Consiglio Nazionale delle Ricerche

La gestione dei token nelle API e nel client di firma digitale sviluppati nell'ambito della collaborazione IIT-SECETI

L. Bechelli, D. Fais

IIT B4-03/2002

Nota Interna

Ottobre 2002

Istituto di Informatica e Telematica

INDICE

1 Introduzione

Con l'emanazione della legge n° 59 nel 1997 e la promulgazione del DPR n° 513 dello stesso anno, sostituito successivamente dal testo unico del DPR n° 445 del 2000, ha inizio in Italia una vera e propria rivoluzione che a partire dai rapporti dei cittadini con la pubblica amministrazione, molto presto cambierà le abitudini delle aziende e presumibilmente entro un prossimo futuro delle singole persone in molti aspetti della propria vita quotidiana. Infatti, questa legislazione attribuisce valore legale al documento informatico che può sostituire completamente il documento cartaceo sia negli atti amministrativi sia in quelli che attestano le transazioni tra privati. Tuttavia, secondo il DPR n° 445 del 2000, affinché il documento informatico abbia valore legale è necessario che abbia determinati requisiti che garantiscano l'autenticità, l'integrità ed il non ripudio e ciò è possibile grazie alla firma digitale. La legislazione citata indica quali requisiti il documento informatico e la firma digitale su di esso apposta debbano avere, e più dettagliatamente le regole tecniche che l'AIPA (l'Autorità Italiana per la Pubblica Amministrazione) ha avuto cura di stilare con la partecipazione di esperti nel settore, mostrano le tecnologie e gli standard da utilizzare e le politiche di sicurezza da seguire.
Il DPR n° 445 del 2000 è stato superato dall'approvazione del D.L.vo 10 del gennaio del 2002, che recepisce la direttiva quadro europea in materia di firma digitale.
Signo è un'applicazione di firma digitale che è stata sviluppata nell'ambito di una accordo di collaborazione scientifica tra IIT e SECETI S.p.A.; essa è stata progettata quando era in vigore il DPR 513 e le sue modificazioni successive e quindi ottempera ai requisiti di quella legge per permettere all'utente, in tutta semplicità, di apporre la "propria" firma su un documento informatico. Tuttavia, *Signo* risulta ancora conforme ai requisiti della legge attuale, che impone vincoli meno severi per l'apposizione di una firma digitale su un documento informatico.
Per questa ragione, nella descrizione tecnica dello strumento si farà riferimento al DPR 513 o al DPR 445 già citati, considerando che, soddisfacendo vincoli imposti da queste leggi, automaticamente lo strumento di firma risulta adeguato alla vigente normativa.

Per il DPR 513 del 97, la firma digitale è equivalente alla firma autografa, e ciò comporta che l'apposizione della firma digitale su un documento informatico deve essere effettuata in modo accorto, dato che l'utente non potrà in seguito disconoscerla. Pertanto per l'utente è obbligo seguire delle indicazioni di sicurezza tra le quali

1. conservare con estrema cura, mantenendola segreta, la chiave privata con la quale l'utente è in grado di apporre una firma digitale ad un qualsiasi documento;
2. effettuare l'operazione di firma in ambiente sicuro, senza possibilità di intrusione

Per la conservazione della chiave privata e la creazione della firma digitale i dispositivi di memorizzazione di massa dei moderni PC non sono ritenuti sufficientemente sicuri, poiché facilmente accessibili da parte di applicazioni su cui spesso l'utente non ha il completo controllo oppure perché facilmente corruttibili. A questo scopo sono invece utili dispositivi di cui l'utente possa facilmente disporre, dalle limitate e ben determinate funzionalità e che permettano di memorizzare un certo numero di dati. In questo documento ci riferiremo a questi dispositivi con il nome di *token*. In generale un token per la firma digitale dovrebbe permettere di memorizzare la coppia di chiavi, pubblica e privata, ed il certificato che garantisce l'associazione tra chiave pubblica e utente. La chiave privata dovrebbe non essere leggibile in alcun modo e a tale scopo il token dovrebbe prevedere dei meccanismi di sicurezza che lo impediscano. Infine il dispositivo dovrebbe permettere di effettuare quelle operazioni crittografiche necessarie per creare la firma o per criptare dati senza che la chiave privata sia esportata all'esterno del dispositivo. Esempi di dispositivi con questi requisiti sono le Smartcard e i Token USB.

Figura 1. Il Token USB Aladdin

Per accedere alle funzionalità di memorizzazione e alle funzionalità crittografiche di un generico token la RSA Security ha rilasciato uno standard, il PKCS#11 giunto alla versione 2.11, che specifica un'API, detta Cryptoki, che permette di astrarre dal dispositivo effettivamente utilizzato.
Sulla base di questo standard e grazie ad altre tecnologie, di cui faremo menzione, è possibile costruire un'architettura che permetta di utilizzare alternativamente o congiuntamente dispositivi di produttori diversi e semplificare le procedure per utilizzare tali dispositivi per apporre la firma digitale sui documenti.

Scopo di questo documento è descrivere l'architettura, le tecnologie e le soluzioni implementative adottate nella creazione delle API Java sulle quali l'applicazione Signo si basa. In particolare saranno approfondite le parti delle API che hanno per scopo la gestione dei token crittografici.

2 Stratificazione delle API

Le API dell'applicazione Signo sono strutturate su tre livelli logici, di seguito descritti in ordine crescente di astrazione:

- **api_interface** e **token**: il primo package è un'interfaccia di copertura per l'accesso alle librerie prodotte da terzi, il secondo implementa l'accesso ai token crittografici sfruttando il modello ad oggetti decritto dallo standard PKCS#11.
- **toolkit**: questo package costituisce uno strato di *ingegnerizzazione* delle librerie dello strato sottostante. Questo livello offre al livello superiore, classi e metodi che semplificano notevolmente le complesse procedure di *preparazione* per l'accesso alle funzionalità di base delle librerie e dei token crittografici.
- **application**: è il package che costituisce il livello più elevato delle API, altamente specializzato, che semplifica ulteriormente la gestione degli elementi di una PKI. Questo package è pressoché l'unico visibile al programmatore dell'applicazione finale. È composto da:
  - API a disposizione dei clienti per l'integrazione della firma digitale nelle applicazioni locali o web based
  - GUI, ovvero componenti grafici minimali, come viewers e finestre di dialogo, finalizzate all'integrazione nelle applicazioni desktop

Il package **application** comprende anche il ***client di firma*** Signo.

3 Descrizione delle tecnologie utilizzate

3.1 Terminologia

In seguito in questo documento, per rendere più precisa la descrizione e facilitare la lettura saranno adottati i seguenti termini

- *Crypto Device* è in generale il software o l'hardware dedicato all'esecuzione di algoritmi crittografici simmetrici e asimmetrici. Le implementazioni software sono realizzate utilizzando le specifiche Java Cryptography Architecture, che permettono di ottenere un'elevata interoperabilità ed estendibilità grazie alla tecnica del Security Provider. Tra i sistemi hardware gli strumenti utilizzati sono le smartcard ed token USB conformi allo standard PKCS#11.
- *Security Module*: sono sistemi hardware o software adibiti alla conservazione sicura di chiavi crittografiche. In generale corrispondono agli stessi dispositivi crittografici anche se nell'architettura che di seguito descriveremo ciò non è propriamente esatto. In particolare in questo documento saranno distinti:
  - *Hardware Security Module* (HSM): i sistemi hardware di memorizzazione, come ad esempio le Smartcard e i token USB;
  - *Protected Storage* (PS): il sistema software di memorizzazione delle chiavi e dei certificati che come vedremo fa riferimento allo standard PKCS#12, che definisce la struttura di un particolare file che possa contenere in forma criptata chiavi e certificati.
- *Credenziale*: per credenziale in genere intendiamo la chiave pubblica e il certificato che ne garantisce la corrispondenza con un determinato soggetto

3.2 Security Provider

In Java® la tecnica comunemente utilizzata per accedere ad un'implementazione hardware o software di un algoritmo crittografico è quella del ***Security Provider***.
Un Security Provider, nell'implementazione della Virtual Machine Java, è un meccanismo che implementa un certo numero di funzioni di sicurezza. È possibile installare, in una VM Java, un numero arbitrario di Security Provider: qualora un'applicazione in esecuzione richieda all'ambiente di run-time una certa funzione di sicurezza, come ad esempio l'utilizzo di un particolare algoritmo di crittografia, la VM restituisce un'istanza della classe del primo Security Provider in elenco che offre l'implementazione della funzione.
L'*installazione* del Security Provider consiste nella registrazione, nell'ambiente di esecuzione, del nome delle classi del provider in relazione alle operazioni che mettono a disposizione, come ad esempio:

```
Provider.add("AlgoritmoRSA", "it.seceti.pki.implRSA.class");
```

Per poter accedere in modo trasparente alle funzioni fornite dal provider, è necessario che questo rispetti un'interfaccia, stabilita dalla SUN, chiamata JCE, acronimo di Java Cryptography Extension.
JCE implementa anche alcuni algoritmi crittografici simmetrici, asimmetrici, funzioni hash, ad altri standard relativi alla sicurezza dei sistemi ed alla firma digitale, come PKCS#12, X.509 ed altri.
L'API, qui descritta, gestisce la registrazione di due Security Provider nell'ambiente di esecuzione. I provider forniscono l'implementazione di alcune funzioni crittografiche che le librerie IAIK possono richiedere ed istanziare mediante il metodo getInstance(). In questo modo l'API può sfruttare tutte le potenzialità delle librerie IAIK.

