tutto; tuttavia, come si vedrà nel prossimo paragrafo, in alcuni casi PKCS#7 risulta ancora insufficiente per le caratteristiche necessarie per ottenere una firma digitale a norma di Legge.

2.4 Il pacchetto di file firmati: S/MIME

Ci si rende subito conto che PKCS#7 è comodo per firmare un singolo file di dati (che può essere inserito come stringa di ottetti in contentInfo), ma risulta insufficiente qualora sia necessario impacchettare più file[10], come del resto è

[9] Come visto prima a proposito di BER, l'encoding degli authenticatedAttributes non è univoco; al fine di evitare ambiguità lo standard stabilisce che ad essere firmata sia una copia in DER degli attributi e non gli attributi stessi.

[10] Si potrebbe pensare di ideare uno standard di impacchettamento per i dati ed inserire tutto nel contentInfo di PKCS#7; tuttavia, un siffatto standard non è stato ancora creato e, pertanto, ci si deve accontentare degli standard attualmente disponibili.

previsto dalla Legge italiana in alcuni casi. In questo caso, si fa riferimento ad un altro standard, S/MIME, che consente di impacchettare più file, accodandovi la relativa firma inserita sotto forma di PKCS#7 degenere.

S/MIME è l'acronimo di Secure Multipurpose Internet Mail Extensions ([RFC2633]) e nasce come estensione per aggiungere a MIME ([RFC204X]) la possibilità di crittografare e/o firmare file. Ciò viene ottenuto creando un nuovo tipo di pacchetto MIME (multipart/signed) e due tipi di dati (corrispondenti a application/x-pkcs7-signature per la firma e application/x-pkcs7-mime per i dati crittati ed eventualmente firmati[11]).

Escludendo il caso, non pertinente a questo lavoro, della crittazione, ci si limiterà a trattare quello di un pacchetto contenente uno o più file da firmare.

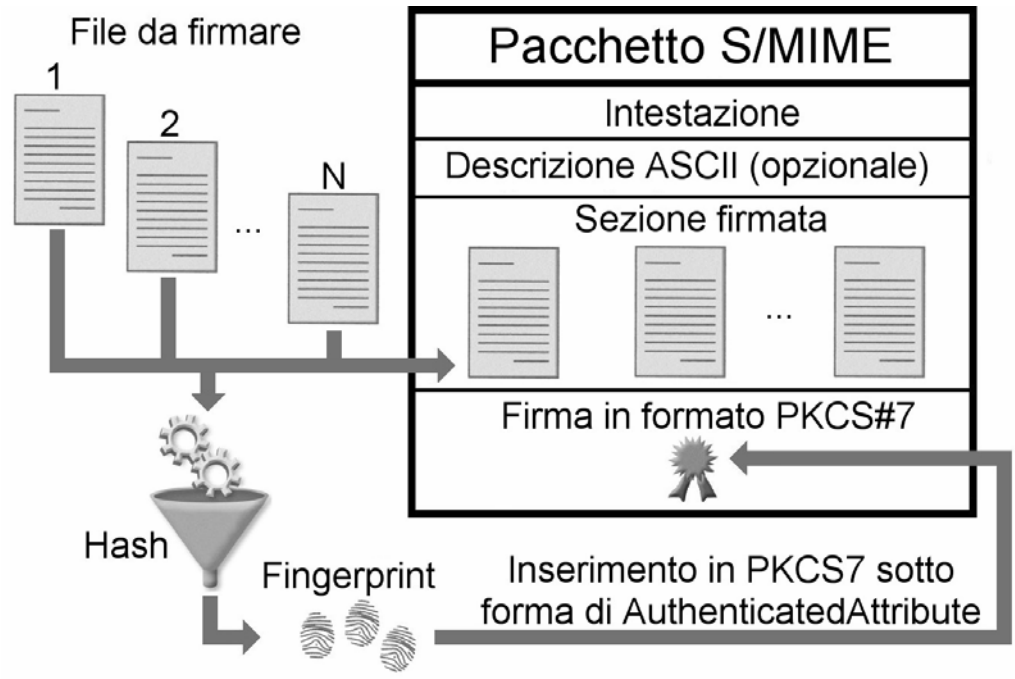

Figura 2.4:

Struttura di un file S/MIME

[11] Il tipo pkcs7-mime è utilizzato anche, quando abbinato ad un file con estensione .P7C, per la trasmissione di certificati (codificati PKCS#7 nella apposita forma degenere vista nel paragrafo precedente); questa tecnica, tuttavia, è impiegata quasi esclusivamente per la trasmissione via web dei certificati (in particolare durante la fase di creazione del certificato stesso).

