Cohesion Sign - Applet firma

Al fine di supportare le funzionalità di firma per tutte le carte contenenti certificati validi a tal scopo, si è reso necessario riscrivere completamente l'applet originale in quanto questa utilizzava un approccio completamente diverso e non standard per l'accesso ai certificati della carta. La novità principale della nuova applet è l'implementazione di un wrapper per l'interfaccia PKCS#11 (standard di interfaccia di tutte le librerie proprietarie e non, per l'accesso alle operazioni della carta) che garantisce la possibilità di interfacciarsi con tutte le librerie per smartcard. A differenza di tutte le applet open source presenti in rete per la firma digitale, tale wrapper è scritto in java mediante tecnologia JNA (Java Native Access), permettendo così interoperabilità con tutti i sistemi operativi dotati di librerie PKCS#11. La nuova applet fornisce inoltre supporto sia in firma singola che multipla anche al formato p7m, mediante il quale è possibile firmare non solo file pdf ma ogni tipo di documento. La firma p7m è nel formato standard CaDes, mentre la firma pdf nello standard PaDes.

Di seguito il dettaglio di tutte le nuove funzionalità implementate:

- Utilizzo di un **modulo PKCS#11** per l'accesso alla smartcard scritto in Java **mediante tecnologia JNA**. La vecchia applet utilizzava il JCE standard di java per interfacciarsi alle librerie della carta ed estrarre il keystore dalla smartcard contenente la chiave privata. Tale chiave veniva usata per la firma la quale era calcolata tramite codice Java. Molti driver però non supportano l'estrazione del keystore in quanto non è definito per lo standard PKCS#11. Il modo corretto di firmare un dato usando questo standard prevede invece di richiamare la funzione "*sign*" esposta dalla carta la quale gestirà internamente le operazioni di firma di un dato hash, senza esporre la propria chiave privata. L'interfaccia per il collegamento con le librerie native della carta è stata creata usando JNA (Java Native Access) che permette di accedere alle funzioni di librerie scritte in formato nativo del SO senza il prerequisito che queste siano state compilate con il supporto a JNI (Java Native Interface). In questo modo si è evitato di utilizzare librerie native aggiuntive esterne supportanti JNI, con la sola funzione di wrapper tra codice JAVA e libreria PKCS#11 nativa (modo attualmente in uso da tutte le applet di firma digitale analizzate).

- **Supporto di tutti i SO e smartcard** dotati di libreria in formato standard PKCS#11. L'applet non contiene internamente librerie scritte in codice nativo ed è quindi usabile su ogni SO. Le librerie di accesso alla smartcard però, debbono ovviamente essere presenti nel pc dell'utente. E' sufficiente passare come parametro la lista di tutte le librerie PKCS#11 native (.dll, .so e .dylib) conosciute affinché l'applet selezioni automaticamente quella corretta da utilizzare sulla base della carta inserita e del SO in uso.

- **Riconoscimento certificati multipli**: Possibilità di scegliere il certificato da usare per la firma tra tutti quelli trovati nella carta. Saranno visibili in particolare solo i certificati designati allo scopo di firma digitale e non ripudio, favorendo questi ultimi qualora siano disponibili.

- **Riconoscimento smartcard multiple**: l'applet è in grado di rilevare i certificati di tutte le smartcard collegate contemporaneamente al pc e visualizzarli nell'apposita lista.

- **Firma massiva**: l'applet permette la firma di più documenti contemporaneamente, sia in modalità silent come la vecchia applet (i documenti base64 vengono passati all'applet mediante parametro o tramite apposita funzione) che interattiva (l'utente può selezionare tramite interfaccia grafica tutti i documenti da firmare).

- **Firma p7m**: Se il documento che si vuole firmare non è un pdf, l'applet ne permette comunque la firma utilizzando il formato PKCS#7 (file di tipo p7m). Se il documento da firmare è già un p7m contenente altre firme, allora verrà aggiunta la firma a quelle già esistenti (senza invalidare le altre). E' inoltre possibile specificare di salvare comunque un pdf in formato p7m. La firma sarà effettuata in formato standard CAdES.

