# Table of Contents

Introduction ................................................................. 1
  Purpose of the document ........................................... 1
  Scope of the document ............................................... 1
  Abbreviations and Acronyms ........................................ 1
  References .................................................................. 4
  Useful links .................................................................. 5
General framework structure ........................................... 5
  DSS Utils ................................................................. 8
  DSS CRL Parser ........................................................ 8
  DSS PAdES .............................................................. 9
Available demonstrations ............................................... 9
Signature's profile simplification .................................... 9
The XML Signature (XAdES) ............................................. 10
  XAdES Profiles .......................................................... 10
  Various settings .......................................................... 27
  Multiple signatures ..................................................... 27
  The XML Signature Extension (XAdES) ......................... 27
  XAdES-BASELINE-T ................................................... 28
  XAdES-BASELINE-LT and -LTA .................................... 29
  XAdES and specific schema version ............................... 29
The signature validation .................................................. 30
  Validation Process ....................................................... 30
  EU Trusted Lists of Certification Service Providers ............ 34
  Validation Result Materials .......................................... 36
  Customised Validation Policy ....................................... 50
CAAdES signature (CMS) .................................................. 58
PAdES signature (PDF) ................................................... 59
  PAdES Visible Signature .............................................. 62
ASiC signature (containers) ............................................. 64
Available implementations of DSSDocument ..................... 67
Management of signature tokens .................................... 67
  PKCS#11 ................................................................. 67
  PKCS#12 ................................................................. 68
  MS CAPI ................................................................. 69
  Other Implementations ................................................. 69
Management of certificates sources ................................. 69
Management of CRL and OCSP sources ............................ 70
  Other implementations of CRL and OCSP Sources ............. 71
CertificateVerifier configuration ......................................................... 72
TSP Sources .................................................................................. 74
  Time-stamp policy ........................................................................ 74
  Composite TSP Source ................................................................. 74
Supported algorithms ................................................................. 75
Multi-threading ............................................................................ 76
  Resource sharing ......................................................................... 76
  Caching ....................................................................................... 76
Additional features ....................................................................... 77
  Certificate validation ................................................................. 77
  Extract the signed data from a signature ...................................... 78
REST Services ............................................................................... 78
  REST signature service ............................................................ 79
  REST server signature service .................................................. 89
  REST validation service ............................................................. 92
Introduction

Purpose of the document

This document describes some examples of how to develop in Java using the DSS framework. The aim is to show to the developers, in a progressive manner, the different uses of the framework. It will familiarise them with the code step by step.

Scope of the document

This document provides examples of code which allow easy handling of digital signatures. The examples are consistent with the Release 5.4.3 of DSS framework which can be downloaded via https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/DSS+-+releases

Three main features can be distinguished within the framework:

- The digital signature;
- The extension of a digital signature and;
- The validation of a digital signature.

On a more detailed manner the following concepts and features are addressed in this document:

- Formats of the signed documents: XML, PDF, DOC, TXT, ZIP...;
- Packaging structures: enveloping, enveloped, detached and internally-detached;
- Forms of digital signatures: XAdES, CAdES, PAdES and ASiC-S/ASiC-E;
- Profiles associated to each form of the digital signature;
- Trust management;
- Revocation data handling (OCSP and CRL sources);
- Certificate chain building;
- Signature validation and validation policy;
- Validation of the signing certificate.

This is not an exhaustive list of all the possibilities offered by the framework and the proposed examples cover only the most useful features. However, to discover every detail of the operational principles of the framework, the JavaDoc is available within the source code.

Please note that the DSS framework is still under maintenance and new features will be released in the future.