Poiché la firma è salvata come sezione a parte (e, in particolare, come file la cui estensione è .P7S) il pacchetto dovrà essere sempre multipart/signed, anche se contenente un solo file; inoltre, fra i parametri dell'argomento Content-Type di MIME dovrà essere aggiunto "micalg=XXX", dove XXX è la sigla corrispondente all'algoritmo di hash utilizzato per la firma (per SHA-1 è "sha1", mentre, purtroppo, RIPEMD-160 non è ancora previsto).

Sebbene multipart, il pacchetto dovrà essere costituito da due soli elementi: la parte da firmare e la firma; non è assolutamente possibile creare un pacchetto costituito da tre o più parti. Ciò non deve trarre in inganno: poiché il MIME prevede la possibilità che una sezione contenga ulteriori MIME, per inserire più di un file è sufficiente fare in modo che la parte da firmare sia a sua volta un multipart, come visibile nella seguente tabella:

Encoding corretto di un pacchetto con un solo file	**Encoding corretto di un pacchetto con due file**	**Encoding errato di un pacchetto con due file**
Header Sezione 1: File Sezione 2: PKCS#7	Header Sezione 1: Sezione 1.1: File 1 Sezione 1.2: File 2 Sezione 2: PKCS#7	Header Sezione 1: File 1 Sezione 2: File 2 Sezione 3: PKCS#7

Tabella 2.4: esempi di encoding S/MIME di più file

La seconda sezione del file è costituita da un file PKCS#7 in una delle sue forme degeneri, e, precisamente, in quella che prevede authenticatedAttributes, ma non contentInfo.

La firma è inserita, ovviamente, fra gli authenticatedAttributes, ed è costituita dai seguenti attributi (tutti definiti in [PKCS#9]):

- SigningTime, contenente un tipo UTCTIME indicante l'ora e la data di firma;

- MessageDigest, costituito da una stringa di ottetti contenente l'hash della prima sezione dell'S/MIME.

In conclusione, va notato che, mentre PKCS#7 effettua il digest sul *contenuto* dei dati da firmare (infatti, come si è visto, gli authenticatedAttributes vengono prima codificati in DER e poi firmati, quindi l'utente è libero di scegliere la codifica che gli è più comoda o, comunque, di aggiungere o togliere parti del blocco non firmate – come ad esempio un certificato in più), S/MIME *firma se stesso*: uno spazio in più o in meno inserito non solo nel blocco dati, ma addirittura nella sua intestazione, può falsare irrimediabilmente la firma.

2.5 L'interfaccia verso la smartcard: PKCS#11

Terminata la lista di standard che si possono definire "software", resta in sospeso un ultimo problema: come colloquiare con la smartcard. Come già detto nel capitolo precedente, ISO 7816 definisce solo il formato fisico della carta e le basi del protocollo di colloquio con il lettore; nulla è specificato sulla struttura dei pacchetti che devono essere inviati o ricevuti dal dispositivo, per cui ogni produttore ha ampia scelta in merito e, quindi, si rende necessario un ulteriore strato che virtualizzi le operazioni gestibili dalla smartcard stessa.

Questo strato software, che riceve in ingresso richieste di operazioni da effettuare e si preoccupa di farle giungere alla smartcard, è stato formalizzato da RSA Labs nello standard [PKCS#11] (che prende anche il nome di *Cryptoki*), attualmente alla sua versione 2.10, datata Dicembre '99. Purtroppo, però, tale standard ha risentito pesantemente della guerra commerciale in corso in quegli anni fra i vari produttori di browser internet, col risultato che si sono venuti a creare due standard paralleli: Netscape adotta per il proprio browser PKCS#11, mentre Internet Explorer si serve di uno standard proprietario che si appoggia alle CryptoAPI di Windows. Il risultato finale è che una smartcard inizializzata con Netscape non funzionerà su

Explorer e viceversa[12]. In questo lavoro di tesi, alla luce di considerazioni che verranno discusse nel prossimo capitolo, lo standard scelto come base per la realizzazione dell'applicazione di firma digitale è PKCS#11, quindi verranno ignorati gli aspetti di compatibilità con Internet Explorer.

