LIBERSIGN

APPLETS DE SIGNATURE ELECTRONIQUE

Spécifications techniques détaillées
& Manuel d'exploitation

Description du document :

Nom de cette version	**API_libersign_v1.9**
Date de cette version	**mercredi 8 avril 2012**
Nom de la 1ère version	API_libersign_v1.0
Date de la 1ère version	26-09-2008

Historique des versions :

Date	Objet / modifications	Version
26-09-2008	Descriptif des spécifications techniques de libersign	v-1.0
21-11-2008	Enrichissement de l'API pour la signature de PES v2	v-1.2
15-03-2009	Schémas et mode opératoire pour le packaging de l'applet	v-1.3
15-05-2009	Changement d'API pour signature par lot multi-format	v-1.4
05-10-2009	Changement d'API: permette la co-signature XAdES	v-1.5
11-01-2010	Évolution d'API: co-signature dans un même fichier PKCS#7	v-1.6
08-03-2010	Correction d'API sur co-signature dans un même fichier PKCS#7	v-1.7
28-06-2010	Précisions sur le packaging, remise en forme	v-1.8
09-12-2010	Évolution d'API : signatures CMS/PKCS#7 horodatées RFC-3161	v-1.9

Auteurs : Xavier Facelina (NETHEOS), Stéphane Vast (ADULLACT Projet)
Tél : 04 67 65 96 44 Fax : 04 67 65 93 92
Emails: x.facelina@netheos.net, stephane.vast@adullact-projet.coop

Table des matières

1. Présentation de Libersign

Ce document définit la structure, l'environnement et le fonctionnement de deux Applets JAVA impliquées dans le fonctionnement d'un système de signature électronique de documents.

- une applet de signature ,
- une applet de vérification de signature.

Il y a également une application serveur permettant de fournir les condensats à l'applet de signature.

Ces libriciels (logiciels libres) sont déposés sous licence CeCILL V2 sur la Forge de l'ADULLACT, les codes sources sont accessibles à l'adresse:

http://adullact.net/projects/libersign/

2. Applet de signature

2.1. Description

En matière de signature électronique, il est souhaitable que tout ou partie des opérations de signature électronique soit effectué sur le poste client.
L'usage d'applets JAVA ou de contrôles ActiveX (sur plate forme Microsoft™ IE) est courant en la matière.

L'applet JAVA qui nous concerne réalise les opérations suivantes :

- signature d'un ou plusieurs documents ;
- signature renvoyée au format CMS/PKCS#7 ou XAdES ;
- **signature fragmentée** (hash côté serveur et chiffrement côté utilisateur) ;
- sélection automatique du certificat de signature en fonction du certificat d'authentification au système (paramètres de l'applet), ou à défaut sélection par l'utilisateur du certificat de signature ;
- selon paramètre :
  - publication des signatures sur une adresse URL de type HTTPS (mode 'http'),
  - ou mise à disposition de(s) signature(s) pour utilisation dans un champ de formulaire HTML (mode 'form').

La signature est réalisée sur le poste client à partir de document(s) hébergé(s) a-priori sur un serveur.

Afin d'optimiser les temps de transfert et ne pas trop faire patienter le signataire, le processus est « fragmenté », seul le condensat SHA-1 du document est transmis à l'applet.

2.2. Packaging de l'applet de signature

2.2.1. Objectifs du packaging de l'applet

L'objectif est de fournir à l'applet l'ensemble des informations nécessaires à son bon fonctionnement. Pour l'heure, il s'agit exclusivement des CRL et des certificats des AC reconnues.

2.2.2. Description des fichiers contenus

Le packaging de l'applet doit regrouper, l'applet exécutable (ensemble de classes JAVA compilées), un *keystore* regroupant les certificats des ACs nommé « **ac-truststore.jks** » et un fichier contenant la liste de CRL à consulter, nommé « **crl-list.conf** ».

- ac-truststore.jks :
  - Ce fichier est un keystore java au format JKS (non JCEKS?) renfermant les certificats X.509 (publics) des AC reconnues (« de confiance ») de l'applet.
  - Il peut contenir autant de certificats que nécessaire.
  - Son mot de passe doit être défini en amont (ac-truststore.password) et connu de

l'applet

- crl-list.conf:
  - Il s'agit d'un fichier texte encodé en UTF-8.
  - Il contient une URL par ligne pointant directement sur des fichiers CRL
  - Ces URL sont des URL HTTP

Remarque: Définition des chemins d'appel de ces ressources
Les fichiers doivent être à la racine du CLASSPATH (pour des raisons de simplicité lors du « repackaging »). Ainsi ils seront appelés de la manière suivante:

this.class.getResourceAsStream(« /ac-truststore.jks »); (inputstream vers le keystore)

this.class.getResourceAsStream(« /crl-list.conf »); (inputstream vers le fichier de configuration des CRL)

2.2.3. Opération de packaging

Pré-requis:

- En présence d'une version clé-en-main, l'applet a été compilée avec deux fichiers d'exemple présents à la racine du classpath.
- Dans le cas contraire (par exemple après téléchargement du code source), les fichiers `ac-truststore.jks` et `crl-list.conf` ne sont naturellement pas présents. Il faut les créer et les inclure dans le projet de compilation.