- **Firma Pdf**: La firma di documenti Pdf risulta valida su tutti gli applicativi di verifica firma come Adobe Reader o FirmaCerta. Se il pdf è già firmato, la firma viene affiancata a quella già esistente e nel caso si scelga di mostrare la firma, verrà automaticamente identificata la posizione migliore in cui posizionarla in modo da non sovrapporsi alle firme preesistenti. La firma sarà effettuata in formato standard PAdES.

- **Firma ristretta a determinati utenti**: Come per la vecchia applet è possibile specificare come parametro, il codice fiscale dell'utente che può firmare. Tale codice fiscale viene confrontato con quello estratto dal certificato nella smartcard e solo se coincidono è possibile effettuare la firma con quel certificato. In questa nuova applet la procedura di estrazione del codice fiscale dalla carta è stata resa universale e compatibile con tutti i certificati.

- **Firma verificata**: l'applet verifica automaticamente la firma appena generata e se questa non è valida genera un avviso e non crea il documento firmato, in quanto sarebbe inutilizzabile.

- **Firma in modalità SHA-256**. Sia per la firma pdf che p7m l'hash della firma è calcolato in SHA-256.

- **Data di firma sicura**. La data e l'ora della firma vengono ottenuti da un server sicuro e non compromettibile. Questa viene firmata insieme all'hash così da garantirne la correttezza. L'utente non può così in alcun modo eludere la scadenza del certificato e l'anticipazione della scadenza del documento impostando l'ora indietro nel proprio pc. La data restituita è firmata e riconosciuta valida dall'applet solo se firmata con un certificato pre-impostato e fisso.

- **Controllo host** che fornisce l'applet. Al fine di evitare che l'applet venga scaricata e utilizzata in altri ambiti e siti, è stato integrata una funzionalità di verifica dell'host. Se questo non è tra i censiti allora l'applet non sarà utilizzabile.

- **Firma da codice**: L'applet espone metodi di firma utilizzabili direttamente da codice lato browser tipo Javascript illustrati di seguito.

- **Caricamento di documenti già firmati**: Tramite un apposito bottone è possibile selezionare documenti firmati esternamente e restituirli al browser nello stesso formato utilizzato dall'applet per la firma. L'applet si preoccuperà di controllare che i documenti da caricare siano in un formato supportato e contengano una firma valida per lo specifico utente.

- **Firma di documenti di grande dimensione:** L'applet permette di aggirare i limiti di lunghezza dei dati inviati da javascript verso l'applet e viceversa, mediante metodi di invio incrementale dei dati da firmare e ricezione incrementale dei dati firmati.

- **Accesso Facilitato:** L'applet è appositamente sviluppata per poter essere utilizzata anche da utenti non vedenti. Ogni volta che si richiede un interazione con l'utente (ad esempio inserimento del PIN o pressione di bottoni di conferma) viene riprodotto un segnale sonoro. Tale segnale è un singolo "beep" qualora interazione è relativa al normale flusso di azioni da svolgere, mentre è un doppio "beep" in caso di messaggi di errore. Tutti i bottoni sono accessibili da tastiera; in particolare i campi principali di interazione sono impostati con il focus abilitato di default (es: bottone firma, casella di inserimento PIN, ecc.).

L'applet ha due modalità di funzionamento:

- **Grafica**: L'applet mostra un interfaccia di scelta del certificato e firma, richiamabile mediante il bottone di firma dell'applet o direttamente da codice Javascript.
- **Da Codice**: L'applet espone dei metodi utilizzabili da Javascript per firmare i documenti.

Per entrambe le modalità i dati in input da firmare vanno caricati nell'applet in uno specifico formato ed obbligatoriamente tramite una o più chiamate javascript al metodo pubblico dell'applet "*updateDataToSignBase64(String dataToSignBase64)*".