Prima di soffermarsi sulla smartcard è però il caso di spendere qualche parola sul dispositivo di interfaccia tra la carta e il computer; tale dispositivo, che prende il nome di *Smartcard Reader*, non è mai stato standardizzato in qualche documento ufficiale. Ne esistono, pertanto, svariate versioni (in particolare ve ne sono alcune che si collegano alla porta seriale del PC, altre a quella parallela, altre, infine, a quella PS/2); tutte però presentano lo stesso problema: poiché per scrivere i dati nella EEPROM della carta è necessaria una quantità di corrente non trascurabile (non presente sulle interfacce RS-232 o Centronics), il lettore riceve corrente dall'unica porta del computer che, per standard, fornisce una corrente ben definita e più che sufficiente per pilotare la smartcard: quella della tastiera. Fra le varie versioni di lettori, quella che ha assunto il ruolo di standard de facto è quella seriale, che prende il nome di *PC/SC*; sebbene il supporto PC/SC sia ormai integrato nei moderni sistemi operativi, introducendo un ulteriore livello di virtualizzazione (Applicativo -> PKCS#11 -> Driver PC/SC -> Smartcard), ogni produttore è, comunque, libero di fornire il lettore che più ritiene opportuno semplicemente integrando nello strato PKCS#11 il supporto per il lettore proprietario e non servendosi, quindi, del driver di sistema.

PKCS#11 (il cui nome completo è *Cryptografic Token Interface Standard*) è fornito come libreria da collegare ai propri applicativi. Laddove il sistema operativo ne fornisca il supporto, viene preferita la distribuzione sotto forma di libreria a linkaggio dinamico (sotto Windows i moduli PKCS#11 sono, infatti, delle DLL), il che consente all'utente di poter disporre di una libreria per ogni dispositivo

[12] La documentazione in dotazione alla smartcard utilizzata per realizzare questa tesi sostiene che un certificato installato con Netscape dovrebbe funzionare anche con Outlook (Express). In realtà, mentre sotto Windows 2000 Outlook non sembra affatto accorgersi della presenza della smartcard, sotto Windows 98 compare una richiesta di importazione di certificati dalla carta, ma alla fine non ne viene importato alcuno.

utilizzato; ove invece non disponibile, la libreria è fornita staticamente, ma questo implica la necessità di dover ricompilare il software se si cambia il dispositivo.

PKCS#11 astrae tutte le operazioni effettuabili con un generico dispositivo crittografico[13] servendosi di quattro concetti generici: la *sessione*, l'*oggetto*, la *funzione* e il *meccanismo;* ciascuna entità gestita dalla libreria è referenziata mediante un *handle*, esattamente come avviene per tutti gli oggetti sotto Windows o per i file sotto Unix.

La comunicazione con la smartcard comincia, innanzi tutto, con la ricerca di un lettore in cui sia inserita una carta; a questo punto viene inizializzata una *sessione*, fornendo il PIN di accesso (operazione di *login*). Esistono due PIN, uno per l'utente e uno per l'amministratore (*Security Officer*); il primo può effettuare tutte le operazioni, tranne il cambio del PIN e il reset della carta, che sono prerogativa esclusiva del secondo. All'atto dell'apertura della sessione va specificato se la richiesta è per una sessione utente o amministrativa; occorre prestare estrema attenzione nel login alla smartcard, in quanto una sequenza di accessi errati consecutivi (il numero varia a seconda del produttore della carta) può bloccare il dispositivo.

Terminato il login, è, quindi, possibile, a partire dall'identificatore di sessione ottenuto, effettuare tutta una serie di operazioni sulla carta; queste operazioni possono essere sostanzialmente di due tipi: accesso ai dati ed elaborazioni.

Appartengono alla prima categoria quelle operazioni che consentono di accedere a tutte le strutture dati memorizzate nella carta. Tali strutture, che prendono il nome di *oggetti*, sono distinguibili mediante un identificatore di classe che ricorda quello di BER, che è l'unico elemento comune a tutti gli oggetti presenti sulla carta. Gli oggetti si dividono sostanzialmente in tre tipi: vi sono oggetti virtuali, non fisicamente presenti nella EEPROM della carta, ma che forniscono informazioni sul modello e sullo stato del dispositivo (tali oggetti sono principalmente read-only); vi

[13] Esistono, infatti, alcune implementazioni PKCS#11 – tra cui quella fornita di default con Netscape – che utilizzano non una smartcard, bensì il calcolatore stesso per compiere le operazioni di firma. Tuttavia, la Legge Italiana impone che tali operazioni vadano effettuate da un dispositivo con particolari requisiti di sicurezza, per cui l'argomento non verrà approfondito ulteriormente.

sono, poi, oggetti "reali", che si dividono in oggetti definiti dallo standard (certificati, coppie di chiavi, ...) e oggetti definiti dall'utente. C'è, infine, una categoria di oggetti del tutto particolare che è quella cui afferiscono le chiavi (singola in caso di DES e in coppia in caso di RSA): queste sono generate dalla carta su richiesta dell'utente e, per rispettare i vincoli imposti dalla Legge, hanno una porzione di dati che non può essere letta dall'esterno (naturalmente la parte privata).