Manuel:

1. « Dé-zipper » le ficher '.jar' de l'applet
   A réception de l'applet compilée, il convient de « dé-zipper » le fichier .jar de l'applet dans un répertoire, dans lequel les opérations suivantes seront effectuées.

2. Remplacer les 2 fichiers sus-nommés
   Deux fichiers d'exemple nommés 'ac-truststore.jks' et 'crl-list.conf' sont présents à la racine du répertoire.
   Il convient de les remplacer par les fichiers à jour.

3. Création du nouveau fichier .jar de l'applet
   « Re-zipper » le répertoire mis à jour, puis renommer le fichier '.zip' obtenu en '.jar'.

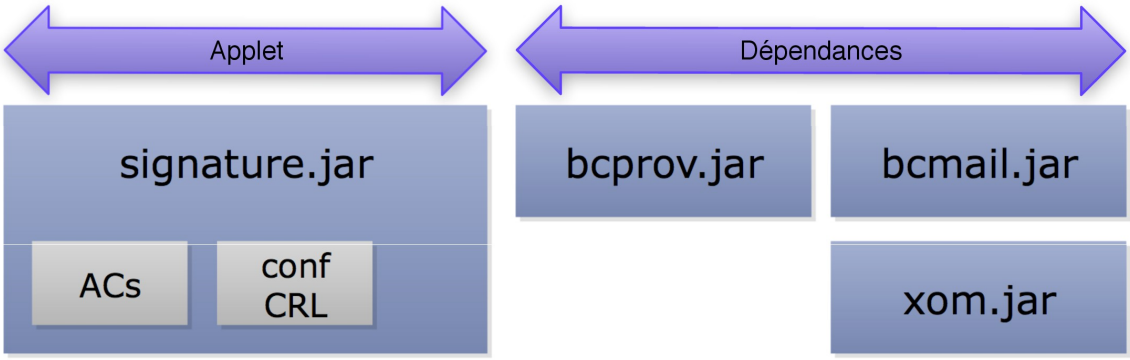
4. Signature du fichier '.jar' de l'applet
   Le fichier de l'applet ayant été modifié, sa précédente signature n'est plus valide. Il convient donc de le signer à nouveau:

   ```
   jarsigner -keystore keystore.ks -storepass motDePasse -keypass password applet.jar nom_de_la_clé_à_utiliser
   ```

2.2.4. Spécifications initiales et exigences

Les éléments à prendre en compte à l'intégration ou au déploiement sont présentés ici.

Schéma de packaging de l'applet



Ce schéma de packaging met en avant le fait que **les certificats d'AC de confiance sont embarqués dans le fichier JAR de l'applet au moment du packaging**, avant la signature de l'applet elle-même.

Le fait que la liste des AC de confiance et l'adresse URL optionnelle d'une CRL locale soient signées par l'exploitant du système constitue une brique importante en matière de sécurité.

En matière de validation de certificat numérique X.509, l'utilisation d'OCSP n'est pas envisagée, ce protocole n'étant pas plus économe en bande passante que la solution présentée ici.

Le fait que ces données soient contenues au sein même de l'applet ne constitue pas une nécessité. Leur externalisation dans une dépendance spécifique (par ex. security-env.jar) serait tout aussi efficace. Quoiqu'il en soit, l'ensemble des jars doivent être signés par l'exploitant. Cette signature du code est indispensable au fonctionnement de l'applet et sa gestion fait partie de la politique de sécurité du système.

Bien que cette politique de sécurité soit à la charge de l'exploitant, nous attirons votre attention sur l'importance de cette problématique (confiance des postes utilisateur dans un certificat de signature d'application, habilitation d'usage de ce certificat par l'exploitant, politique de sécurité JAVA liées aux applets et plus particulièrement aux applets signées via ce certificat...).

Attention : Il existe une exception en matière de signature d'application : les "cryptography providers" de JAVA. Ces fichiers 'jar' doivent être signés par un éditeur de cryptography reconnu par SUN/ORACLE et ne doivent pas, en conséquence, être signés par l'exploitant. C'est le cas des dépendances *bcprov.jar* bcmail et bctsp.

2.3. Retour sur les ACs de confiance

(les "Autorités de confiance" de confiance !)

N'importe quelle autorité de confiance n'est pas de confiance du point de vue de l'exploitant. Seule une liste exhaustive d'autorités de confiance bénéficie de ce privilège. Cette liste d'autorités de confiance est embarquée dans l'applet JAVA sous la forme d'un **fichier de type keystore** (.ks au format JCEKS). Ce keystore est de fait un conteneur de certificats des AC de confiance.

Notons que chaque certificat d'AC de confiance comprend normalement un champs avec l'adresse URL de la CRL de l'AC. Ce champs sera utilisé pour vérifier la validité du certificat au moment de la signature (ou de la vérification).