3.2.1 Il Provider IAIK

Le librerie IAIK forniscono un provider cui corrisponde un'implementazione a livello software di molte funzioni crittografiche, tra cui gli algoritmi asimmetrici RSA e DSA, le funzioni hash SHA, e RIPEMD-160, gli standard X.509v3, PKCS#1, 5, 7, 9, 10, 12, ed una serie di elementi per costruire strutture dati di tipo ASN.1. La nostra API utilizza innanzi tutto il Provider IAIK che costituisce il nucleo del *Crypto Device Software*. Per utilizzare il Device Software occorre registrare l'implementazione del Provider IAIK per primo tra i Security Provider della Virtual Machine Java.

3.2.2 Il provider SECETI

Le API di Signo forniscono un Security Provider specifico che mette a disposizione l'implementazione a livello hardware dell'algoritmo RSA. Questo provider permette di utilizzare l'unità di elaborazione di dispositivi hardware quali Smartcard o token USB. In linea con le disposizioni di legge, l'implementazione hardware di RSA di tale provider è utilizzato per realizzare la firma ad un più alto livello di sicurezza. Per quanto riguarda le funzioni hash, la verifica delle firme, le funzioni per la creazione e la manipolazione delle strutture dati, definite in PKCS#7 e X.509, le API utilizzano sempre le implementazioni fornite dal Provider IAIK.
Quando da applicazione, si vuole effettuare una firma, le API gestiscono autonomamente la registrazione dei due Provider nell'ambiente a runtime. Più in dettaglio per le operazioni di firma RSA PKCS#1 mediante smartcard, l'API registra il Provider SECETI per primo e successivamente il Provider IAIK.

Ad esempio, ipotizziamo che l'utente dell'applicazione finale voglia verificare la firma di un documento e quindi firmarlo a sua volta. Il funzionamento delle API, invisibile all'utente, seguirà pressappoco questo schema di funzionamento:

1) Applicazione client: Avvio con smartcard inserita nell'apposito lettore;
2) API: registrazione del provider SECETI;
3) API: registrazione del provider IAIK;
4) Applicazione client: richiesta di verifica di una firma;
5) API: rimozione dei provider precedentemente registrati;
6) API: registrazione del provider IAIK;
7) IAIK: richiesta di un'implementazione della funzione di hash SHA-1;
8) Java VM : interrogazione del primo provider registrato a runtime che implementa la funzione SHA-1: Provider IAIK;
9) Java VM: creazione e rilascio di un'istanza della classe che implementa la funzione SHA-1;
10) IAIK: richiesta di un'implementazione dell'algoritmo di verifica RSA;
11) Java VM: interrogazione del primo provider registrato a runtime che implementa l'algoritmo di verifica RSA: Provider IAIK;
12) Java VM : creazione e rilascio di un'istanza della classe che implementa l'algoritmo di verifica RSA;
13) IAIK: verifica della firma del documento in formato PKCS #7;
14) Applicazione client: richiesta di firma di un documento;
15) API: rimozione dei provider precedentemente registrati;
16) API: registrazione del provider SECETI;
17) API: registrazione del provider IAIK;
18) IAIK: richiesta di un'implementazione della funzione di hash SHA-1;
19) Java VM : interrogazione del primo provider registrato a runtime che implementa la funzione SHA-1: Provider IAIK;
20) Java VM : creazione e rilascio di un'istanza della classe che implementa la funzione SHA-1;
21) IAIK: richiesta di un'implementazione dell'algoritmo di firma RSA;
22) Java VM : interrogazione del primo provider registrato a runtime che implementa l'algoritmo di firma RSA: Provider SECETI;
23) Java VM : creazione e rilascio di un'istanza della classe che implementa l'algoritmo di firma RSA;
24) IAIK: firma del documento in formato PKCS #7;

3.2.3 Security Module

Un Security Module fornisce l'accesso a dati protetti in relazione al dispositivo di memorizzazione utilizzato. L'API implementa due tipi di Security Module:

- Hardware Security Module (*HSM*): i dati sono memorizzati su un dispositivo hardware dedicato. L'API, quali HSM, supporta i token hardware conformi allo standard PKCS#11;
- Protected Storage (*PS*): i dati sono memorizzati in file secondo lo standard PKCS#12, la cui implementazione è fornita dalle classi IAIK.

3.2.3.1 Hardware Security Module

L'implementazione degli Hardware Security Module utilizza l'interfaccia definita dallo standard PKCS#11. Infatti PKCS #11 definisce un'API in linguaggio ANSI C che permette di invocare i comandi dei dispositivi crittografici. Per poter utilizzare un token conforme al PKCS#11 sono necessarie:

- una libreria dinamica (DLL) fornita dal produttore del device: questa libreria converte le chiamate PKCS#11 in operazioni sul processore crittografico;
- un lettore di smartcard, accessibile mediante protocollo PC/SC, fornito assieme al sistema operativo o dal produttore del device assieme ai driver;
- una libreria Java che, mediante Java Native Interface (JNI), trasformi le chiamate Java in invocazioni alla libreria dinamica o DLL PKCS#11

La libreria che svolge il compito descritto in quest'ultimo punto è la JavaToPKCS11 di IAIK, costituita da tre livelli di stratificazione: il primo, quello più in basso è costituito da un modulo scritto in ANSI C che ridefinisce le funzioni specificate dallo standard PKCS#11 secondo la sintassi e le regole indicate dalla tecnologia JNI. Il secondo livello è costituito da un package Java che mappa esattamente le funzioni C in corrispondenti metodi Java. Infine il terzo livello raccoglie i metodi esportati dal livello inferiore ricostruendo in Java il modello ad oggetti descritto dallo standard PKCS#11.

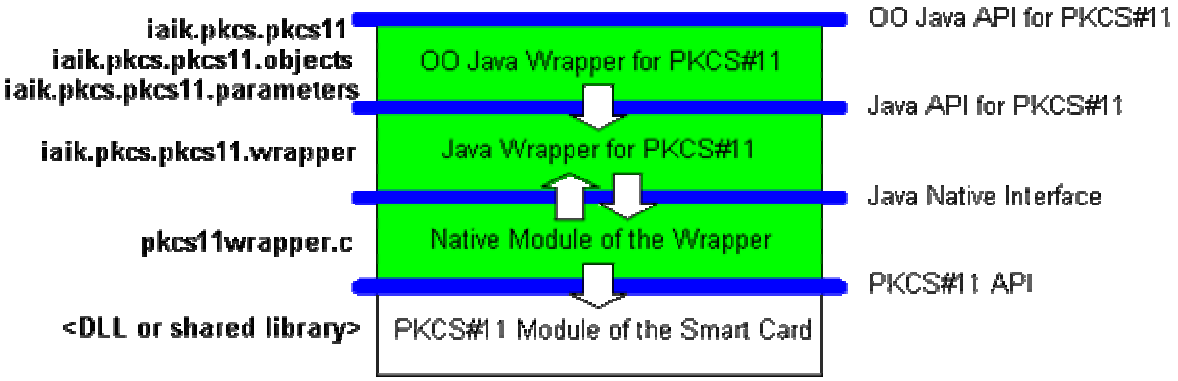

Figura 2. Stratificazione delle librerie IAIK JavaToPKCS11

3.2.3.2 Il PKCS#11

Questo standard contiene le specifiche delle API che forniscono l'accesso ai token hardware. In particolare lo standard descrive un modello ad oggetti utile per gestire più dispositivi dello stesso tipo e per trattare il loro contenuto. Lo standard prevede l'inizializzazione della libreria e quindi l'accesso al contenuto degli slot, cioè gli alloggiamenti dove è possibile inserire i token. Se uno slot

contiene un token è possibile ottenere un reference ad esso e quindi aprire una sessione.
Esistono diversi tipi di sessioni che possono essere aperte: innanzi tutto si distinguono le sessioni dell'operatore da quelle dell'utente. Le sessioni dell'operatore permettono d'inizializzare il token e di sbloccare la password d'accesso qualora il token abbia subito un certo numero di tentativi d'autenticazione da parte dell'utente, nessuno dei quali andati a buon fine. Una sessione aperta dall'utente invece permette di accedere agli oggetti contenuti nel token e di realizzare operazioni crittografiche. Lo standard definisce una complessa gerarchia di oggetti, ognuno dei quali caratterizzato da un preciso insieme di attributi ai quali andranno associati altrettanti valori. Tra gli oggetti definiti particolare rilievo per le API di Signo assumono le chiavi pubbliche, le chiavi private e i certificati conformi allo standard X509 e gli oggetti contenenti dati generici. Tra gli attributi vi è uno particolarmente importante che stabilisce se l'oggetto deve essere *privato* oppure no. Agli oggetti privati è possibile accedere solo previa autenticazione da parte dell'utente. L'autenticazione avviene mediante l'inserimento di un PIN che solo il proprietario del token hardware dovrebbe conoscere. Ad esempio la chiave privata facente parte di una coppia di chiavi certificate, è un oggetto privato e risulta visibile all'applicazione solo in seguito all'autenticazione dell'utente. Allo stesso modo per effettuare la firma digitale di un documento mediante token hardware occorre accedere alla chiave privata e ciò è possibile soltanto mediante autenticazione da parte dell'utente.

