Local Signing & Encryption with Centralised Decryption

This Ascertia Solution Sheet describes how local signing and encryption together with central decryption services can be implemented using ADSS Server. The main market this feature set is any type of sealed information or sealed tender project where good central controls are required to prevent accidental or unauthorised release of information.

Background

Business users are expected to interact with an online web application (e.g. e-tendering application) as depicted below:

Users ←→ Web Application ←→ ADSS Server

Users wanting to upload documents need to interact with the business application and then sign, timestamp and encrypt the package of documents. Later, at an appropriate time, the business application can request decryption of these packages by making calls to the ADSS Decryption Service. Decryption is controlled by standard ADSS Server client authentication. The timestamp embedded in the signature is used to confirm the time of signature using a central trustworthy time source and the two items provided indelible evidence as to the date/time of submission.

Data Tender Encryption Key Creation and Certification

The business application can make a call to the ADSS Certification Service to generate an encryption key pair and a certificate using the ADSS CA Service. It is expected that applications request a separate encryption key for each project. As an alternative ADSS Server operations staff can use the admin interface to create a new decryption profile and a new non-exportable decryption key and certificate probably using the high security dual control option that ADSS Server offers.

For security reasons ADSS Server retains all private key data and only returns a public key certificate to the web application. For higher security an HSM can be used to protect the private key. Multiple decryption profiles can be supported.

Client-side Functionality

As part of the document submission users need to sign & encrypt document packages before they are uploaded to the application. ADSS Server and ADSS Go>Sign Client products provide this functionality:

- The required functionality is implemented within ADSS Go>Sign Client running in the user's web-browser.
- Only one data object can be signed and uploaded, if multiple documents are used then must first be zipped into one package by the user (using any 3rd party zip utility). The original documents may be of any format. The data size can be large, e.g. 20MB.
- Users will sign the package using locally held keys on a smartcard or USB token or even soft certs or roamed credentials. The signature format is expected to be XAdES-T, using local hashing and signing. An enveloping signature is used with the data package wrapped inside.
- Once signed, the ADSS Go>Sign Client requests a timestamp from the web application to create the XAdES-T signature object. The web application makes a timestamp request to ADSS Server. This can either be satisfied using the TSA Service module or it

can be routed to an external RFC 3161 compliant TSA. Access to an external TSA can be authenticated using SSL client certificates if required:

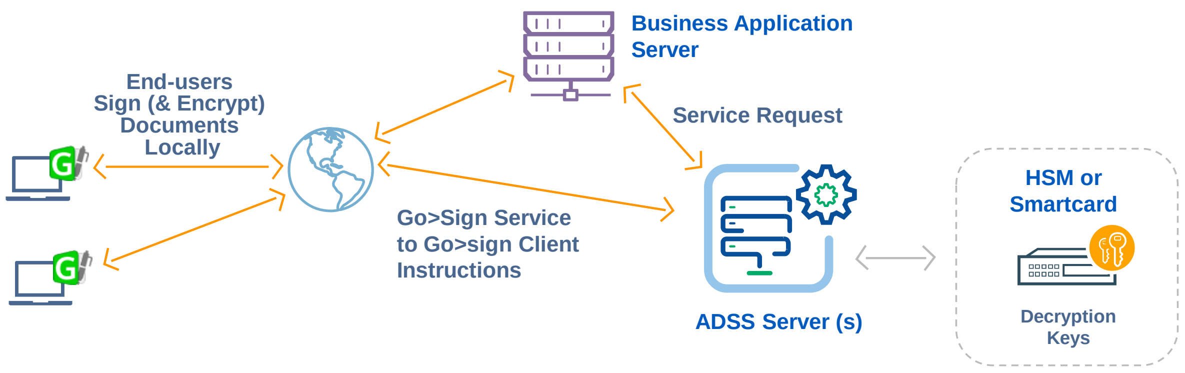
Go>Sign ←→ Web App ←→ ADSS Server ←(SSL-Client-Auth)→ External TSA Provider