Il parametro richiesto da tale funzione è una stringa contenente il documento da firmare codificato in Base64. Si possono firmare più documenti separandoli con 3 simboli "*%*" ("*%%%*"). Inoltre ogni documento può contenere un identificativo (usualmente il percorso) prima del contenuto Base64 separato da 1 "*%*". Questo formato sarà lo stesso utilizzato anche per i documenti firmati restituiti in output. Es:"*File_Path_1%Contenuto_File_1_Base64%%%Contenuto_File_2_Base64%%%File_Path_3%Contenuto_File_3_Base64* ";

Tale metodo permette un caricamento incrementale della stringa così da evitare il limite fisico sulla lunghezza massima di caratteri trasferibili da Browser ad Applet che si presenta durante la firma di documenti di grandi dimensioni. Se la stringa da caricare è eccessivamente grande è necessario quindi prima dividerla in parti usando una funzione javascript come la seguente che ritorna un array di stringhe di lunghezza "*length*":

```
String.prototype.splitInArrayOfLength = function (length) {
   var str = "[\\s\\S]{1,"+length+"}";
   var regex = new RegExp( str, 'g' );
   return this.match(regex) || [];
};
```

Queste parti andranno caricate nell'applet sequenzialmente nel seguente modo:

```
var dataToSignBase64 = "very big string base64 %%% second File %%% ecc..";
//Cancello i dati precedentemente caricati, cosi da poter procedere con l'incremento
document.myApplet.clearDataToSignBase64();
//Divido la stringa in array di 1000000 caratteri (circa 1Mb)
var dataToSignBase64PartArray = dataToSignBase64.splitInArrayOfLength(1000000);
for(i=0;i<dataToSignBase64PartArray.length;i++)
   document.myApplet.updateDataToSignBase64(dataToSignBase64PartArray[i]);
```

Qualora si volessero caricare documenti da firmare dal computer locale dell'utente è possibile utilizzare la funzione "*loadLocalFileToSign()*" che si preoccuperà di far selezionare all'utente i documenti da firmare e caricarli nell'applet nel corretto formato di input per la firma. Tale funzione restituirà "*true*" o "*false*" a seconda se siano stati caricati o meno, documenti da firmare.

```
var ret = document.myApplet.loadLocalFileToSign();
if(ret)
   alert("Documenti caricati correttamente");
```

I parametri che l'applet richiede per entrambe le modalità di funzionamento sono i seguenti:

- "**java_arguments**" con valore: *"-Xmx1024m"*: Necessario per firmare grandi file; richiede una versione di java maggiore di 1.6 update 10.

- "**PathDll**": Contiene i nomi delle librerie PKCS11 di accesso alla smartcard, per ogni sistema operativo, separati dal simbolo "*%*". Può essere specificato o solo il nome della libreria (ed in tal caso verrà ricercata nei percorsi più comuni) o tutto il percorso. E' inoltre possibile richiamare una pagina web appositamente predisposta che ritorni la lista nel formato appena descritto. Per utilizzare questa funzionalità è sufficiente inserire in questo parametro, l'url http di tale pagina. In tal caso, l'applet effettuerà una chiamata GET a questa pagina recuperando la lista. Le librerie da usare verranno scelte automaticamente in base al sistema operativo dell'utente. Se nessuna libreria viene specificata o il parametro non è presente allora è possibile scegliere di usare una lista prevaricata internamente, o scegliere manualmente la libreria da utilizzare direttamente dall'applet nel pc dell'utente. La scelta tra queste due modalità di utilizzo è definita dal parametro "*useInternalDllList*" specificabile solo mediante chiamata javascript e impostato di default a "*true*".

- "**FirmaPDFComeP7m**": se impostata a "*si*" firmerà i pdf in formato P7M, quindi la firma non verrà integrata nel PDF e i parametri "*FirmaVisibile*" e "*NumeroPagina*" non verranno considerati. Impostando "*no*" la firma verrà integrata nel pdf cercando di non sovrapporsi alle firme precedenti qualora ci siano; se il parametro è vuoto o non presente il valore di default è "*no*".