3 Implementazione

Le API gestiscono autonomamente HSM, PS e Crypto Device come entità separate, mettendole in relazione nel modo opportuno secondo le operazioni richieste al livello dell'applicazione.
La filosofia alla base dell'architettura delle API prevede che si possa scegliere a livello utente un *token crittografico* da utilizzare sia per implementare le funzioni crittografiche, sia per memorizzare i dati protetti. Qualora sia selezionato un HSM, cioè una token hardware, le operazioni crittografiche che richiedono un livello di sicurezza maggiore, sono effettuate utilizzando il provider SECETI, che sfrutta l'unità di elaborazione del token hardware (smartcard o token USB). Viceversa, qualora sia utilizzato un PS software, le funzioni crittografiche sono implementate mediante le librerie software del Security Provider IAIK.
Questa gestione è nascosta a livello *application* delle API e quindi tanto più al livello della vera e propria applicazione. Infatti PS e HSM possiedono una comune interfaccia che permettono al livello superiore di astrarre dal tipo di Security Module, *software* o *hardware*, effettivamente utilizzato.

Di seguito analizziamo in dettaglio la struttura delle API, partendo dal livello più basso, fino ad arrivare al layer applicativo.

3.3 Livello API_Interface

3.3.1 I Provider

Al livello API_Interface è presente il package

it.seceti.pki.api_interface.security.provider

che contiene le implementazioni dei due provider, IAIK e SECETI precedentemente descritti.
Il provider SECETI, in particolare, oltre ai metodi standard previsti nel rispetto dell'interfaccia di definizione java, ha in più una funzione di installazione di un device crittografico: questo metodo consente di registrare, a runtime, una istanza della classe che gestisce l'accesso al token crittografico correntemente in uso.

3.3.2 I Token PKCS#11

I token hardware conformi allo standard PKCS #11 v2.11 hanno come si è detto libreria nativa con una interfaccia comune e questa caratteristica è sfruttata dalle API per la gestione dei token. Attraverso la teconologia JNI, è possibile caricare a runtime la libreria nativa fornita dal produttore insieme al token, cosicché sia possibile invocare le funzioni che eseguono le operazioni crittografiche sul token, e le funzioni che permettono di accedere ai dati in esso contenuti.
A basso livello le API di Signo utilizzano il package it.seceti.pki.token.pkcs11 che contiene le classi per gestire i token ed accedere al loro contenuto. In particolare le classi presenti sono:

- Pkcs11Module: istanziando un oggetto di questa classe avviene il caricamento dinamico della libreria nativa associata al token. Quindi attraverso di esso è possibile accedere a tutti i token compatibili riconosciuti, presenti nei vari alloggiamenti e ad informazioni che permettano di identificare ciascuno dei token riconosciuti o che descrivano gli slot in cui sono inseriti.
- Pkcs11Token: un oggetto di questa classe è associato in modo biunivoco ad uno specifico token. I token possono essere identificabili mediante il numero di serie assegnatogli dal produttore e quest'informazione è alla base del riconoscimento tra l'oggetto istanziato e il token hardware.
- Pkcs11Object: una istanza di questa classe corrisponde ad un oggetto contenuto nel token. Essa fornisce un insieme di metodi per riconoscere il tipo di oggetto tra quelli descritti nello standard PKCS#11 ed estrarne il contenuto.

I metodi che i livelli superiori delle API invocano per accedere alle funzionalità offerte dai token hardware sono specificate nell'interfaccia it.seceti.pki.token.token: in tal modo gli strati superiori delle API possano utilizzare in modo trasparente qualsiasi tipo di token hardware astraendo dalle peculiarità di ciascuno.
Attualmente le API garantiscono l'accesso esclusivamente ai token hardware di tipo PKCS#11 mediante la classe tuttavia con quest'architettura è possibile estendere facilmente le API creando nuove classi conformi a tale interfaccia.

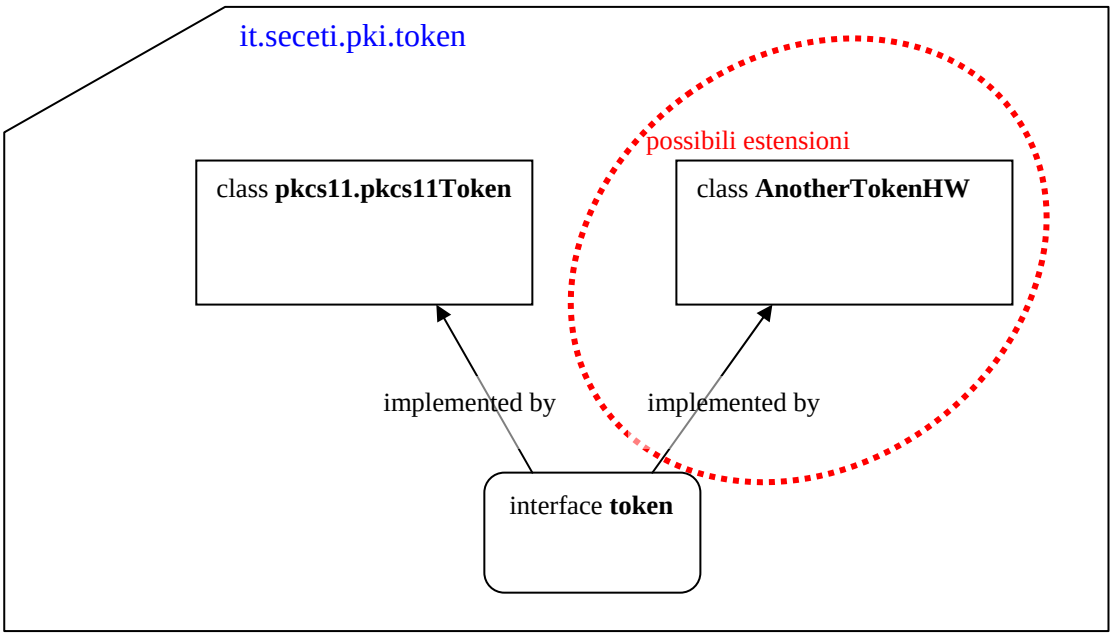

Figura 3. Relazione di ereditarietà tra le classi contenute nel package Token e possibili estensioni.

3.3.3 Protected Storage PKCS#12

PKCS#12 definisce il formato dei file contenenti le chiavi private e i certificati utente in modo protetto, ovvero cifrato mediante algoritmi di crittografia simmetrici come Triple-DES.
L'implementazione del formato PKCS#12 è fornita dalle librerie IAIK, ed è ripresa dalla classe it.seceti.pki.api_interface.jce.pkcs12.pkcs12PS, che semplifica la maggior parte delle operazioni di accesso al Protected Storage.

3.4 Livello Toolkit

L'architettura del livello toolkit costituisce il fulcro sulla quale si basa l'estendibilità della libreria e la gestione dinamica dei token. A questo livello si opera una separazione logica tra i Security Module ed i Crypto Device, cui corrispondono specifiche classi. La gestione dinamica dei token operata dal livello *toolkit* consiste nell'associare un Crypto Device con il corretto Security Module quando scelto un token ed una credenziale a livello applicativo si decide di effettuare una determinata operazione. Al livello applicativo risulta completamente trasparente l'associazione tra Crypto Device e Security Module. Quindi attraverso il token crittografico è possibile al livello superiore accedere al Security Module o al Crypto Device senza occuparsi del tipo di Security Module o di Crypto Device effettivamente istanziato in corrispondenza di quel token. Ciò è reso possibile da un parte grazie all'interfaccia comune implementata dai Security Module che permettono di non occuparsi se il dispositivo di memorizzazione è hardware o software, dall'altra parte attraverso una serie di classi e di metodi che permettono la gestione dinamica dei Security Provider Java, scegliendo di volta in volta quello che deve essere utilizzato secondo le operazioni da effettuare. Grazie a tale gestione dinamica, come vedremo in seguito, l'unico onere riservato al livello dell'applicazione è scegliere il token crittografico da utilizzare.

3.4.1 I Security Module

Ai Security Module è dedicato il package it.seceti.pki.toolkit.securityModules. Che siano di tipo hardware oppure di tipo software, essi devono rispettare l'interfaccia SecurityModule, la quale specifica un insieme di metodi per gestire le credenziali, le chiavi non certificate e i certificati presenti sul dispositivo: ad esempio è possibile ottenere l'elenco delle credenziali, delle chiavi non certificate e le informazioni ad essi correlate, oppure crearne di nuove.
Le implementazioni disponibili sono:

- Pkcs11, è la classe che corrisponde ad un token hardware (smartcard o token USB) e i cui metodi permettono di accedere ad uno specifico token. Rispetto al package token una istanza di questa classe implementa una cache per le credenziali, le chiavi pubbliche non certificate e i certificati contenuti nel token. La presenza della cache permette di caricare il contenuto del dispositivo hardware solo una volta per sessione, ed effettuare così più velocemente le successive operazioni. Viene creato un oggetto Pkcs11 per ogni specifico token rilevato. L'impostazione del corretto Security Module quindi può avvenire esclusivamente indicando a livello superiore la tipologia e il numero di serie del token da prendere in considerazione tra quelli presenti negli alloggiamenti.
- Pkcs12, è la classe che corrisponde al token Software. Questo Security Module permette di accedere e gestire le credenziali contenute in file del formato descritto dallo standard PKCS#12, utilizzando il Protected Storage descritto precedentemente. A differenza della classe Pkcs11 essa non implementa una cache perché l'accesso a file è di per se sufficientemente veloce e non crea problemi prestazionali. Ad ogni sessione le API permettono di istanziare un solo token di tipo Software e quindi non avremo più di una istanza della classe Pkcs12.