Le **fichier de configuration local-crl-list.conf** comprend un paramètre "local-crl" qui peut être renseigné avec l'adresse URL d'une CRL locale.
Cela répond au besoin de déclarer invalide un certificat auprès du système (une personne n'a plus le droit de se connecter, de signer ou de vérifier) sans pour autant que le certificat de la personne soit révoqué par son AC émettrice (elle n'a pas changé d'identité, a payé sa redevance).
Son utilisation n'est dont pas nécessaire, c'est une commodité pour l'exploitant.

2.4. Constitution d'un magasin d'AC

A partir des codes sources, il n'y a pas de magasin de clé 'ac-truststore.jks'. La méthode proposée repose sur l'usage de l'outil **Porte-Cle** (logiciel libre JAVA, testé en version : 1.5).

https://sourceforge.net/projects/portecle

Créer un nouveau keystore format JKS, si il n'existe pas, au format JKS.

L'exemple ci-après montre l'enrichissement du truststore avec une nouvelle AC :

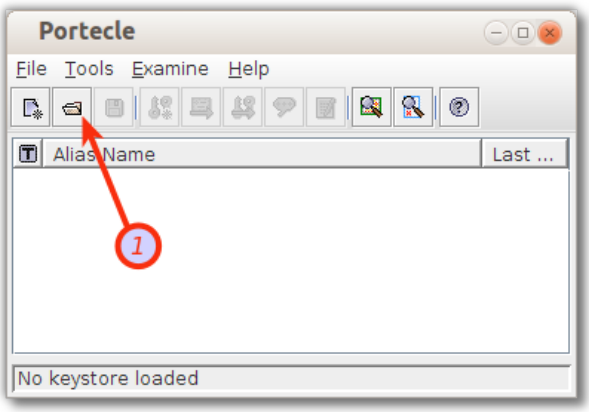
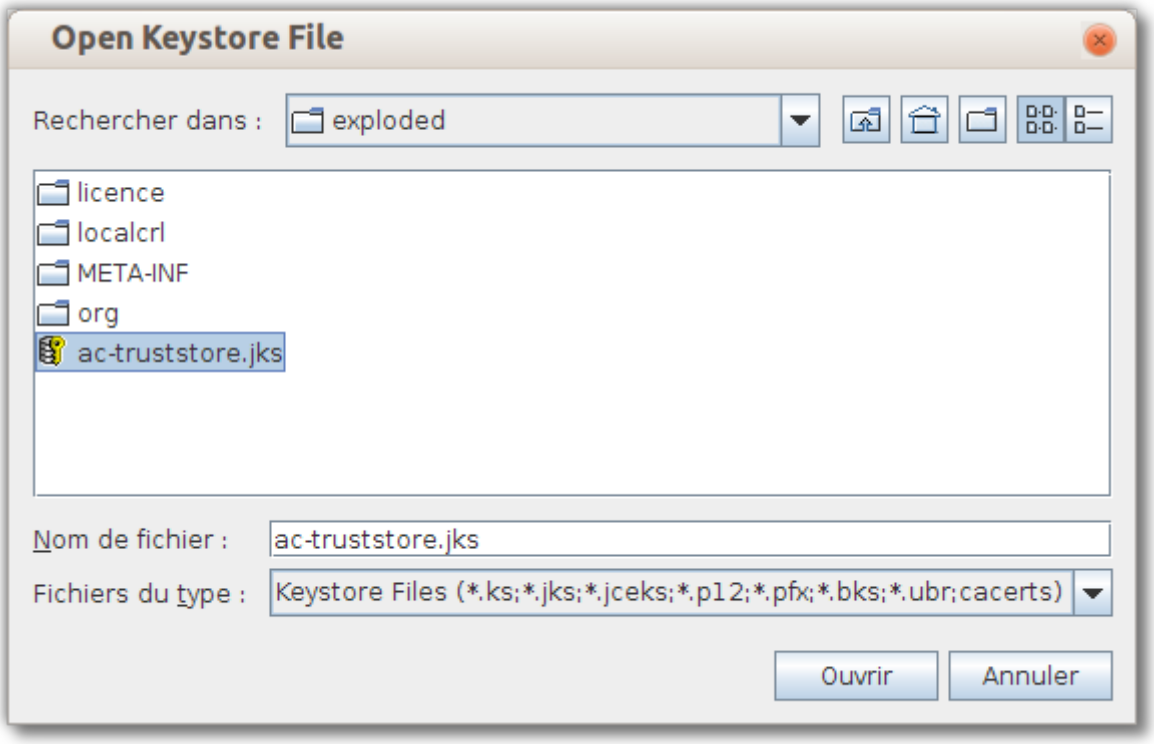