Sebbene, come già detto, l'unico campo comune a tutti gli oggetti è l'identificatore di classe, quasi tutti hanno un numero seriale o una label; non essendo PKCS#11 basato su direttori, è spesso necessario collegare tra loro oggetti diversi (come ad es. un certificato con la relativa coppia di chiavi): in questo caso lo standard impone che oggetti logicamente correlati presentino numero di serie e/o label uguali.

PKCS#11 fornisce un set di funzioni per accedere agli oggetti estremamente essenziale, ma non per questo poco efficace: è possibile ricercare oggetti in base al contenuto dei loro campi (in primis la classe) ed effettuare variazioni su singoli campi o sull'intero oggetto, oltre naturalmente alle operazioni di inserimento e cancellazione. Va inclusa in questa classe anche la funzione che consente di generare una coppia di chiavi.

Come si sarà capito, tutte le operazioni sugli oggetti vanno eseguite mediante delle *funzioni*, che sono a tutti gli effetti delle RPC (*Remote Procedure Call*) invocate dal PC, ma eseguite dalla carta: la libreria mette a disposizione tutta una serie di chiamate per effettuare le più disparate operazioni a partire dagli oggetti presenti nella carta stessa. Si noti che, sebbene lo standard risulti estremamente esaustivo sulle funzioni gestibili dalla smartcard, è consentito implementare solo parte delle funzionalità previste.

Le funzioni si dividono nei seguenti tipi principali:

- gestione della sessione (login, logout, modifica PIN, ...);
- accesso, modifica e ricerca di oggetti;

- generazione di chiavi;
- funzioni di hash semplice;
- crittografazione di blocchi dati;
- firma digitale (hash + crittografazione);
- generazione di numeri casuali.

I primi tre tipi sono già stati trattati, mentre i successivi tre, sostanzialmente, mettono a disposizione dell'utente tutte le operazioni crittografiche già elencate nel capitolo precedente. Interessante è poi la possibilità di ottenere un numero casuale tramite l'ultimo gruppo di funzioni: poiché, come noto, un calcolatore è in grado di generare numeri pseudocasuali (e non casuali!) e spesso l'algoritmo di generazione risulta in qualche modo prevedibile, è estremamente utile una sorgente di entropia esterna che, come tale, è difficilmente riproducibile.

Resta, infine, da chiarire il concetto del *meccanismo*. Tutte le funzioni della smartcard che riguardano crittografia e firma digitale sono disponibili in varie versioni a seconda degli algoritmi di hash e crittografia supportati dalla scheda; piuttosto che creare una funzione diversa per ogni algoritmo, nello standard si è preferito fornire alle funzioni crittografiche un parametro in più, che specifica il *meccanismo* (intendendo con questo termine una struttura che contiene sia il tipo di algoritmo che i suoi parametri specifici) con cui effettuare l'operazione.

2.6 Il futuro: PKCS#15

Sebbene la sola descrizione di PKCS#11 è sufficiente per chiarire il campo di analisi di questa tesi, si preferisce aggiungere una breve descrizione dell'ultimo standard nato in casa RSA, quello che (si spera) dovrebbe risolvere i problemi ancora irrisolti da PKCS#11.

Sostanzialmente, PKCS#11 è uno standard completo per quello che riguarda la funzionalità di un dispositivo crittografico; anzi, risulta così completo che non

esiste alcuna implementazione che copra tutte le funzioni e/o i meccanismi disponibili. Il vero punto debole dello standard sta proprio nel modo in cui esso è stato concepito, ossia come libreria che ciascun produttore di smartcard fornisce insieme alla smartcard stessa; quando, nell'aprile 1995, lo standard fu rilasciato, nessuno si preoccupò del fatto che PKCS#11 non fosse *Plug & Play*, ossia che l'utente intenzionato a spostare la propria smartcard da una postazione all'altra sarebbe stato costretto a portare con sé anche il dischetto con i driver. Tale soluzione risulta assolutamente impraticabile laddove, ad esempio, si intenda distribuire una carta d'identità elettronica in grado di effettuare firme digitali, come previsto in Italia nel prossimo quinquennio.