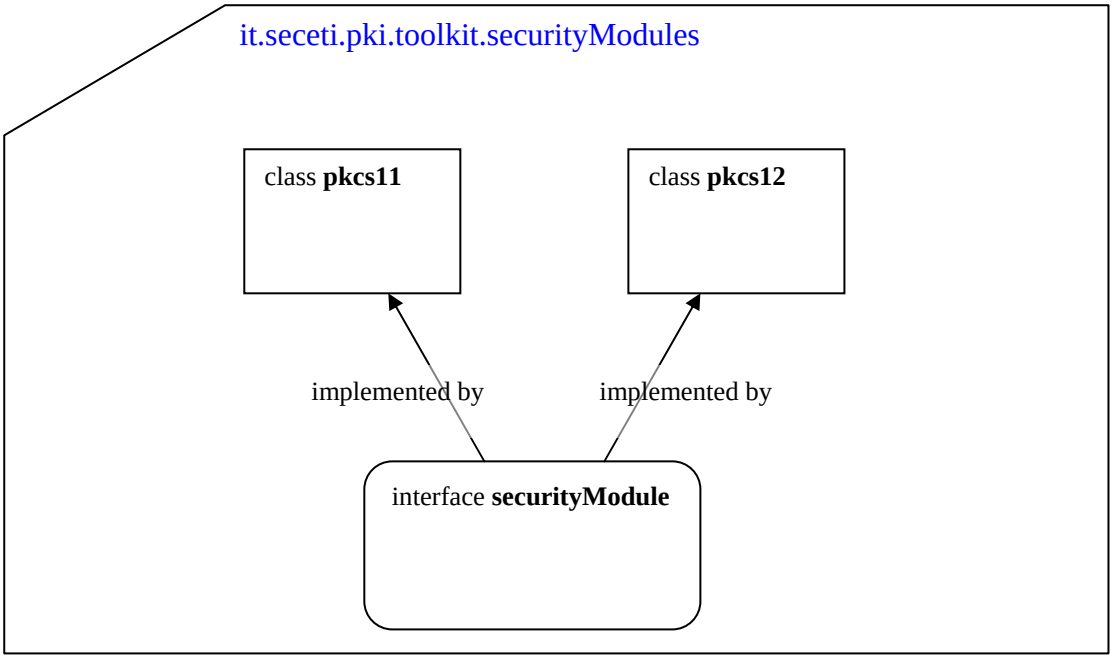

Figura 4 Relazione di ereditarietà tra le classi contenute nel package SecurityModules

3.4.2 Crypto Devices

Come già visto precedentemente nella trattazione della tecnologia dei provider, i Crypto Device devono essere gestiti in modo che

- l'algoritmo RSA, implementato mediante l'unità di elaborazione della smartcard, sia utilizzato solo per la firma, ma non per la verifica dei documenti;
- al Crypto Device sia associato il Security Module di tipo PKCS #11 se il token utilizzato è una smartcard, e quest'ultima sia adoperata per realizzare la firma dei documenti;
- al Crypto Device sia associato il Security Module di tipo PKCS #12 se il token utilizzato è di tipo software.

A tale scopo la soluzione adottata dalle API, prevede una gerarchia di classi, contenute nel package it.seceti.pki.toolkit.dynamicProvider il cui scopo è caricare dinamicamente in memoria la corretta sequenza di provider, in base alle operazioni da svolgere.
Alla base di questa gerarchia c'è l'interfaccia dynamicProvider che fornisce dei metodi standard per effettuare le seguenti operazioni

- caricamento della lista di provider di default
- caricamento della lista di provider per effettuare le operazioni di firma
- caricamento della lista di provider per effettuare le operazioni di verifica

Indichiamo con il termine Dynamic Provider le classi che implementano tale interfaccia.

Attualmente l'API è corredata da due Dynamic Provider:

- defaultProvider è la classe che permette di caricare il provider IAIK che fornirà quindi l'implementazione di tutte le operazioni crittografiche. Come vedremo successivamente questo Dynamic Provider è associato al token di tipo Software.
- pkcs11Provider è la classe responsabile del caricamento della giusta sequenza di provider quando il Crypto Device è di tipo PKCS #11. In particolare registra a runtime
  - il provider IAIK come provider di default
  - il provider SECETI e successivamente il provider IAIK per le operazioni di firma
  - il provider IAIK per le operazioni di verifica.

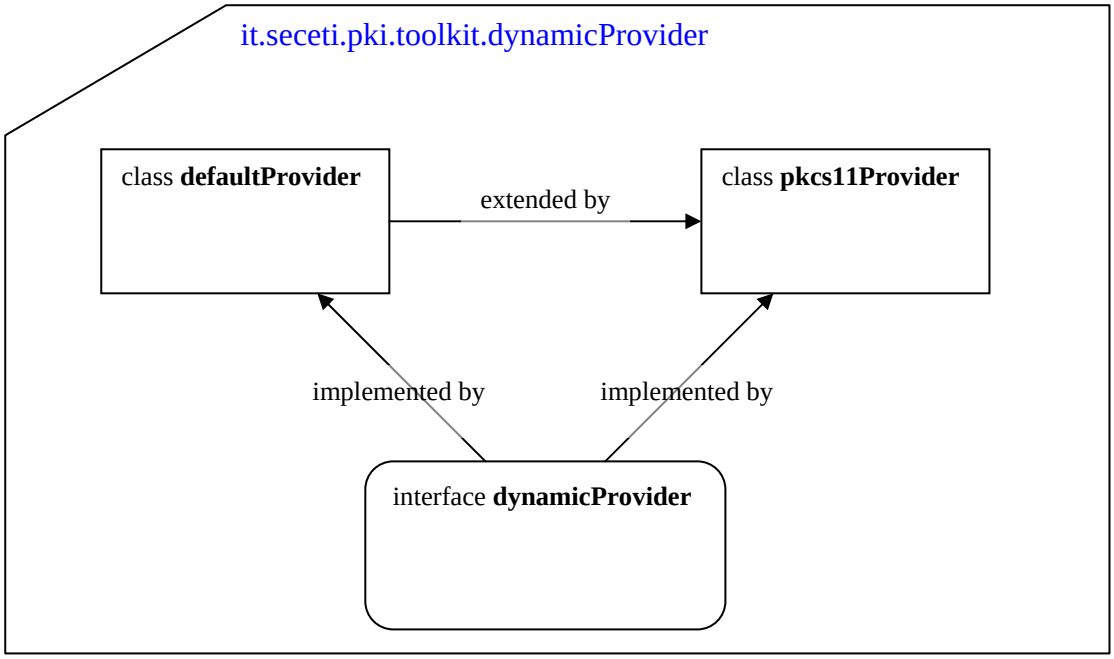

Figura 5 Relazione d'ereditarietà tra le classi del package dynamicProvider

3.4.3 Embedded Token

L'utilizzo di Security Modules e di Crypto Devices nella corretta associazione è garantito da un sistema di classi appartenenti al package it.seceti.pki.toolkit.embedded, che rispondono all'interfaccia embedded. Le classi che implementano tale interfaccia permettono di accedere ad un Security Module e a Crypto Device senza doversi occupare del tipo specifico di tali entità logiche. Il numero di *embedded* implementati corrispondono a quello dei tipi di Security Module precedentemente descritti.

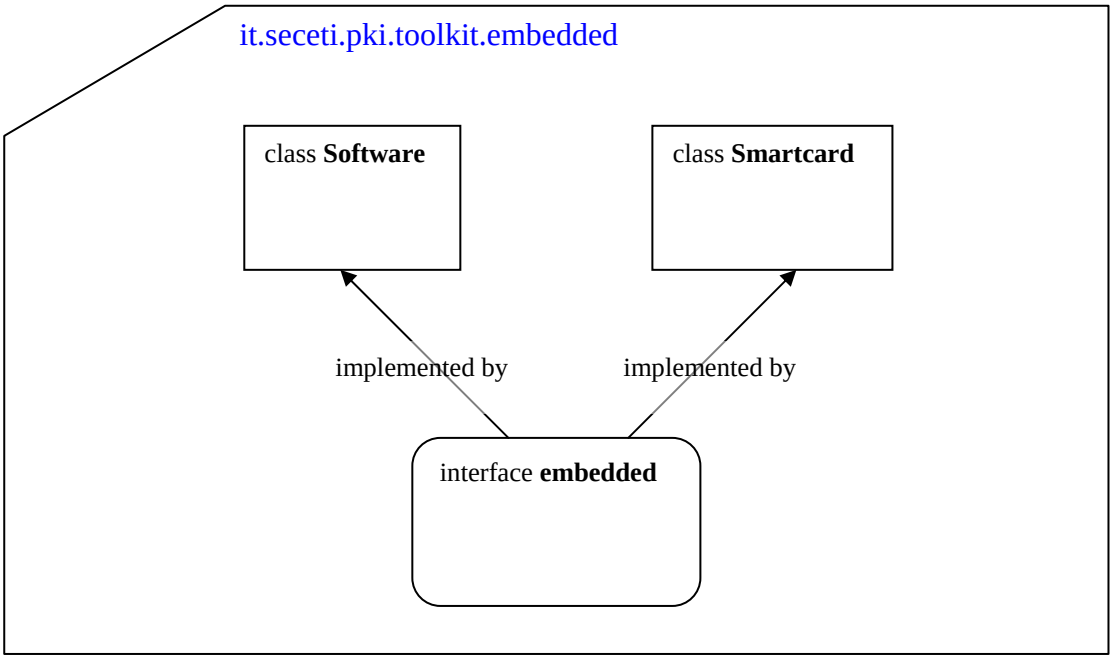

Figura 6. Relazione d'ereditarietà tra le classi del package embedded.