Abbreviations and Acronyms

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdES</td>
<td>Advanced Electronic Signature</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Full Form</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ASiC</td>
<td>Associated Signature Containers</td>
</tr>
<tr>
<td>BB</td>
<td>Building Block (CEF)</td>
</tr>
<tr>
<td>CA</td>
<td>Certificate authority</td>
</tr>
<tr>
<td>CAAdES</td>
<td>CMS Advanced Electronic Signatures</td>
</tr>
<tr>
<td>CD</td>
<td>Commission Decision</td>
</tr>
<tr>
<td>CEF</td>
<td>Connecting Europe Facility</td>
</tr>
<tr>
<td>CMS</td>
<td>Cryptographic Message Syntax</td>
</tr>
<tr>
<td>CRL</td>
<td>Certificate Revocation List</td>
</tr>
<tr>
<td>CSP</td>
<td>Core Service Platform (CEF)</td>
</tr>
<tr>
<td>CSP</td>
<td>Cryptographic Service Provider</td>
</tr>
<tr>
<td>DER</td>
<td>Distinguished Encoding Rules</td>
</tr>
<tr>
<td>DSA</td>
<td>Digital Signature Algorithm - an algorithm for public-key cryptography</td>
</tr>
<tr>
<td>DSI</td>
<td>Digital Service Infrastructure (CEF)</td>
</tr>
<tr>
<td>DSS</td>
<td>Digital Signature Service</td>
</tr>
<tr>
<td>EC</td>
<td>European Commission</td>
</tr>
<tr>
<td>eID</td>
<td>Electronic Identity Card</td>
</tr>
<tr>
<td>ESI</td>
<td>Electronic Signatures and Infrastructures</td>
</tr>
<tr>
<td>ETSI</td>
<td>European Telecommunications Standards Institute</td>
</tr>
<tr>
<td>EUPL</td>
<td>European Union Public License</td>
</tr>
<tr>
<td>FSF</td>
<td>Free Software Foundation</td>
</tr>
<tr>
<td>GS</td>
<td>Generic Service (CEF)</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HSM</td>
<td>Hardware Security Modules</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>I18N</td>
<td>Internationalisation</td>
</tr>
<tr>
<td>Java EE</td>
<td>Java Enterprise Edition</td>
</tr>
<tr>
<td>JavaDoc</td>
<td>JavaDoc is developed by Sun Microsystems to create API documentation in HTML format from the comments in the source code. JavaDoc is an industrial standard for documenting Java classes.</td>
</tr>
<tr>
<td>JAXB</td>
<td>Java Architecture for XML Binding</td>
</tr>
<tr>
<td>JCA</td>
<td>Java Cryptographic Architecture</td>
</tr>
<tr>
<td>JCE</td>
<td>Java Cryptography Extension</td>
</tr>
<tr>
<td>JDBC</td>
<td>Java DataBase Connectivity</td>
</tr>
<tr>
<td>LGPL</td>
<td>Lesser General Public License</td>
</tr>
<tr>
<td>Acronym</td>
<td>Full Form</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>LOTL</td>
<td>List of Trusted List or List of the Lists</td>
</tr>
<tr>
<td>LSP</td>
<td>Large Scale Pilot</td>
</tr>
<tr>
<td>MIT</td>
<td>Massachusetts Institute of Technology</td>
</tr>
<tr>
<td>MOCCA</td>
<td>Austrian Modular Open Citizen Card Architecture; implemented in Java</td>
</tr>
<tr>
<td>MS / EUMS</td>
<td>Member State</td>
</tr>
<tr>
<td>MS CAPI</td>
<td>Microsoft Cryptographic Application Programming Interface</td>
</tr>
<tr>
<td>OCF</td>
<td>OEBPS Container Format</td>
</tr>
<tr>
<td>OCSP</td>
<td>Online Certificate Status Protocol</td>
</tr>
<tr>
<td>ODF</td>
<td>Open Document Format</td>
</tr>
<tr>
<td>ODT</td>
<td>Open Document Text</td>
</tr>
<tr>
<td>OEBPS</td>
<td>Open eBook Publication Structure</td>
</tr>
<tr>
<td>OID</td>
<td>Object Identifier</td>
</tr>
<tr>
<td>OOWXML</td>
<td>Office Open XML</td>
</tr>
<tr>
<td>OSI</td>
<td>Open Source Initiative</td>
</tr>
<tr>
<td>OSS</td>
<td>Open Source Software</td>
</tr>
<tr>
<td>PAdES</td>
<td>PDF Advanced Electronic Signatures</td>
</tr>
<tr>
<td>PC/SC</td>
<td>Personal computer/Smart Card</td>
</tr>
<tr>
<td>PDF</td>
<td>Portable Document Format</td>
</tr>
<tr>
<td>PDFBox</td>
<td>Apache PDFBox - A Java PDF Library: <a href="http://pdfbox.apache.org/">http://pdfbox.apache.org/</a></td>
</tr>
<tr>
<td>PKCS</td>
<td>Public Key Cryptographic Standards</td>
</tr>
<tr>
<td>PKCS#12</td>
<td>It defines a file format commonly used to store X.509 private key accompanying public key certificates, protected by symmetrical password</td>
</tr>
<tr>
<td>PKIX</td>
<td>Internet X.509 Public Key Infrastructure</td>
</tr>
<tr>
<td>RSA</td>
<td>Rivest Shamir Adleman - an algorithm for public-key cryptography</td>
</tr>
<tr>
<td>SCA</td>
<td>Signature Creation Application</td>
</tr>
<tr>
<td>SME</td>
<td>Signature Creation Device</td>
</tr>
<tr>
<td>SMO</td>
<td>Stakeholder Management Office (CEF)</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SSCD</td>
<td>Secure Signature-Creation Device</td>
</tr>
<tr>
<td>SVA</td>
<td>Signature Validation Application</td>
</tr>
<tr>
<td>TL</td>
<td>Trusted List</td>
</tr>
<tr>
<td>TLManager</td>
<td>Application for managing trusted lists.</td>
</tr>
<tr>
<td>TSA</td>
<td>Time Stamping Authority</td>
</tr>
</tbody>
</table>
References