- "**FirmaVisibile**": Se il valore è "*si*" viene mostrata la firma nel pdf ; se il parametro è vuoto o non presente il valore di default è "*no*".

- "**NumeroPagina**": se impostata seleziona il numero di pagina del pdf su cui applicare la firma se questa è visibile; se il parametro è vuoto o non presente o uguale a "*-1*" e la firma è visibile allora questa viene applicata all'ultima pagina.

- "**PosizioneFirma**": Permette di specificare la posizione della firma nella pagina del pdf. I valori possibili sono "*Sinistra*" e "*Destra*". La posizione della firma usando questi parametri sarà la stessa della vecchia applet di firma. Nel caso il parametro sia vuoto, non presente o diverso da uno dei due valori accettati, allora verrà scelta una posizione che non si sovrapponga ad altre firme.

- "**TimeServerUrl**": Contiene l'indirizzo della pagina certificata da cui prendere la data da impostare sulla firma; la pagina controlla anche che l'host su cui è caricata l'applet sia censito ed abilitato all'utilizzo. Qualora la connessione all'url necessiti di un proxy, questo sarà automaticamente impostato sulla base del proxy di sistema (che deve quindi essere settato).

Nel caso si scelga di utilizzare la modalità grafica è inoltre possibile settare i seguenti parametri:

- "**ControlloCodiceFiscaleFirmatario**": se impostato permette di limitare la firma al solo utente con codice fiscale specificato. Il controllo è valido anche per il caricamento di documenti già firmati.

- "**SalvataggioLocale**": se impostato a "*si*" verrà chiesto dove salvare il file firmato nel pc, altrimenti verrà richiamata la funzione "*PassaFileFirmato(dataManagerObject)*" (che deve essere presente nella stessa pagina dell'applet). L'applet una volta conclusa la firma passa a questa funzione un oggetto java DataManager che contiene metodi per la lettura dei dati firmati in modo incrementale. La seguente funzione javascript mostra un esempio di lettura di tali dati a partire dall'oggetto dataManager:

```
function getDataSignedBase64(dataManagerObject){
   var dataSignedBase64 = "";
   var tmp = "";
   //Imposto quanti caratteri leggere ogni ciclo
   var limit = 1000000;
   do{
      tmp = dataManagerObject.getPartialDataSignedBase64(limit);
      dataSignedBase64 += tmp;
   }while(tmp != "");
   //Resetto il contatore incrementale per successive letture
   dataManagerObject.resetGetPartialDataSignedBase64();
   return dataSignedBase64;
}

function PassaFileFirmato(dataManagerObject){
   var dataSignedBase64 = getDataSignedBase64(dataManagerObject);
```

La variabile "*dataSignedBase64*" ritornata sarà dello stesso formato dei dati in ingresso; se il parametro è vuoto o non presente il valore di default è "*no*".

- "**MostraTuttiICertificati**": Se impostato a "*si*" verranno mostrati nella lista dei certificati, tutti quelli presenti nelle carte connesse, abilitati alla firma digitale (compresi quelli di autenticazione). Se

impostato a "*no*", verrà data priorità ai soli certificati di firma (non ripudio). Qualora questi non siano presenti in nessuna delle carte allora verranno mostrati tutti i certificati. Se il parametro è vuoto o non specificato il valore di default è "*no*".

- "**SignButtonText**": Se impostato, il parametro definisce il testo da visualizzare nel bottone che mostrerà la schermata di firma. Se il parametro è vuoto o non presente il bottone sarà nascosto.