Gli *embedded* forniti dalle API corrispondono sostanzialmente alle tipologie di token crittografici che possono essere scelti a livello dell'applicazione. Essi sono:

- **Embedded Software**: associa il Security Module pkcs12, con un Crypto Device dello stesso tipo. In pratica il Crypto Device pkcs12 si sostanzia nella creazione di un oggetto di tipo deafaultProvider cioè del Dynamic Provider che provvede a registrare a runtime il provider IAIK sia per le operazioni di firma, sia per le operazioni di verifica.
- **Embedded Smartcard**: associa il Security Module pkcs11, con un Crypto Device dello stesso tipo. In pratica viene istanziato un oggetto di tipo pkcs11Provider che provvede a registrare a runtime i provider in modo tale sia utilizzata l'implementazione hardware dell'algoritmo di firma RSA, mentre per ogni altra funzione crittografica sia utilizzata l'implementazione software fornita dal provider IAIK.

Possiamo riassumere quanto detto con la seguente tabella:

Embedded	tipologia di *Security Module* e *Crypto Device*	Provider per la firma	Provider per la verifica ed altre funzioni crittografiche
Smartcard	pkcs11	SECETI (implementazione hardware)	IAIK
Software	pkcs12	IAIK	IAIK

Ciascuna istanza di un **Embedded Smartcard,** in più si occupa del caricamento della libreria nativa di una determinata tipologia di token e di rilevare i token appartenenti a tale tipologia negli slot. A tale scopo fornisce un insieme di metodi che permettono di controllare se un token hardware è ancora inserito nel suo alloggiamento oppure se è stato rimosso. Per ogni token hardware rilevato viene istanziato un oggetto pkcs11Token un Security Module (un oggetto pkcs11) che carica la cache delle credenziali e delle chiavi pubbliche non certificate. L'impostazione del token hardware da utilizzare per effettuare le operazioni crittografiche avviene scegliendo uno dei token inseriti tramite il relativo numero di serie o lo slot in cui risulta inserito. Inoltre un oggetto Smartcard mappa le credenziali con i token che le contengono: ciò è possibile perché il certificato che compone la credenziale è identificato univocamente dal nome della CA emittente del certificato e da un numero di serie. In questo modo è possibile impostare il corretto token hardware per effettuare le operazioni crittografiche, scegliendo la credenziale da una lista che comprenda tutte quelle contenute nei token presenti in un dato istante.

3.4.4 Le classi cryptoToken ed InputToken

La classe it.seceti.pki.toolkit.token.cryptoToken permette di istanziare ed accedere ad un token crittografico identificato mediante un oggetto InputToken, la cui classe appartiene allo stesso package. Il token crittografico ottenuto creando un oggetto cryptoToken è un *token virtuale* che corrisponde in sostanza ad un determinato tipo di token, hardware o software, mediante il quale è possibile accedere ai dati protetti oppure utilizzarne le funzionalità crittografiche. Esso assolve il compito di nascondere al livello superiore la gestione degli *embedded* e quindi il coordinamento tra Security Module e Security Provider.

La filosofia alla base della classe cryptoToken è uniformare il procedimento con cui accedere alle funzionalità del token, qualunque sia il tipo associato. Ad esempio, mentre l'uso delle smartcard o dei token USB impone che l'utente apra una sessione con la carta e per attivare operazioni sui dati protetti è necessaria una procedura di login, l'utilizzo di PS software non richiede l'apertura di una sessione di comunicazione ma una password per l'accesso ai dati, anche se essi non sono critici come per il caso dei certificati. La soluzione adottata dalla classe cryptoToken è raccogliere tutti i

vincoli ineludibili e definire un insieme di procedure che siano applicabili ad entrambi, token software o hardware. Quindi proseguendo con l'esempio, anche se l'utilizzo di un PS software non richiede l'apertura di una sessione essa comunque viene simulata, mentre qualunque sia il Security Module utilizzato, l'accesso ai dati, pubblici o privati, richiede comunque un PIN.

La classe InputToken costituisce un elemento di comunicazione verticale tra l'applicazione finale, il livello *application* ed il livello *toolkit* delle API. Il suo scopo principale è fornire alle API informazioni sul tipo di token da utilizzare, cosicché a livello *toolkit* la classe cryptoToken possa scegliere l'*embedded* da istanziare, ed eventualmente l'*embedded* possa caricare la libreria nativa associata al token hardware scelto a livello applicativo. L'InputToken può essere utilizzato anche per indicare alle API di Signo comportamenti specifici di alcune tipologie di token, in modo da poter fornire un migliore supporto ed astrarre effettivamente dal dispositivo utilizzato.
Dunque per istanziare un oggetto di tipo cryptoToken è necessario un InputToken, che contiene un identificatore univoco per il token scelto ed un insieme di proprietà peculiari per ciascuna tipologia di token. L'attuale implementazione della classe InputToken prevede l'utilizzo di un nome simbolico per ciascuna smartcard, che può essere assegnato arbitrariamente a livello applicativo oppure agendo sui file di configurazione gestiti dal livello *application* delle API. Viceversa, per l'utilizzo del token di tipo software, le API riconoscono il solo InputToken istanziato mediante il nome simbolico "Software".

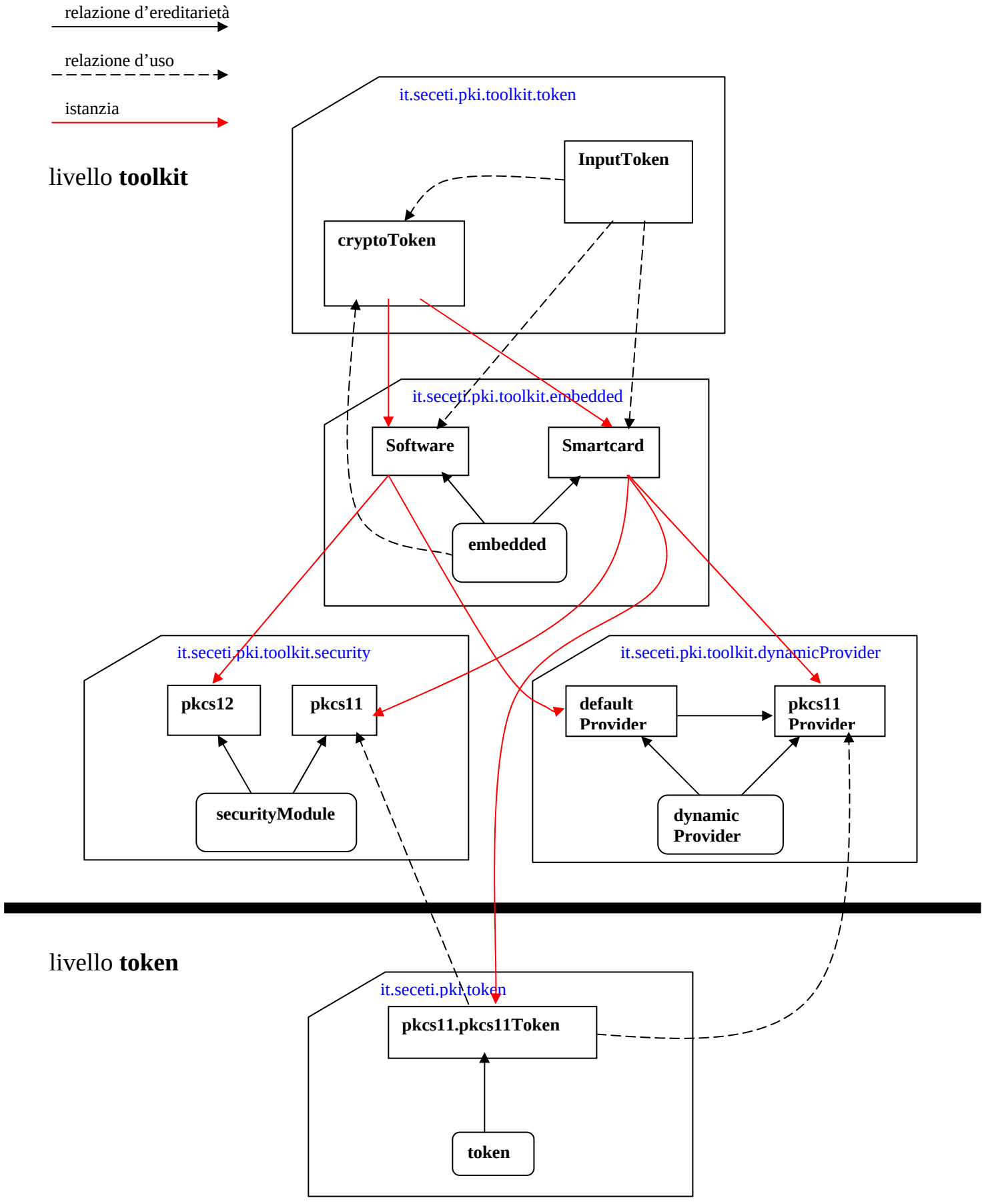

Figura 7. Panoramica dei package e delle classi adibite alla gestione dei token nelle API di Signo, e le varie relazioni che intercorrono tra essi.