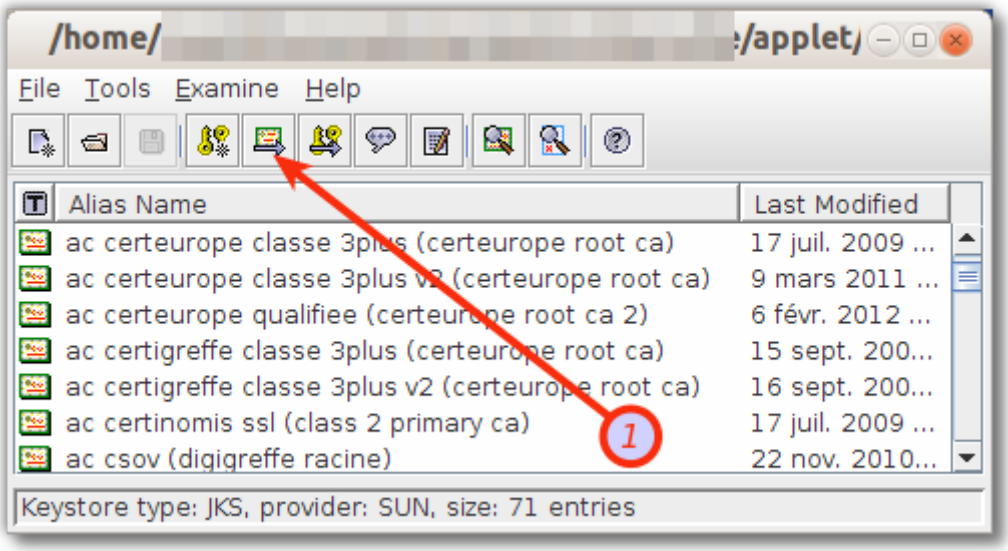
Figure : Le logiciel Portecle

L'ouverture d'un keystore se fait en cliquant sur le bouton (1).

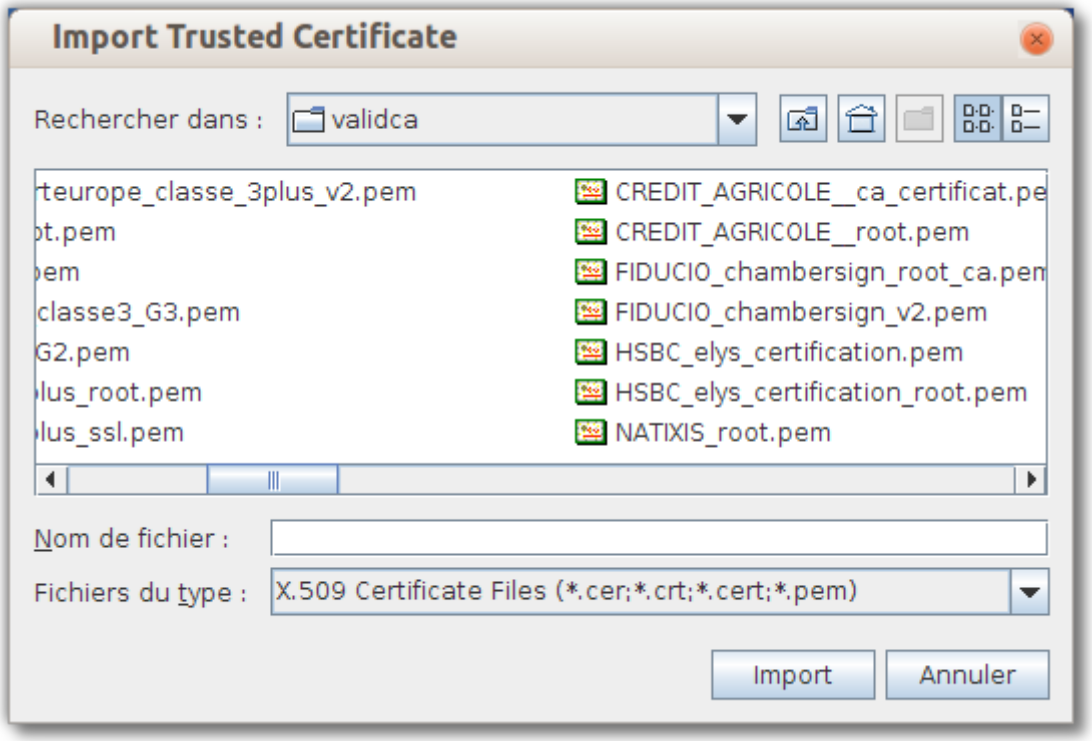
Après avoir dézippé l'applet, on a accès au keystore « ac-truststore.jks » :



Il contient les AC de confiance couramment reconnues par l'applet, l'ajout d'une nouvelle autorité de certification est réalisée avec le bouton (1) :