- "**UploadButtonText**": Se impostato, il parametro definisce il testo da visualizzare nel bottone che permette il caricamento di documenti già firmati. Il bottone permette di scegliere documenti dal pc dell'utente e se questi sono in una formato corretto (p7m o pdf firmati), li passa alla funzione "*PassaFileFirmato(dataManagerObject)*" nello stesso formato dei dati restituiti dalla firma. Qualora sia valorizzato il parametro "*ControlloCodiceFiscaleFirmatario*", l'applet controllerà che ogni documento sia stato firmato in modo valido con un certificato relativo all'utente con codice fiscale specificato. Se il parametro "*ControlloCodiceFiscaleFirmatario*" è vuoto o non specificato allora ogni documento sarà valido se tutte le firme contenute risulteranno valide. Se il parametro è vuoto o non presente il bottone sarà nascosto.

Per l'utilizzo della firma da codice Javascript, l'applet espone i seguenti metodi :

- **void updateDataToSignBase64(String dataToSignBase64):** La funzione permette di caricare in maniera incrementale i dati da firmare come precedentemente descritto.
  Es:
  ```
  document.myApplet.updateDataToSignBase64(dataToSignBase64PartArray[i]);
  ```

- **void clearDataToSignBase64():** La funzione resetta tutti dati da firmare precedentemente caricati tramite le funzione "*updateDataToSignBase64*" così da poter procedere ad un nuovo caricamento incrementale.
  Es:
  ```
  document.myApplet.clearDataToSignBase64();
  ```

- **boolean loadLocalFileToSign():** La funzione si occuperà di far selezionare all'utente i documenti da firmare e caricarli nell'applet nel corretto formato di input per la firma. Tale funzione restituirà "*true*" o "*false*" a seconda se siano stati caricati o meno, documenti da firmare.
  Es:
  ```
  var ok = document.myApplet.loadLocalFileToSign();
     if(ok)
        alert("Documenti caricati correttamente");
  ```

- **void showSignFrame():** La funzione permette di richiamare tramite codice javascript la finestra grafica di scelta dei certificati e firma nello stesso modo del bottone di firma interno all'applet.
  Es:
  ```
  document.myApplet.showSignFrame();
  ```

- **String[] getCertificateList()**: restituisce un array degli ID dei certificati trovati dall'applet in tutte le smartcards inserite dando priorità ai soli certificati di firma (non ripudio). Qualora questi non siano presenti in nessuna delle carte allora verranno mostrati tutti i certificati.

  Es:

```
var certList = document.myApplet.getCertificateList();
for (i=0;i<certList.length;i++)
   alert(certList[i]);
```

- **String[] getAllCertificateList()**: restituisce un array di tutti gli ID dei certificati presenti nelle carte connesse, abilitati alla firma digitale (compresi quelli di autenticazione).

  Es:

```
var allCertList = document.myApplet.getAllCertificateList();
for (i=0;i<allCertList.length;i++)
   alert(allCertList[i]);
```

- **String getCFFromCertificate(String idCert):** La funzione, dato l'identificativo di un certificato, tenta di estrarne il Codice Fiscale.

  Es:

```
var cfLettoDaCert = document.myApplet.getCFFromCertificate(certList[0]);
```

- **X509Certificate getX509Certificate(String idCert):** La funzione, dato l'identificativo di un certificato, lo restituisce come classe java java.security.cert.X509Certificate. Tramite l'oggetto restituito sarà poi possibile tramite javascript ottenere tutti i dati del certificato, richiamando semplicemente i metodi java standard per la classe X509Certificate (come ad Es: getSubjectDN()).

  Es:

```
var certificatoX509 = document.myApplet.getX509Certificate(certList[0]);
var soggetto = certificatoX509.getSubjectDN().getName();
alert("Nome soggetto del certificato letto: " + soggetto);
```

- **boolean checkCF(String idCert, String CF)**: La funzione dato l'identificativo di un certificate, e un codice fiscale, controlla se il certificato è relativo all'utente con CF specificato. Serve a limitare la firma all'utente specificato.