Mediante un oggetto InputToken è possibile associare ad un particolare token un insieme qualunque di proprietà. Tuttavia attualmente sono previste alcune particolari proprietà come ad esempio la proprietà “library”, il cui valore corrisponde al nome della libreria nativa per i token hardware.

3.4.5 Uso di un token virtuale

L’inizializzazione di un token virtuale avviene istanziando un oggetto cryptoToken. A tale evento corrisponde la creazione di un oggetto *embedded* corrispondente al tipo di token scelto al livello dell’applicazione. L’embedded come si è visto a sua volta rileva i token effettivamente presenti della tipologia indicata, alloca i Security Provider opportuni ed istanzia i Security Module.
A questo punto per effettuare operazioni di firma o per accedere a dati protetti occorre selezionare una tra le credenziali presenti nei token rilevati. Nel caso di smartcard o di token USB, l’elenco delle credenziali è ricavato da una indagine all’interno di esso per ricercare le coppie certificato-chiave pubblica, mentre nel caso di PS *software*, l’elenco è ottenuto da un file contenente il nome simbolico ed il path completo per ciascun file di tipo PKCS#12, precedentemente registrato.
Dopo aver selezionato la credenziale, l’utente deve effettuare il login sul token che contiene la credenziale: per tale operazione viene richiesto un PIN all’utente, il quale può corrispondere, secondo il token scelto, al PIN del token hardware oppure alla chiave segreta necessaria per decifrare il file di tipo PKCS #12.
Per effettuare le operazioni di firma o di verifica occorre richiamare delle procedure preliminari, rispettivamente initSign e initVerify, che si occupano di registrare i Security Provider nell’ambiente a runtime della JVM nel giusto ordine ed effettuano un login sul token richiedendo all’utente il PIN di accesso alla chiave privata. Al termine dell’operazione di firma i Security Provider sono deallocati e viene terminata automaticamente la sessione sul token mediante logout. Questo metodo rispetta le disposizioni di legge giacché fa sì che il PIN venga richiesto per ogni firma digitale da eseguire.
Il livello *application* delle API semplifica ulteriormente le procedure di firma e di verifica, mediante i metodi *sign* e *verify* della classe it.seceti.pki.document.bustaCrittografica, il primo dei quali richiede il solo inserimento del PIN.

3.5 Livello application

Questo livello definisce classi che permettono di effettuare agevolmente la firma e la verifica di un documento, ed un ulteriore insieme di classi che esportano metodi statici atti a semplificare la gestione dei vari elementi della PKI. Per quanto riguarda la gestione dei token, la classe di riferimento è it.seceti.pki.application.token.tokenManager.

3.5.1 Token Manager

Il Token Manager permette di gestire più token virtuali contemporaneamente, in modo che si possa mantenere più sessioni aperte contemporaneamente su token di diverso tipo, sia che ad essi corrispondano. Quindi le API di Signo permettono di gestire un numero indeterminato di alloggiamenti e di token hardware sia dello stesso tipo, attraverso un singolo token virtuale (una istanza di cryptoToken), sia di tipo diverso, istanziando più oggetti cryptoToken. Avendo più istanze, è possibile passare dall’una all’altra senza dover ogni volta passare attraverso la complessa procedura di inizializzazione del token e ciò rende l’applicazione più efficiente.
Dopo aver scelto la tipologia di token da utilizzare, e ciò si ottiene semplicemente istanziando il Token Manager, è possibile utilizzare un sottoinsieme dei metodi forniti dal cryptoToken per accedere ai dati protetti contenuti nei token di quella tipologia effettivamente rilevati.

In secondo luogo il Token Manager coordina *due database* garantendo la consistenza delle informazioni contenute. L'accesso ai database da parte dell'applicazione finale è fornito da un insieme di metodi statici che il Token Manager esporta.
I database sono implementati mediante le classi fornite dal package it.seceti.pki.toolkit.database. Queste classi consentono di gestire dei semplici database in due diverse modalità a seconda che l'applicazione finale sia di tipo stand-alone oppure sia una applet:

- la prima modalità consente la memorizzazione del contenuto dell'archivio su file in un formato standard leggibile del tipo "chiave = valore". Ciò consente di modificare il contenuto del database anche manualmente, semplicemente editando i file.
- la seconda modalità gestisce il database esclusivamente in memoria RAM.

Il Token Manager carica i database in memoria automaticamente, quando da applicazione viene invocato un metodo ad essi relativo, che ne richiede o ne modifica il contenuto.

Il *primo database* utilizza il file *tokens.inf*, e consente di

- memorizzare informazioni e proprietà generali sui token virtuali gestiti dalle API,
- mantenere l'archivio dei token correntemente installati nella libreria.

La proprietà fondamentale tokens contiene la lista dei nomi simbolici dei token attualmente installati; il Token Manager per ognuno di questi token carica dal database le proprietà ad esso relative con le quali prepara automaticamente gli InputToken necessari ad istanziare i token virtuali. Da applicazione è così possibile scegliere il token virtuale da istanziare indicandone solamente il nome simbolico. Attualmente la libreria richiede che per ciascun token virtuale installato, token *Software* escluso, sia definita la proprietà library. È anche possibile associare a ciascun token altre proprietà, che possono essere gestite dall'applicazione finale. Ad esempio il client di firma che accompagna le API, gestisce la proprietà generale selected e la proprietà status associata ad ogni token installato. Il valore della proprietà selected corrisponde al nome simbolico del token virtuale predefinito, mentre la proprietà status indica se il token è abilitato ad effettuare le operazioni di firma.
Se non è presente il file *tokens.inf* il Token Manager crea un database contenente il solo token *Software*. Tuttavia la classe esporta anche dei metodi per aggiungere nuovi token al database dei token installati. Inoltre è possibile istanziare un token virtuale non presente nel database dei token installati, che verrà aggiunto automaticamente con tutte le proprietà ad esso associate.

Il *secondo database* contiene l'elenco dei token hardware supportati dalle API, con le rispettive librerie native. Per accedere a questo database il Token Manager fornisce un insieme di metodi che prevedono

- il caricamento del database a partire dal file *tokenDatabase.inf*;
- l'inserimento e la rimozione di token e delle relative proprietà;
- la memorizzazione del database e la sua deallocazione.

In questo database la proprietà fondamentale è tokens che, in questo contesto, contiene la lista dei token hardware "conosciuti".
Mentre il database dei token installati viene mantenuto in memoria per tutta la sessione dell'applicazione finale, per questo secondo database è previsto che il caricamento in memoria sia temporaneo per non occupare eccessive risorse, in quanto l'archivio dei token conosciuti potrà prevedibilmente essere anche molto grande.

Per utilizzare da applicazione i metodi che consentono di inserire o rimuovere elementi dai due database occorre utilizzare oggetti di tipo InputToken. Come già visto precedentemente, mediante InputToken è possibile definire un insieme arbitrario di proprietà da associare all'identificatore simbolico di un token virtuale. Inserire un oggetto InputToken in uno dei database equivale ad inserire il nome simbolico nella lista tokens e ad aggiungere le proprietà relative nell'archivio.

3.5.2 La cache dei token

La gestione di una molteplicità di token inseriti in più slot fa nascere l'esigenza di una ottimizzazione nella procedura di recupero dei dati in essi contenuti. A tale scopo alla gerarchia di classi descritta nei precedenti paragrafi si affianca una ulteriore gerarchia di classi che raccoglie tutte le informazioni relative ai token. Rispetto alle classi precedentemente descritte il cui scopo principale è effettuare operazioni sui token, tale gerarchia permette l'accesso a informazioni quali:

1. le informazioni di stato relative ai token
2. il contenuto dei token: in particolare l'elenco delle credenziali e dei certificati

A partire dal pkcs11Token per finire al TokenManager ciascuna classe introduce informazioni riguardanti la gestione ed il contenuto dei token. Da ognuna di esse è possibile ottenere una istanza di una classe con il relativo contenuto informativo. Queste informazioni sono quindi raccolte nella gerarchia di classi che costituisce la cache.
La seguente tabella mostra le corrispondenze tra le classi per la gestione dei token hardware o software e le classi che contengono i dati che vengono raccolti durante la gestione dei token.

