Local Signature Computation

Frank Cornelis
frank.cornelis@fedict.be
December 15, 2011
Version 1.0

1 Motivation

The signing protocols of the OASIS DSS specification mainly focus on centralized key management systems. Such architecture makes sense for situations where the connecting clients don't own tokens with signing capabilities themselves. However, large-scale signing token deployments (like it is the case for national eID cards) reduces the need for a centralized key management system. In such scenarios it is still interesting to keep a centralized system in place for several reasons:

- Despite the fact that each connecting client owns a token with signing capability, he/she might not have the appropriate software installed on the system for the creation of electronic signatures. It might be easier to maintain a lightweight applet solution instead of a full blown token middleware solution that has to be installed on every participating client's system. The diversity among the client platforms is also easier to maintain from a centralized platform instead of by distributing token middleware to all participating clients. Furthermore managing the configuration of the signature policy to be used for creation and validation of a signature within a certain business context might be easier using a centralized platform.
- When transforming a paper-world business work flow to a digital equivalent that includes the creation and/or validation of signatures, it might be interesting to offer the sub-process of creating/validating an electronic signature as an online service. Given the technicality of signature creation and validation, a clean separation of concerns in the service architecture is desired.

So the role of the centralized system shifts from key management to providing a platform that manages the technicalities of singing documents using client tokens.

2 Sample Application

Illustration 1: eID DSS Signature Pipeline shows the design of a digital signature service that uses the Belgian eID card as client signing token.

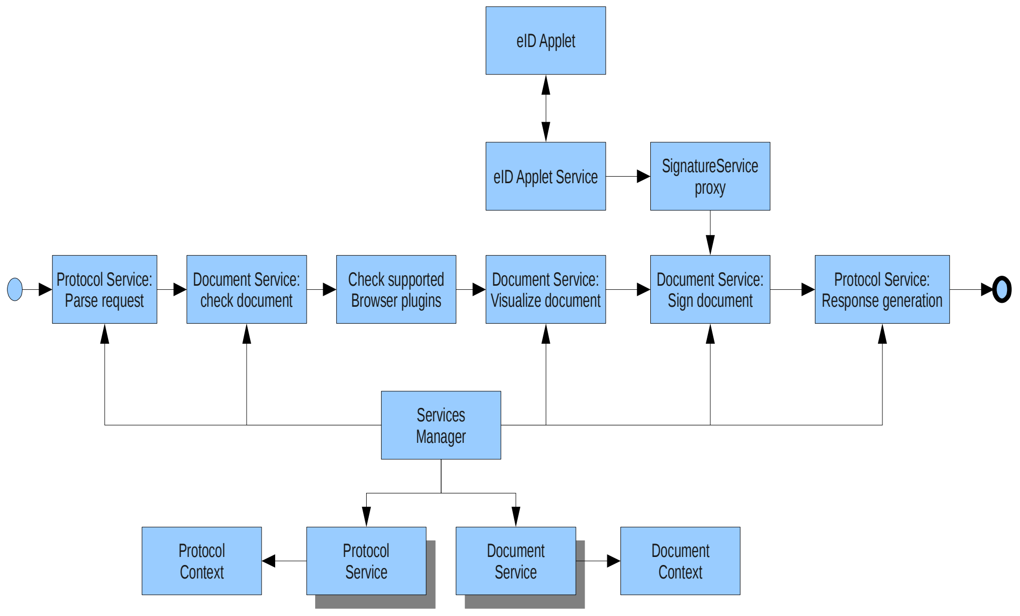
eID Applet
eID Applet Service
SignatureService proxy
Protocol Service: Parse request
Document Service: check document
Check supported Browser plugins
Document Service: Visualize document
Document Service: Sign document
Protocol Service: Response generation
Services Manager
Protocol Context
Protocol Service
Document Service
Document Context

Illustration 1: eID DSS Signature Pipeline

An end-user enters the DSS signature pipeline via some protocol. First of all the appropriate protocol service parses the request. At this step the mime type of the incoming document is determined. Via the mime type the appropriate document service can be selected. The document service will first check the incoming document (syntax, ...). Next the web browser capabilities are being queried in order for the document service to be able to correctly visualize the received document. After the user's consent the document service will orchestrate the document signing process using a web browser Java applet component. Finally the signed document is returned via the protocol service that also handled the incoming protocol request.

The advantage of such a generic signature pipeline architecture is that one can easily add new supported document formats by providing a new document service implementation. Because the protocol handling is also isolated in protocol services, one can also easily add new DSS protocols to the platform. Another advantage of such a signature pipeline is that every Relying Party that uses the platform is guaranteed that the user followed a certain signature ceremony and is fully aware of the content of the signed document. This guarantee can be interesting from a legal point of view.

3 Sample scenario

A sample protocol flow is shown in Illustration 2: Sequence diagram of a simple protocol flow.

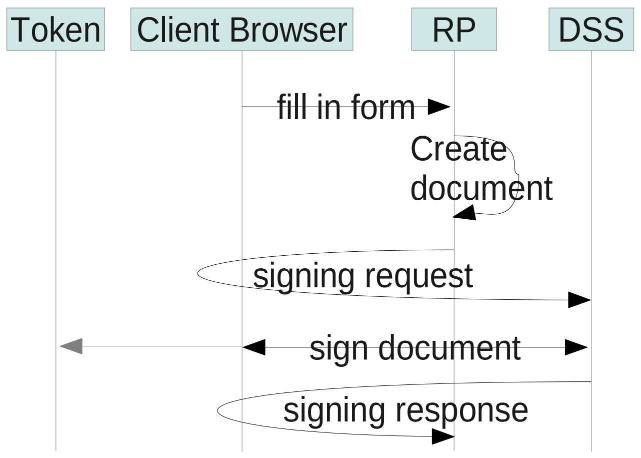

Illustration 2: Sequence diagram of a simple protocol flow

Here the client navigates via a web browser to the web application of the relying party. As part of the business work flow, the client fills in a web form. The relying party's web application converts the received form data into a document that needs to be signed by the client. Now the relying party's web application redirects the client web browser to the DSS web application. The DSS web application takes care of the signing ceremony using Java applet technology to connect to the client's token. Finally the DSS web application redirects the client's web browser back to the relying party. The relying party can now further process the signed document as part of the implemented business work flow.

In such scenarios it is difficult to use the existing OASIS DSS protocol messages as is because the OASIS DSS protocol does not provide the security mechanisms required to secure the communication between relying parties and the DSS in the context of web browsers. Various MITM attacks are possible at different points during the signature ceremony. Similar to the OASIS SAML Browser POST profile we need to define additional wrapper messages to be able to guarantee secure transportation of the DSS requests and responses via web browsers.

A disadvantage of the simple protocol shown is that the entire document is being transferred between relying party and DSS (and back) using the client's web browser. Given the upload limitation of most client's internet connection, this might result in a bad end-user experience when trying to sign a document. So additionally we should define some form of artifact binding. Here the relying party sends the to be signed document via a SOAP DSS web service to the DSS. The DSS stores the document in some temporary document repository. The relying party receives back a document identifier which it passes as parameter when redirecting the client's web browser towards the DSS. At the end of the protocol flow, the relying party can fetch the signed document from the DSS web service using the document identifier.