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Title</th>
<th>Reference</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>R01</td>
<td>ESI - XAdES digital signatures</td>
<td>ETSI EN 319 132 part 1-2</td>
<td>1.0.0</td>
</tr>
<tr>
<td>R02</td>
<td>ESI - CAdES digital signatures</td>
<td>ETSI EN 319 122 part 1-2</td>
<td>1.1.1</td>
</tr>
<tr>
<td>R03</td>
<td>ESI - PAdES digital signatures</td>
<td>ETSI EN 319 142 part 1-2</td>
<td>1.1.1</td>
</tr>
<tr>
<td>R04</td>
<td>ESI - Associated Signature Containers (ASiC)</td>
<td>ETSI EN 319 162 part 1-2</td>
<td>1.1.1</td>
</tr>
<tr>
<td>R05</td>
<td>Document management - Portable document format - Part 1: PDF 1.7</td>
<td>ISO 32000-1</td>
<td>1</td>
</tr>
<tr>
<td>R07</td>
<td>Internet X.509 Public Key Infrastructure - Time-Stamp Protocol (TSP)</td>
<td>RFC 3161</td>
<td></td>
</tr>
<tr>
<td>R08</td>
<td>ESI - Procedures for Creation and Validation of AdES Digital Signatures</td>
<td>ETSI EN 319 102-1</td>
<td>1.1.1</td>
</tr>
</tbody>
</table>
Useful links

- CEF Digital
- TL Browser
- Source code (GitHub)
- Source code (EC Bitbucket)
- Source code demonstrations (EC Bitbucket)
- Report an issue (EC Jira)
- Old Jira

General framework structure

DSS framework is a multi-modules project which can be built with Maven.

You can easily download them with the following Maven repository:

```xml
<repository>
  <id>cefdigital</id>
  <name>cefdigital</name>
  <url>
    https://ec.europa.eu/cefdigital/artifact/content/repositories/esignaturedss/
  </url>
</repository>
```

dss-model
Data model used in almost every modules.

dss-token
Token definitions and implementations for MS CAPI, PKCS#11, PKCS#12.

dss-document
Common module to sign and validate document. This module doesn’t contain any implementation.
**dss-asic-common**
Common code which is shared between dss-asic-xades and dss-asic-cades.

**dss-asic-cades**
Implementation of the ASiC-S and ASiC-E signature, extension and validation based on CAdES signatures.

**dss-asic-xades**
Implementation of the ASiC-S and ASiC-E signature, extension and validation based on XAdES signatures.