Classe esistente	**Competenza**	**Classe della cache**	**Competenza**
TokenManager	gestione centralizzata delle tipologie di token (installate e supportate) e mediante impostazione del numero di serie accesso alle operazioni crittografiche fornite dal token	TokenCache	accesso alla cache relativa di ciascuna tipologia di token e mediante impostazione del numero di serie accesso alla informazioni relative ad un token specifico
Smartcard / Software	creazione del Security Module, impostazione del Dynamic Provider (*solo per Smartcard*: inizializzazione della libreria nativa di una particolare tipologia di token, istanziazione di un pkcs11Token per ogni token rilevato, selezione del token da utilizzare per le operazioni crittografiche)	TokenTypeResource	accesso alle informazioni relative alla tipologia di token ed accesso mediante impostazione del numero di serie alle informazioni relative ad uno specifico token
pkcs11 / pkcs12	accesso ad uno specifico token per l'effettuazione di operazioni di memorizzazione, (mantiene l'elenco delle credenziali e dei certificati)	TokenResource	accesso all'elenco delle credenziali e dei certificati e alle informazioni di stato di uno specifico token (contiene in realtà un reference allo stesso pkcs11)
pkcs11Token / pkcs12PS	accesso alle funzionalità offerte dalla libreria nativa per effettuare operazioni su uno specifico token	TokenInfo / SoftInfo	mantiene le informazioni di stato relative ad uno specifico token

E' evidente che mentre mediante TokenManager e la gerarchia di classi sottostante è possibile modificare lo stato del token, ciò non è possibile mediante la gerarchia delle classi che formano la cache. In questo modo qualora uno specifico token sia disinserito dal suo apposito slot sarà possibile chiudere la sessione con tale token e deallocare le relative strutture dati mantenendo tuttavia tutte le informazioni che lo riguardavano accessibili mediante le classi che costituiscono la cache.
A livello applicativo l'intera cache è rappresentata dall'istanza di un oggetto TokenCache. Per quanto detto attraverso di esso è possibile accedere alle informazioni riguardanti uno specifico token che sia stato istanziato almeno una volta durante una sessione dell'applicazione che utilizza le API di Signo, anche se esso non risulta più inserito in uno degli slot.
Un oggetto TokenCache conserva soltanto i reference degli oggetti contenenti le informazioni relative ai token e ciò implica la cache sarà automaticamente aggiornata ogni qual volta si verifichi un evento che modifichi lo stato od il contenuto di un token.
Ottenendo quindi dal TokenManager un istanza dell'oggetto TokenCache, a partire da esso è sempre possibile accedere a qualsiasi informazione senza bisogno di riottenere un nuovo oggetto

in quanto il suo contenuto è sempre allineato con l'effettivo contenuto del token e con le relative informazioni di stato. Ad esempio, l'aggiunta o la cancellazione di una credenziale non richiede chiamate ad altri metodi per ottenere nuovamente una cache aggiornata.

Nella figura seguente viene mostrata più in dettaglio la gerarchia di classi che costituisce la cache, con l'indicazione dei metodi principali.

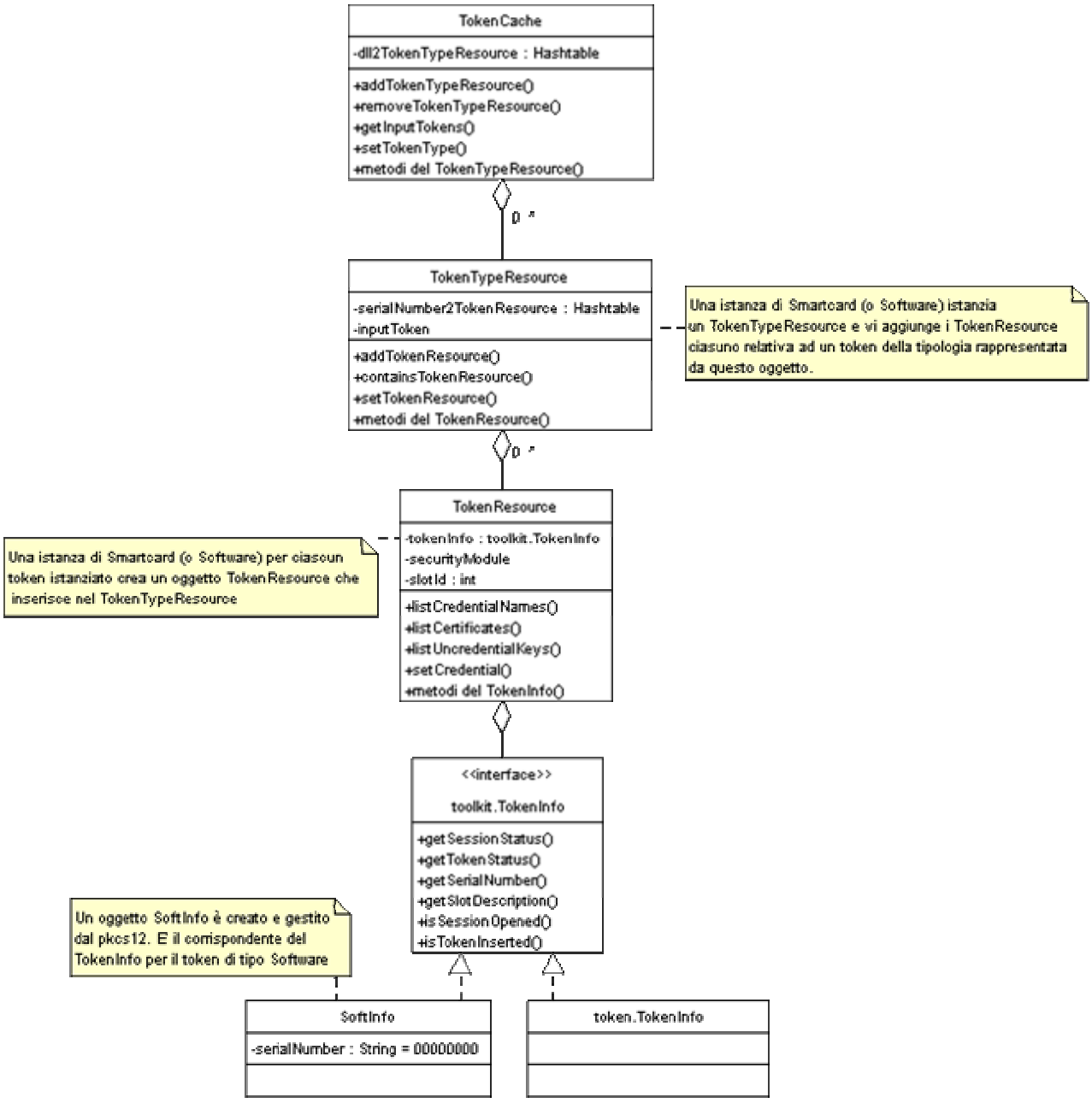

Figura 8. Diagramma UML delle classi che compongono la cache delle credenziali e delle chiavi non certificate.

3.6 L'applicazione Signo

3.6.1 Rilevazione automatica dei token

L'applicazione Signo è costituita da una serie di procedure guidate che consentono di effettuare la firma dei documenti, la verifica di documenti firmati, creare una coppia di chiavi asimmetriche e richiederne la certificazione, effettuare particolari operazioni sui token utilizzati. Inoltre vi è la possibilità di agire su varie opzioni di configurazione.

Alcune procedure richiedono la rilevazione automatica dei token alloggiati negli appositi slot, siano essi token USB o smartcard inserite in lettori PC/SC, a partire dall'elenco delle tipologie di token installati.
Il token *Software* risulta sempre presente. A tale scopo il Token Manager fornisce un metodo che esegue la seguente procedura:

1. caricamento dell'elenco delle tipologie di token installate
2. per ciascuna tipologia di token
    a. inizializzazione della libreria nativa
    b. per ciascuno slot rilevato dal sistema
        i. controllo della presenza di un token inserito
        ii. istanziazione del token e acquisizione del numero di serie

In un secondo tempo è possibile caricare la cache delle credenziali e delle chiavi non certificate.

Tra le procedure che richiedono la rilevazione automatica dei token vi è il pannello di configurazione che permette di visualizzare un albero con la lista dei tipi di token installati. Se è in uno slot è presente un token di una determinata tipologia, nell'albero compare il numero dello slot in cui il token inserito da cui dipartono i rami ciascuno con l'indicazione di una delle credenziali presenti nel token.

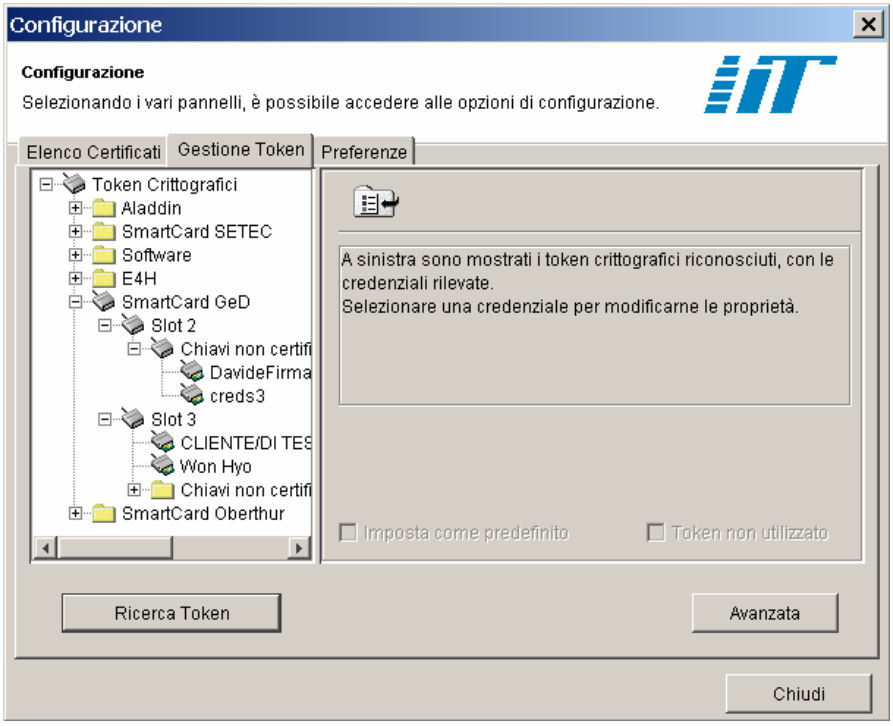

Figura 9. Vista dell'albero dei tipi di token installati, con due Smartcard GeD inserite in corrispondenza degli slot 2 e 3.