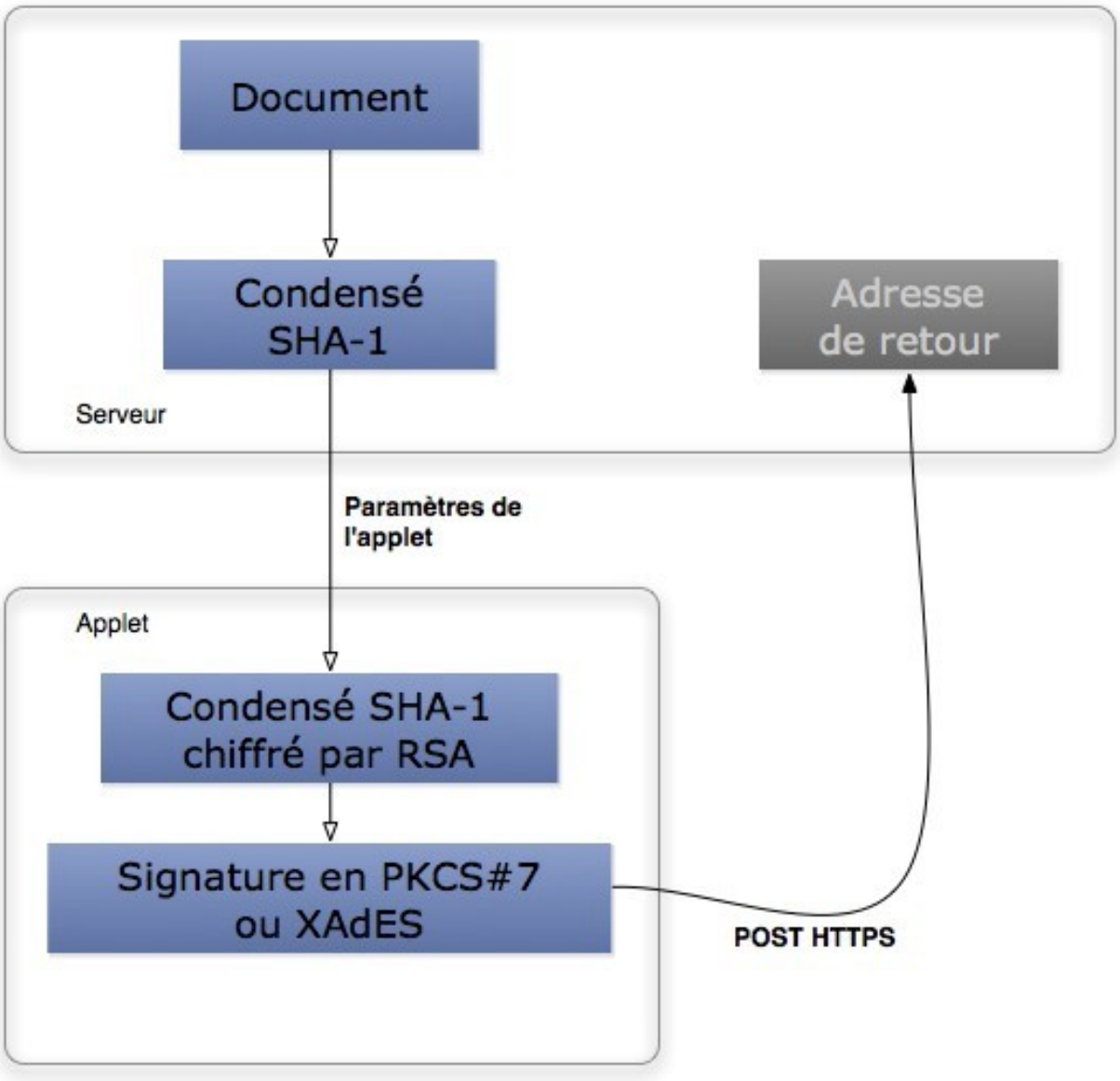
Sélectionner le fichier souhaité :