- The timestamp token is obtained and returned to Go>Sign by the web application. Go>Sign embeds this within the signature to create the XADES-T signed object.
- Go>Sign now encrypts the signed object using standard XML Encryption techniques. This uses an approach similar to S/MIME, in that a random symmetric key encrypts the content, and the symmetric key is encrypted with the receiver's public key). The encryption certificate created earlier is passed to Go>Sign by the web application. The symmetric encryption algorithm used is currently AES 128 bit since AES 256 is not supported within older Windows crypto-service providers.
- From the user's perspective they sign in one operation. It is recommended that the Go>Sign service profile is set to filter the list of acceptable signing certificates to just one, for example an EU qualified certificate. The details of the symmetric encryption algorithm, encryption certificates or complexities such as timestamping, are all hidden from the user and controlled by the ADSS Go>Sign Service Profile.

Server-side Functionality

The web application receives the encrypted package from the user and saves it for later decryption. At a future date/time the web application can request that the ADSS Decryption Service decrypt the package. This process is described below:

- The web application creates a decryption web services request message that contains the encrypted data package. ADSS Server authenticates the web application using OriginatorID, SSL client certificate or request signatures – the same way it does for other service requests.
- ADSS Decryption Service will perform the usual authentication and authorisation checking process, e.g.:
  - Check the request is correctly formatted
  - Check business application is registered, authenticated and approved to use this decryption profile (and key)
- If all the above checks succeed the ADSS Decryption Service will decrypt the encrypted data package. The decrypted data package is provided back to the client application. The transaction is securely logged within the Decryption Service for later audit as may be required.
- The client application now verifies the signature on the data package by making a separate signature verification request to the ADSS Verification Service.


Business Application Server
End-users Sign (& Encrypt) Documents Locally
Service Request
Go>Sign Service to Go>sign Client Instructions
ADSS Server (s)
HSM or Smartcard
Decryption Keys

Server-side Authorised Decryption Functionality

A future version of ADSS Server will allow authorisation profiles to be used. These are already available for digital signature creation requests. The process flow is described below and this highlights how authorised decryption works:

- The web application creates a decryption web services request message that contains the encrypted data package. ADSS Server authenticates the web application using OriginatorID, SSL client certificate or request signatures – the same way it does for other service requests.
- ADSS Decryption Service will perform the usual authentication and authorisation checking process, e.g.:
  - Check the request is correctly formatted
  - Check client app is registered, authenticated and approved for decryption profile
  - Check any optional embargo date/time has been reached
- If all the above checks succeed then the ADSS Decryption Service decrypts the encrypted data package. The decrypted data package is provided back to the client application.
- The web-application should now verify the digital signature on the data package by making a separate signature verification request to the ADSS Verification Service.

Solution Option:

Ascertia has a roadmap item to offer M of N authorisation to the decryption service. Authorisation profiles are already understood for corporate signatures and so this approach could easily be added to the ADSS Decryption service. What this would mean is:

- When decrypting the data the web application requests one of more end-user authorisers to approve the decryption request (based on the M of N authorisation profile for this decryption request). The web application would know the requirements of the authorisation profile and also know who the relevant authorisers are – this is a business logic decision. ADSS Server will be used to enforce this.
- To create the decryption request message, the web application:
  - Creates a hash of the encrypted data package
  - Enables each authoriser to sign this hash using their personal locally held (or roamed) signing keys
  - Places each authoriser's signature within the Decryption Service request message

  Note: as the decryption packages are encrypted the authorisers don't really know what they are approving by looking at just the package or its hash value. To provide better security, authorisers should be asked to sign not only the decryption package but also an identifier for the package (e.g. the tender project ID). Therefore the web application should concatenate the "package ID info" or other metadata and the encrypted package itself, then hash this and ask authoriser to sign it.

Solution Option:

It would also be possible to have an embargo date/time added to a decryption service profile. If this feature was added then a decryption request would be checked against the profile embargo date/time and obviously requests that pre-date this are rejected (and securely logged) thus tightening security.

For further details see the Ascertia website www.ascertia.com or email info@ascertia.com