  Es:

```
var cfOK = document.myApplet.checkCF(certList[0], "FLCDMN85D05E783N");
if(cfOK)
   alert("Il certificato è relativo al cittadino FLCDMN85D05E783N");
```

- **boolean checkValidity(String idCert)**: La funzione controlla se il certificato relativo all'id passato come parametro, è scaduto o meno. La verifica è eseguita sulla base della data presa dal pc dell'utente e non dovrebbe essere bloccante per il processo di firma, ma si dovrebbe mostrare solo un avviso di certificato scaduto.

  Es:

```
var certificateValidityOK = document.myApplet.checkValidity(certList[0]);
if(certificateValidityOK)
   alert("Il certificato selezionato è attualmente valido");
```

- **boolean checkIsSelfSigned(String idCert)**: La funzione controlla se il certificato relativo all'id passato come parametro, è Self Signed o meno. La verifica non dovrebbe essere bloccante per il processo di firma, ma si dovrebbe mostrare solo un avviso di certificato Self Signed.

  Es:

```
var certificateSelfSigned = document.myApplet.checkIsSelfSigned(certList[0]);
if(certificateSelfSigned)
   alert("Il certificato selezionato è Self Signed");
```

- **boolean checkIsRevoked(String idCert)**: La funzione controlla se il certificato relativo all'id passato come parametro, è stato revocato o meno. La verifica non dovrebbe essere bloccante per il processo di firma, ma si dovrebbe mostrare solo un avviso di certificato revocato.

  Es:

```
var certificateRevoked = document.myApplet.checkIsRevoked(certList[0]);
if(certificateRevoked)
   alert("Il certificato selezionato è stato revocato");
```

- **checkIsNonRepudiation(String idCert)**: La funzione controlla se il certificato relativo all'id passato come parametro, è di tipo "Non Ripudio" e quindi valido per la firma a norma di legge. Se il certificano non è di tipo "Non Ripudio", allora è di tipo "Autenticazione", ovvero abilitato alla firma, ma non a norma di legge.

  Es:

```
var certNonRepudiation = document.myApplet.checkIsNonRepudiation(certList[0]);
if(certNonRepudiation)
   alert("Il certificato selezionato è valido per firma a norma di legge");
```

- **DataManager signMultipleDocuments(String idCert)**: Il metodo firma i documenti caricati con il metodo "*updateDataToSignBase64*" usando il certificato specificato. Il metodo restituisce un oggetto "*DataManager*" che può essere processato per ottenere i dati firmati, usando la funzione "*PassaFileFirmato*" descritta precedentemente.

  Es:

```
var dataManagerObject = document.myApplet.signMultipleDocuments(certList[0]);
if(dataManagerObject.containSignedData())
   PassaFileFirmato(dataManagerObject);
```

- **DataManager uploadSignedData():** La funzione permette di caricare uno o più documenti già firmati restituendoli nell'oggetto DataManager nello stesso formato dei documenti firmati dall'applet. Tale oggetto restituito può quindi essere processato usando la funzione "*PassaFileFirmato*" descritta precedentemente. Questo metodo offre le stesse funzionalità del bottone di caricamento di documenti firmati fornito dall'applet e descritto precedentemente, inclusi i controlli sul codice fiscale del firmatario.

  Es:

```
var dataManagerObject = document.myApplet.uploadSignedData();
if(dataManagerObject.containSignedData())
   PassaFileFirmato(dataManagerObject);
```

- **String getSignedData():** La funzione permette di ottenere, a seguito della firma, la stringa "*dataSignedBase64"* senza passare per l'oggetto "*DataManager*" che così non dovrà essere gestito da codice javascript. La stringa restituita sarà intera e quindi nel caso di dati di grandi dimensioni potrebbe non funzionare.