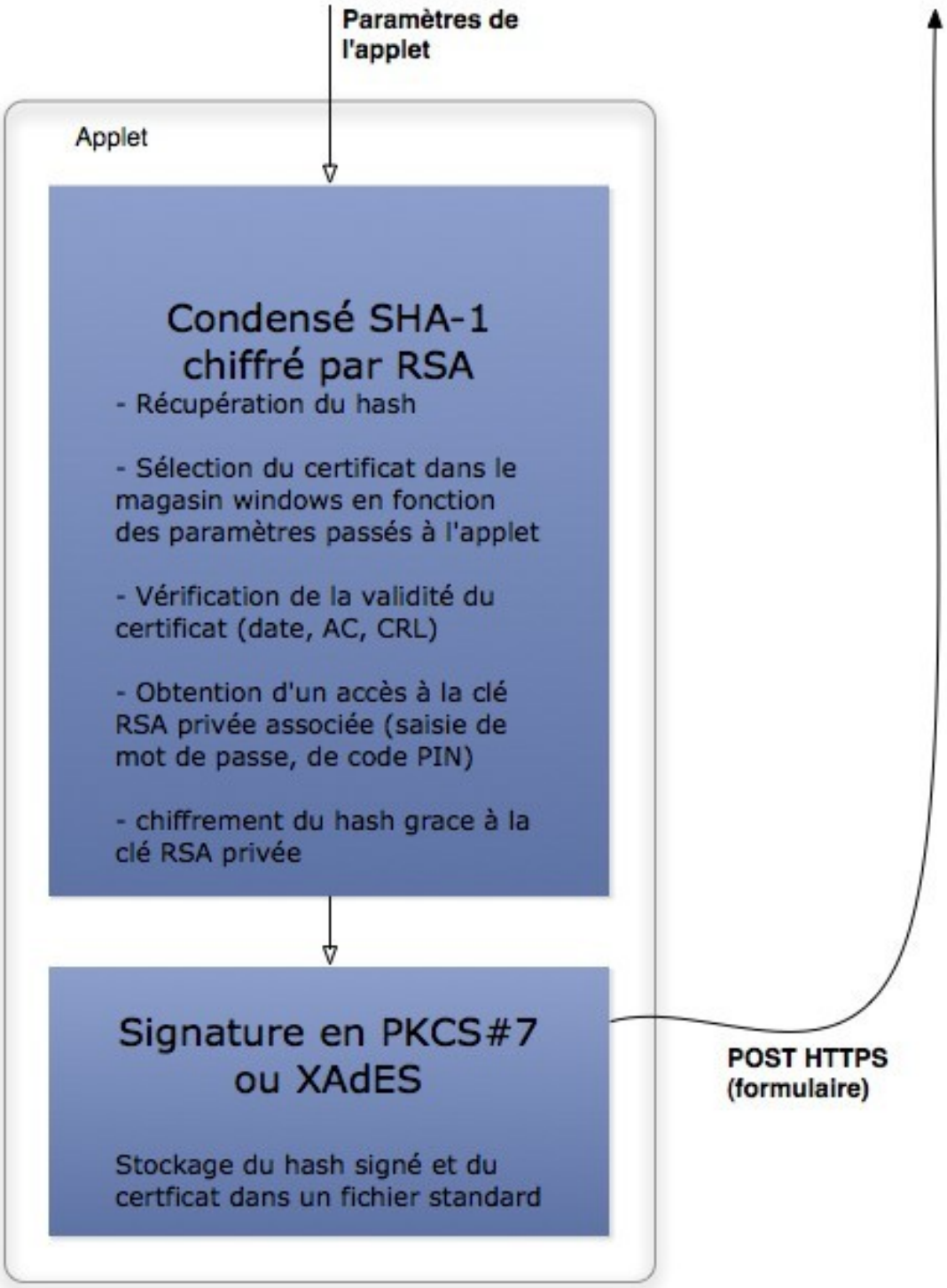
Il est importé dans la liste des AC reconnues, reste à enregistrer le keystore, re-zipper l'applet au format JAR, et surtout **signer l'applet** avec un certificat dédié à la signature de code JAVA.

2.5. Fonctionnement de l'applet de signature

2.5.1. Schéma simplifié



2.5.2. Zoom sur l'applet



2.6. Retour visuel de l'applet

Pour faciliter le travail de l'intégrateur web de l'applet, les retours visuels de l'applet sont directement gérés par celle-ci et non renvoyés à la page appelante.

Ainsi les éventuels messages d'erreurs ("certificat introuvable", "certificat invalide"...) seront affichés par l'applet.

Un code de retour pourra néanmoins être renvoyé au serveur si besoin.

2.7. Usage de l'applet

Cette applet étant mono-fonctionnelle (signature), elle ne possède pas de « méthode publique » d'appel. Un simple appel à l'applet en déclenche le fonctionnement.

Exemple d'appel (mode 'http'):

```
<applet
    codebase = "/signatureApplet"
    code = "org/adullact/parapheur/applets/splittedsign/Main.class"
    archive = "SplittedSignatureApplet.jar,
          lib/bcmail-jdk16-145.jar, lib/bcprov-jdk16-145.jar, lib/xom-1.1.jar"
    name = "appletsignature"
    width = "500"
    height = "257" >

  <param name="hash_count" value="3" />
  <param name="iddoc_1" value="cert113600080829" />
  <param name="iddoc_2" value="docC" />
  <param name="iddoc_3" value="PRESTO_Guide_1_FR163825080902" />
  <param name="hash_1" value="73f227f21065058733cf719533e860d66f04f7b7" />
  <param name="hash_2" value="4ea77484f3a1c7dde4c0cca2f5c40953388f19f5" />
  <param name="hash_3" value="5139ceb0b9f3cf245394f117b28d10bd17c4f0fe" />
  <param name="format_1" value="CMS" />
  <param name="format_2" value="CMS" />
  <param name="format_3" value="CMS" />

  <param name="url_send_content" value="https://www.serveur.local/service" />
  <param name="id_user" value="id=4" />
  <param name="certificat_cn" value="Stephane Vast" />
  <param name="certificat_serial" value="00BA45240B21E7DD57" />
  <param name="certificat_issuer_cnl" value="CA" />

  <param name="return_mode" value="http" />
</applet>
```