Serve quindi un ulteriore strato di virtualizzazione fra l'applicazione e la smartcard: è questo lo scopo dello standard PKCS#15, "Cryptographic Token Information Syntax Standard" (RSA Labs, v1.1, 6 Giugno 2000).

PKCS#15 si propone di creare un framework che, pur risultando perfettamente compatibile con gli standard precedenti, risulti:

- *platform neutral*, garantendo l'interoperabilità tra componenti che girano su varie piattaforme;
- *vendor neutral*, consentendo alle applicazioni l'utilizzo trasparente di dispositivi di diversi produttori;
- *application neutral*, rendendo le applicazioni "pronte" per eventuali innovazioni tecnologiche senza la necessità di modificare il codice [PKCS#15, Background].

Autoesplicativa risulta la figura con cui lo standard si apre, che si riporta qui tradotta:

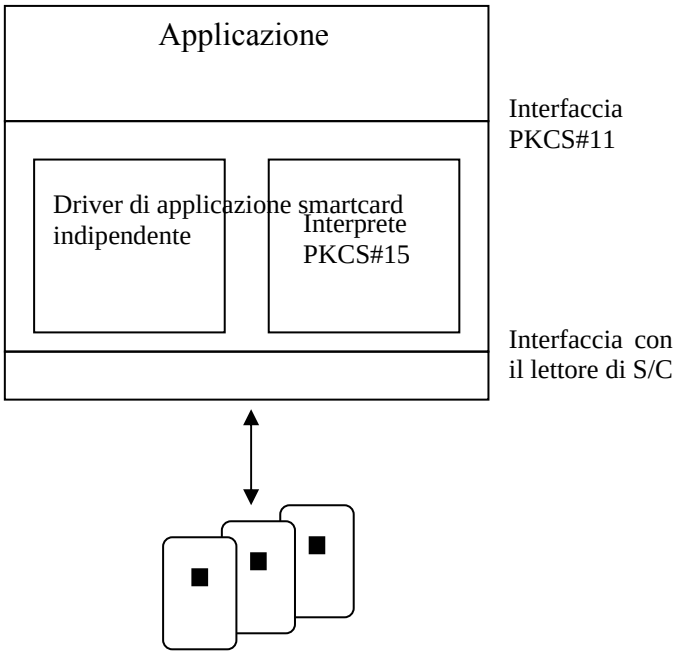

Figura 4.1: struttura di un framework PKCS#15

Senza scendere nei dettagli dello standard, si accenna che l'obiettivo della completa interoperabilità viene perseguito introducendo sulla smarcard un sistema operativo che consente di accedere agli oggetti secondo direttori, in maniera analoga a quanto accade in un file system; le stesse funzioni (o, più precisamente, applicazioni) vengono referenziate come se fossero dei file a cui è possibile inviare determinati tipi di comandi.

Per quanto riguarda il lavoro di questa tesi, risulta fondamentale l'interfaccia che PKCS#15 presenta verso l'alto: essendo questa perfettamente compatibile con PKCS#11, che è lo standard scelto per far comunicare con la smartcard l'architettura di firma qui presentata, risulta automatica la compatibilità del lavoro con lo standard prossimo a venire.

2.7 La legislazione italiana sulla Firma Digitale

Forse non tutti sanno che l'Italia è stato uno dei primi paesi a dotarsi di un'efficiente legislazione in merito alla Firma Digitale, campo in cui risulta attualmente all'avanguardia.

Tutta la giurisprudenza sulla Firma Digitale trae origine da un capoverso della [L_59/97] (che è sostanzialmente un'aggiunta alla Legge sulla riforma amministrativa e sulla semplificazione della P.A.); all'art. 15, infatti, vi si legge:

Gli atti, dati e documenti formati dalla pubblica amministrazione e dai privati con strumenti informatici o telematici, i contratti stipulati nelle medesime forme, nonché la loro archiviazione e trasmissione con strumenti informatici, sono validi e rilevanti a tutti gli effetti di legge.