Anche la procedura di firma permette di scegliere tra la vista delle tipologie di token installati e la vista dei token effettivamente presenti nei vari alloggiamenti. Scegliendo uno dei token installati vengono rilevati i token di tale tipologia inseriti negli slot e per ciascuno di essi viene caricata la lista delle credenziali. Al passo successivo le credenziali rilevate nei vari token sono visualizzate in un'unica tabella ed è possibile sceglierne una qualsiasi per firmare il documento selezionato.

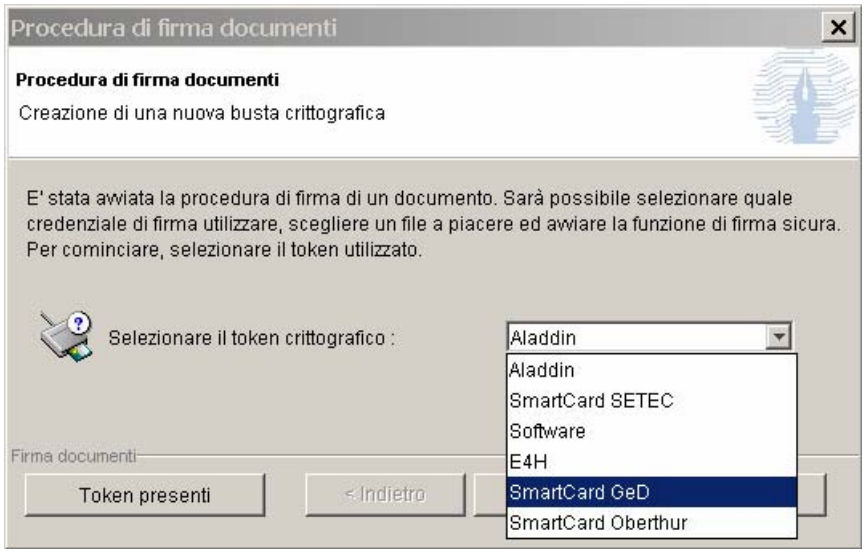

Figura 10. Primo passo della procedura di firma dei documenti con la vista delle tipologie di token installate.

La vista dei token presenti invece richiama la procedura automatica di rilevazione dei token e mostra le tipologie di token per le quali esiste un token hardware inserito in uno degli slot. In questo caso accanto al nome della tipologia di token compare il numero di slot in cui il token risulta inserito.

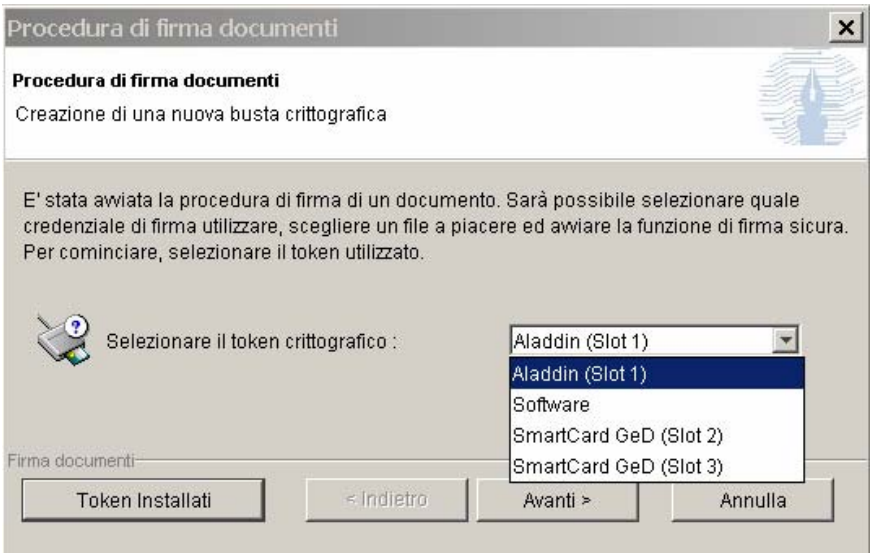

Figura 11. Primo passo della procedura di firma dei documenti con la vista dei token effettivamente presenti.

Allo stesso modo del procedura di firma di documenti, anche la procedura che permette la creazione di una coppia di chiavi e la loro certificazione permette di scegliere tra la vista delle tipologie di token installati e i token effettivamente presenti.

4 Conclusioni

L'architettura della API di Signo permettono una gestione molto semplificata dei token ed una volta scelto il token permettono di astrarre dal tipo di token utilizzato ed eseguire le stesse procedure per firmare e verificare i documenti, accedere alle credenziali contenute e creare nuove credenziali.

Tuttavia accade che alcune Public Key Infrastructure utilizzino i token hardware in modo non standard e talvolta questo fa nascere delle perplessità riguardo alla conformità delle applicazioni proprietarie alla legge italiana. Ad esempio alcune PKI sogliono memorizzare nei token hardware i certificati e/o le chiavi pubbliche come oggetti privati, accessibili solo mediante autenticazione da parte dell'utente. Le applicazioni da essi fornite richiedono all'utente il PIN per l'autenticazione non appena viene rilevato il token e quindi successivamente le operazioni di caricamento delle credenziali e le operazioni di firma dei documenti non richiedono più l'autenticazione. La legge italiana tuttavia specifica che l'utente debba essere pienamente consapevole quando appone la firma su di un documento: la richiesta dell'autenticazione da parte dell'utente ad ogni firma garantisce tale consapevolezza, ma ciò non è ugualmente affermabile altrimenti.
Questo politica di utilizzo pone dei problemi di compatibilità tra Signo e i token hardware così gestiti. Infatti le API di Signo sono state progettate con in mente la modalità di utilizzo standard dei token hardware che consiste nel memorizzare chiavi pubbliche e certificati come oggetti pubblici all'interno del token. Risulta più complicato gestire token hardware utilizzati in modalità diverse, soprattutto visto che Signo permette di utilizzare contemporaneamente più token. Si pone ad esempio il problema di quando e come richiedere l'autenticazione da parte dell'utente e di come accorgersi che certificati e chiavi pubbliche sono memorizzate come oggetti privati piuttosto che come oggetti pubblici.
Per risolvere questo problema è allo studio un meccanismo che utilizzi degli *event listeners* conformi ad una interfaccia, la quale definisca un metodo richiamato dalle API di Signo quando è necessario effettuare il login sul token hardware. In questo modo la libreria in base al profilo del tipo di token istanziato[1], potrà richiedere il login quando ad esempio sia necessario per caricare le credenziali. La modalità con la quale richiedere il PIN potrà essere definita al livello dell'applicazione oppure mediante delle implementazioni di default di event listeners fornite dalle API di Signo[2].

[1] Il profilo di una determinata tipologia di token, descritto mediante le proprietà dell'InputToken, potrebbe ad esempio indicare se il token memorizza i certificati e le chiavi pubbliche come oggetti privati.
[2] Ad esempio gli event listeners di default potrebbero richiedere il PIN utilizzando la modalità testuale o la modalità grafica.

5 Riferimenti bibliografici

[1] Decreto Legislativo 23 Gennaio 2002 n.10, “Attuazione della direttiva 1999/93/CE relativa ad un quadro comunitario per le firme elettroniche”.
[2] Decreto del Presidente della Repubblica 28 Dicembre 2000 n.445, “Testo unico delle disposizioni legislative e regolamentari in materia di documentazione amministrativa”.
[3] Decreto del Presidente della Repubblica 10 Novembre 1997 n.513, Regolamento contenente i criteri e le modalità di applicazione dell'articolo 15, comma 2, della legge 15 marzo 1997, n. 59 in materia di formazione, archiviazione e trasmissione di documenti con strumenti informatici e telematici.
[4] Public-Key Cryptography Standards # 1, “RSA Cryptography Standard”, http://www.rsasecurity.com/rsalabs/pkcs/pkcs-1/index.html.
[5] Public-Key Cryptography Standards # 7, “Cryptographic Message Syntax Standard”, http://www.rsasecurity.com/rsalabs/pkcs/pkcs-7/index.html.
[6] Public Key Cryptography Standards # 10, “Certification Request Syntax Standard”, http://www.rsasecurity.com/rsalabs/pkcs/pkcs-10/index.html.
[7] Public Key Cryptography Standards # 11, “Cryptography Token Interface Standard”, http://www.rsasecurity.com/rsalabs/pkcs/pkcs-11/index.html.
[8] Public Key Cryptography Standards # 12, “Personal Information Exchange Syntax Standard”, http://www.rsasecurity.com/rsalabs/pkcs/pkcs-12/index.html.
[9] Request for Comments 2459, “Internet X.509 Public Key Infrastructure Certificate and CRL Profile”.
[10] IAIK Java Cryptography Extension (IAIK-JCE), http://jce.iaik.tugraz.at/products/01_jce/index.php.
[11] IAIK PKCS#11 Wrapper, http://jce.iaik.tugraz.at/products/14_PKCS11_Wrapper/index.php.