2.8. Explication des paramètres (API)

Paramètre	Description
hash_count	Nombre de condensats « hash » à chiffrer.
hash_n	Condensat de chaque document à signer (SHA-1 hexa)
iddoc_n	Nom du document à signer (sera retourné avec la signature correspondante), optionnel.
url_send_content	URL vers laquelle le résultat des signatures doit être posté, à utiliser si *return_mode='http'*.
id_user	Identifiant du signataire, à utiliser si *return_mode='http'*.
certificat_cn	Champ CN du certificat du signataire.
certificat_serial	Numéro de série du certificat.
certificat_issuer_cnl	CN du certificat de l'AC émettrice (garantit l'unicité du numéro de série)
format_n	- signature détachée: '**CMS**' (PKCS#7), '**cms-allin1**', '**XADES**' - signature enveloppée: '**XADES-env**'='**PESv2**',... (voir ci-dessous)
tsa_token_n	Jeton d'horodatage (optionnel), pour les formats cms et cms-allin1
multisignature	- **'cosign'** : co-signature (mode par défaut pour les signatures) - '*contresign*' : contre-signature (non implémenté)
return_mode	- '**http**': publication de(s) signature(s) sur url_send_content - '**form**': utilisation en mode formulaire (voir ci-dessous)
display_cancel	'**true**' / '**false**': présence d'un bouton Annuler sur l'applet (par défaut à 'false'). Si ce paramètre est positionné à 'true', l'activation du bouton Annuler ordonne la fermeture de la fenêtre parente.

Cas particulier: Utilisation avec *return_mode="form"*.

Au cas où l'applet doive fournir la signature dans un champ de formulaire Web, le clic sur le bouton [Signer] crée la signature dans l'applet, puis appelle dans la page Web parente (afin que le formulaire vienne chercher la signature) l'URL: "javascript:injectSignature();".

La page Web (qui embarque l'applet) doit donc implémenter une fonction JavaScript *injectSignature()*, qui ira récupérer la ou les signature(s) auprès de l'applet. Par exemple:

```
function injectSignature()
{
  var signature = null;
  try {
    signature = document.applets[0].returnSignature("hash_1");
  } catch (e) {
    alert(e);
  }
  if (signature) {
    document.forms[0].elements["signature"].value = signature;
    document.forms[0].elements["finish-button"].click();
  }
    else {   return false;   }
}
```

Ou l'identifiant du document s'il existe...

2.9. Formats de signature

Le paramètre '**format_n**' détermine le format de la signature à produire par condensat fourni. Les formats possibles sont les suivants :

format_n	Description
CMS	Format CMS / PKCS#7 détaché standard. Convient pour la plupart des signatures de documents, notamment pour la signature des ACTES Administratifs soumis au contrôle de légalité.
CMS-Allin1	*[Déconseillé]* Format CMS / PKCS#7 « All in One »: mode particulier de co-signature, pour lequel toutes les signatures sont agrégées dans un seul et unique fichier PKCS#7.
PESv2 ou **XADES-env**	Format XAdES enveloppé, utilisé pour la signature de documents financiers ou comptables XML (programme PESv2 de la DGFIP), nécessite la présence de l'attribut Id pour fonctionner.
XADES-env-xpath	Idem que ci-dessus, mais ne requiert pas la présence de l'attribut Id. Ce format est utilisé pour signer les bordereaux dématérialisés de renonciation anticipée pour les DIA.
XADES	Format XAdES détaché.
XADES-T-env	Format de signature XAdES-T enveloppée, qui inclut un jeton d'horodatage conforme RFC3161.
XADES-C-env	[non implémenté] voir les différentes formes possibles
XADES-A-env	[non implémenté] Intéressant pour l'archivage

Cas particulier: co-signature au format PKCS#7 / CMS détachée.

Dans les bonnes pratiques, un document co-signé au format PKCS#7 détaché fait suivre avec lui autant de fichiers 'p7s' que de signatures; celles-ci pouvant être rassemblées en un seul fichier ZIP. Il convient de continuer à utiliser format_n = CMS .

Cependant, la norme CMS/PKCS#7 prévoit également qu'un fichier PKCS#7 puisse contenir plusieurs « co-signatures ». Pour ce faire, l'applet va avoir besoin du conteneur contenant les signatures précédentes, afin d'y ajouter la nouvelle. A noter que ce cas d'utilisation (en exploitant le paramètre format_n = CMS-allin1) est très marginal, et déconseillé.

L'applet a besoin d'un paramètre supplémentaire nommé « **p7s_n** », où 'n' est le numéro du document sur lequel doit porter la nouvelle co-signature.

Paramètre	Description
format_n	'**cms-allin1**' , constante dédiée à ce mode particulier
hash_n	Condensé de chaque document à signer (SHA-1 hexa).
p7s_n	Fichier PKCS#7, encodé base64, ou 'null' si pas de conteneur.

Cas particulier: Signature d'un fichier XML *PESv2* (PES-Aller)

Les fichiers PESv2 peuvent être signés avec l'applet, moyennant quelques aménagements. En effet, la signature doit être enveloppée (incluse dans le fichier PESv2), et sa génération demande bien plus d'informations que le simple condensé SHA-1 du document.

Afin de garder le bénéfice du mode fragmenté de la signature (économie de bande passante), le programme serveur devra mettre en œuvre des transformations du flux XML selon les spécifications en vigueur (voir « Système d'échange des données du PES », diffusé sous le nom de fichier 070415_H1_3_ET_DOSTEC_SystemeEchangesDonneesPES_V2.doc, §4.2.2 « bloc signature électronique »):

- Sélection des données de l'objet à signer selon l'algorithme standard de création d'une signature enveloppée: http://www.w3.org/2001/xmldsig#enveloped-signature
- Mise sous forme canonique des données à signer du flux XML PESv2, selon l'algorithme suivant: http://www.w3.org/2001/xml-exc14n#
- Calcul d'une empreinte SHA-1 sur ce bloc, à mettre dans le paramètre *hash_n*.