  Es:

```
var dataSignedBase64 = document.myApplet.getSignedData();
```

- **String getPartialSignedData([int lenght]):** La funzione permette di ottenere incrementalmente, a seguito della firma, la stringa "*dataSignedBase64"* senza passare per l'oggetto "*DataManager*" che così non dovrà essere gestito da codice javascript. La funzione opzionalmente accetta per parametro la lunghezza massima della stringa restituita. Se questa non è specificata il valore di default è 1000000 caratteri (circa 1Mb).

  Es:

```
var dataSignedBase64 = "";
var tmp = "";
do{
   tmp = document.myApplet.getPartialSignedData();
   //uso un valore diverso dal default (1000000)
   //tmp = document.myApplet.getPartialSignedData(5000000);
   dataSignedBase64 += tmp;
}while(tmp != "");
if(dataSignedBase64 != "")
   alert("N°" + dataSignedBase64.split("%%%").length + " Documenti firmati");
```

- **void resetGetPartialSignedData():** La funzione permette di resettare l'offset incrementale utilizzato dalla funzione "*getPartialSignedData*" così da poterla richiamare successivamente per una nuova lettura.

  Es:

```
document.myApplet.resetGetPartialSignedData();
```

L'applet permette inoltre di definire il valore delle principali variabili interne direttamente da javascript. Il loro formato è lo stesso relativo e definito nei parametri già descritti.

Es:

```
document.myApplet.variabile_pubblica=valore;
```

Le variabili pubbliche che è possibile impostare sono le seguenti (il nome è esplicativo e il formato è lo stesso dei rispettivi parametri):

- **dllList:** default stringa vuota**.**
- **timeServerUrl:** default stringa vuota**.**
- **codiceFiscaleFirmatario:** default stringa vuota.
- **localSave:** default false.
- **signPdfAsP7m:** default false.
- **pageNumToSign:** default -1.
- **visibleSignature:** default false.
- **signPosition:** default stringa vuota.

- **signLocalFiles:** default false.
- **showAllCerts:** default false.

Oltre a queste variabili relative ai parametri, è poi possibile impostare le seguenti variabili pubbliche aggiuntive:

- **useInternalDllList:** Permette di definire qualora la lista delle dll sia vuota, se usare la lista prevaricata interna o chiedere all'utente il percorso della libreria nel pc. Default true.

- **blockSignatureForInvalidCertificates:** Permette di specificare di bloccare la procedura di firma in caso di scelta di un certificato non valido (es. scaduto, revocato, self-signed o non di firma), altrimenti verrà mostrato solo un avviso ma si potrà procedere con la firma. Default false.

- **uploadNotSigned:** Permette di richiamare la procedura di upload anche per documenti non firmati. Default false.

- **useJna:** Permette di specificare se utilizzare l'engine JNA o il più classico IAIK per le operazioni di interfacciamento alla smart-card. Default true.

Se si sceglie di utilizzare solo la modalità da codice è possibile nascondere i bottoni dell'applet anche impostando il campo '`style="visibility:hidden;"`' nel tag APPLET.

Note relative alla dimensione dei documenti da firmare:

I limiti relativi alla dimensione dei dati da firmare dipendono esclusivamente dalla memoria RAM disponibile nel PC del firmatario e riservata all' heap space di java. Non c'è una limitazione sul numero, ma bensì sulla dimensione totale di tutti i file passati per la firma.
Dato che la firma di documenti online tramite smart-card è possibile solo tramite applet Java, purtroppo (e nonostante l'ottimizzazione del codice), l'overhead dovuto alla gestione interna delle stringhe e dei file causa un uso massivo della memoria anche per documenti non eccessivamente grandi. In particolare, a seguito di diversi test, il rapporto tra la dimensione totale dei documenti caricati e la memoria occupata nell'heap space java è risultato essere di circa 1 a 15 (test effettuato con un documento da 5 Mb che ha richiesto circa 80Mb liberi, e uno da 40 Mb che ha richiesto circa 600 Mb liberi nell'heap space).

Potete trovare una pagina ti test di funzionamento dell'applet di firma a questo link :
http://cohesion.regione.marche.it/SignApplet/SignAppletTest.html