Per la prima volta, la Legge riconosce la validità di un documento in forma elettronica; come poi prassi normale, in questi casi, il decreto stabilisce un ente preposto alla regolamentazione della materia (l'AIPA) e rinvia, quindi, ad un Regolamento ([DPR-10.11.97]) e a Regole Tecniche ([DCM-8.2.99]) per l'applicazione pratica di quanto citato poc'anzi.

E' nel Regolamento che si trova per la prima volta la definizione di Firma Digitale: in tale documento è contenuta, infatti, la definizione generica di tutti i concetti ad essa relativi. Trattandosi di un documento essenzialmente di definizione (e di correzione a Leggi precedentemente esistenti per renderle compatibili con la nuova tecnologia), l'interesse che riveste in questa sede è estremamente limitato.

Più specifico risulta, invece, il documento di Regole Tecniche emanato dall'AIPA, cui si farà riferimento più volte durante lo sviluppo dell'architettura di firma che è oggetto di questo lavoro. Le regole tecniche si dividono in quattro titoli:

I) regole tecniche di base;

II) regole tecniche per la certificazione delle chiavi;

III) regole per la validazione temporale e per la protezione dei documenti informatici;

IV) regole tecniche per le pubbliche amministrazioni;

V) disposizioni finali.

In base agli scopi preposti, il primo titolo è quello di maggiore (se non addirittura esclusivo) interesse; il secondo e il terzo sono relativi, rispettivamente, all'autorità di certificazione e ai sistemi di validazione temporale (questi ultimi non ancora in funzione); infine, gli ultimi due titoli riguardano disposizioni di carattere squisitamente amministrativo. Verranno, pertanto, qui riassunti i concetti di maggiore rilevanza contenuti nelle regole tecniche di base.

Il documento inizia con una serie di definizioni (tra le quali è interessante citare quella di *dispositivo di firma*: "*un apparato elettronico programmabile solo all'origine, facente parte del sistema di validazione, in grado almeno di conservare in modo protetto le chiavi private e generare al suo interno firme digitali*", per poi imporre gli algoritmi di crittografia (RSA o DSA) e hash (RIPEMD-160 o SHA-1).

Seguono, quindi, una serie di articoli sulle chiavi, i cui punti più salienti sono:

- le chiavi sono di tre tipi: sottoscrizione (firma), certificazione, marcatura temporale (qui ci si interesserà solo del primo tipo);
- una chiave può essere attribuita ad un solo titolare;
- ad ogni chiave è assegnato uno scopo specifico (crittazione, sottoscrizione, …) e non può essere utilizzata in altri casi; se la firma è apposta mediante procedura automatica occorre una chiave apposita;
- la lunghezza minima delle chiavi è di 1024 bit;
- la generazione deve essere coerente, garantire chiavi equiprobabili e assicurare l'identità del richiedente;
- se le chiavi sono generate al di fuori del dispositivo di firma (se ne veda la definizione), il canale di comunicazione della chiave privata deve essere assolutamente sicuro; inoltre, il sistema deve controllare la propria integrità prima della generazione della coppia di chiavi;

- è possibile utilizzare un dispositivo per contenere più chiavi, ma è invece vietato duplicare una chiave; il titolare della chiave è responsabile del dispositivo e deve custodirlo con la massima cura, lontano da un'eventuale trascrizione del PIN e richiedere immediatamente la revoca in caso di furto, difetto o compromissione.

Per quanto riguarda, invece, la firma, la Legge impone che:

- la firma deve essere conforme a (non meglio specificate) "specifiche pubbliche";
- il certificato deve sempre essere allegato alla firma;
- i software che consentono la firma devono essere del tipo *What You See Is What You Sign*, ossia l'utente deve vedere chiaramente cosa sta firmando (fanno eccezione le procedure automatiche);
- la firma deve essere generata all'interno del dispositivo;
- il dispositivo deve sempre identificare il titolare prima di apporre una firma.

Molte specifiche di quelle viste sopra discendono da evidenti considerazioni di sicurezza; qui si vuole solo far notare che:

- la legge è carente sulla specifica del formato di firma, il che potrà in seguito causare seri problemi di interoperabilità;
- è prevista la possibilità che il dispositivo non generi al suo interno la coppia di chiavi, anche se ciò è evidentemente sconsigliato; in ogni caso, se la generazione è esterna questa può avvenire solo da parte dell'ente certificatore e non dal titolare.

Il documento contiene, inoltre, ulteriori specifiche per le chiavi di certificazione e validazione temporale, ma essendo questa tesi incentrata sulla sottoscrizione non si approfondirà ulteriormente l'argomento.