En outre, un certain nombre de données sont à extraire du PES_Aller pour renseigner les paramètres à passer à l'applet:

Paramètre	**Description**
hash_n	Condensé du bloc PES à signer (SHA-1 hexa), calculé selon méthode ci-dessus.
pesid_n	Attribut Id de l'élément pes:**PES_Aller**.
pesencoding_n	Nature de l'encodage de la signature XAdES produite (ISO8859-1,...)
nombresignatures_n	Pour les cas de multiples signatures sur le même fichier, indique le nombre de signatures déjà présentes pour l'élément à co-signer.
pespolicyid_n	URN de la politique de signature utilisée, sous la forme OIDAsURN, ira dans l'élément xad:Identifier. (provient du certificat) Exemple: *urn:oid:1.2.250.1.5.3.1.1.10*
pespolicydesc_n	Description textuelle de la politique de signature, ira dans l'élément xad:Description. (provient du certificat)
pespolicyhash_n	Empreinte SHA-1 de la politique de signature, qui ira dans l'élément xad:SigPolicyHash/xad:DigestValue. (provient du certificat)
pesspuri_n	URL de publication de la politique de signature, ira dans l'élément xad:SPURI. (provient du certificat)
pescity_n	Ville de signature, élément xad:SignatureProductionPlace/xad:City.
pespostalcode_n	Code postal de la ville de signature, ira dans l'élément xad:SignatureProductionPlace/xad:PostalCode.
pescountryname_n	Pays, élément xad:SignatureProductionPlace/xad:CountryName.
pesclaimedrole_n	Rôle du signataire, élément xad:ClaimedRole.

3. Applet de vérification de signature

3.1. Description

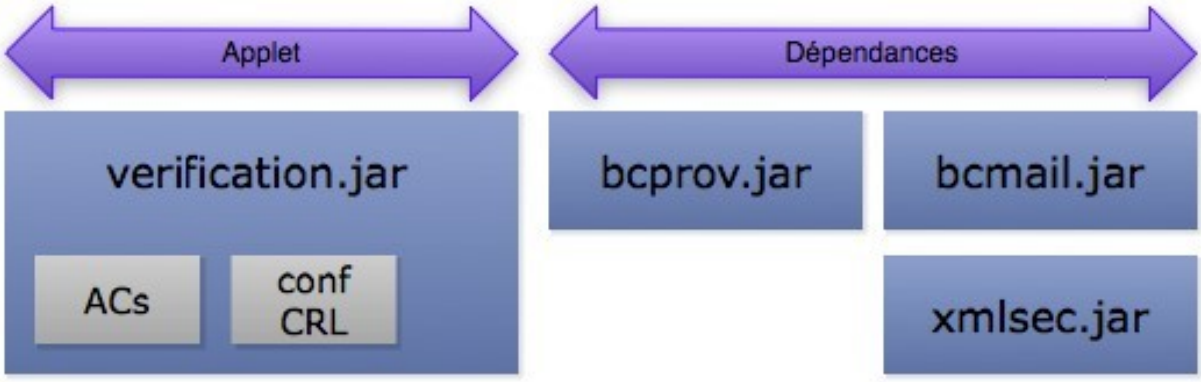
L'applet de vérification de signature a les fonctionnalités suivantes :

- vérification d'une signature détachée au format PKCS#7 ;
- vérification d'une signature détachée au format XAdES ;
- affichage des informations sur le certificat de signature.

Son fonctionnement est le suivant :

L'applet est lancée avec l'adresse URL d'une signature et son format en paramètres. Elle invite l'utilisateur à sélectionner le fichier local et lui permet de vérifier sa signature. Un retour visuel est présenté à l'utilisateur.

Packaging de l'applet de vérification



3.2. Usage de l'applet de vérification

Cette applet étant mono-fonctionnelle (vérification) elle ne possède pas de méthode publique. Un simple appel à l'applet en déclenche le fonctionnement.

Exemple d'appel :

```
<applet
     codebase = "/VerificationApplet"
     code = "org/adullact/VerificationApplet/Main.class"
     archive = "verification.jar,
           lib/bcmail-jdk16-138.jar, lib/bcprov-jdk16-138.jar, lib/xom-1.1.jar"
     name = "appletverification"
     width = "500"
     height = "450" >

   <param name="url_get_signature"
      value="http://www.monserveur.net/get_signature.php?id_doc=433387132" />
   <param name="format" value="CMS" />
</applet>
```

3.3. Explication des paramètres (API)

Paramètre	Description
url_get_signature	URL à laquelle on obtient la signature à vérifier.
format	Format de signature: 'CMS' ou 'XADES'

Il faut noter que l'applet va, après son lancement, afficher des informations sur le contenu de la signature (signataire, etc...), et demander à l'utilisateur de choisir le document dont il faudra vérifier la signature.