**dss-cades**
Implementation of the CAdES signature, extension and validation.

**dss-pades**
Common code which is shared between dss-pades-pdfbox and dss-pades-openpdf.

**dss-pades-pdfbox**
Implementation of the PAdES signature, extension and validation with PDFBox.

**dss-pades-openpdf**
Implementation of the PAdES signature, extension and validation with OpenPDF (fork of iText).

**dss-xades**
Implementation of the XAdES signature, extension and validation.

**dss-spi**
Interfaces, util classes to manipulate ASN1, compute digests,...

**dss-service**
Implementations to communicate with online resources (TSP, CRL, OCSP).

**dss-crl-parser**
API to validate CRLs and retrieve revocation data

**dss-crl-parser-stream**
Implementation of dss-crl-parser which streams the CRL (experimental).

**dss-crl-parser-x509crl**
Implementation of dss-crl-parser which uses the java object X509CRL.

**dss-tsl-validation**
Module which allows to load / parse / validate LOTL and TSLs.

**validation-policy**
Business of the signature’s validation (ETSI EN 319 102).

**dss-rest**
REST webservice to sign (getDataToSign, signDocument methods) and extend a signature.

**dss-rest-client**  
Client for the REST webservice.

**dss-soap**  
SOAP webservice to sign (getDataToSign, signDocument methods) and extend a signature.

**dss-soap-client**  
Client for the SOAP webservice.

**dss-validation-rest**  
REST webservice to validate a signature.

**dss-validation-rest-client**  
Client for the REST webservice.

**dss-validation-soap**  
SOAP webservice to validate a signature.

**dss-validation-soap-client**  
Client for the SOAP webservice.

**dss-remote-services**  
Common code between dss-rest and dss-soap.

**dss-server-signing-common**  
Common code for server signing.

**dss-server-signing-rest**  
REST webservice for server signing.

**dss-server-signing-rest-client**  
REST client for server signing.

**dss-server-signing-soap**  
SOAP webservice for server signing.

**dss-server-signing-soap-client**  
SOAP client for server signing.

**sscd-mocca-adapter**  
Implementation for MOCCA token.

**dss-policy-jaxb**  
JAXB model of the validation policy.

**dss-diagnostic-jaxb**
JAXB model of the diagnostic data.

dss-simple-report-jaxb
JAXB model of the simple report.

dss-detailed-report-jaxb
JAXB model of the detailed report.

dss-reports
  Wrappers to easily manipulate the reports JAXB models (diagnostic-data, simple-report, detailed-report).

dss-tsl-jaxb
JAXB model of the TSL.

dss-utils
  API with utility methods for String, Collection, I/O,...

dss-utils-apache-commons
  Implementation of dss-utils with Apache Commons libraries

dss-utils-google-guava
  Implementation of dss-utils with Google Guava

dss-test
  Mocks and util classes for unit tests.

dss-cookbook
  Samples and documentation of DSS used to generate this documentation.

DSS Utils

The module dss-utils offers an interface with utility methods to operate on String, Collection, I/O,...
DSS framework provides two different implementations with the same behavior:

  - dss-utils-apache-commons : this module uses Apache Commons libraries (commons-lang3, commons-collection4, commons-io and commons-codec);
  - dss-utils-google-guava : this module only requires Google Guava (recommended on Android).

If your integration include dss-utils, you will need to select an implementation.

DSS CRL Parser

DSS contains two ways to parse/validate a CRL and to retrieve revocation data. An alternative to the
X509CRL java object was developed to face memory issues in case of large CRLs. The X509CRL object fully loads the CRL in memory and can cause OutOfMemoryError.

  - dss-crl-parser-x509crl : this module uses the X509CRL java object.
• dss-crl-parser-streams: this module offers an alternative with a CRL streaming (experimental).

If your integration require dss-crl-parser, you will need to choose your implementation.

DSS PAdES

Since the version 5.4, DSS allows to generate/extend/validate PAdES signatures with two different frameworks: PDFBox and OpenPDF (fork of iText). The dss-pades module only contains the common code and requires an underlying implementation:

• dss-pades-pdfbox
• dss-pades-openpdf

DSS permits to override the visible signature generation with these interfaces:

• eu.europa.esig.dss.pdf.IPdfObjFactory
• eu.europa.esig.dss.pdf.visible.SignatureDrawerFactory (selects the SignatureDrawer depending of the SignatureImageParameters content)
• eu.europa.esig.dss.pdf.visible.SignatureDrawer

A new instance of the IPdfObjFactory can be created with its own SignatureDrawerFactory and injected in the PdfObjFactory.setInstance(IPdfObjFactory).

Available demonstrations

With the framework, some demonstrations are provided.

dss-standalone-app
Standalone application which allows to sign a document with different formats and tokens (JavaFX).

dss-standalone-app-package
Packaging module for dss-standalone-app.

dss-demo-webapp
Demonstration web application which presents a part of the DSS possibilities.

dss-demo-bundle
Packaging module for dss-demo-webapp.

⚠️ The demonstrations use a simulated timestamp service (Mock) so that is not recommended for a production usage.

Signature’s profile simplification

The different formats of the digital signature make possible to cover a wide range of real live cases
of use of this technique. Thus we distinguish the following formats: XAdES, CAdES, PAdES and ASIC. To each one of them a specific standard is dedicated. The wide variety of options, settings and versions of the standards makes their interoperability very difficult. This is the main reason for which new standards commonly called “baseline profiles” were published. Their goal is to limit the number of options and variants thereby making possible a better interoperability between different actors.

In general can be said that for each format of the digital signature the number of security levels defined in the new standards has been reduced. Below is a comparative table of old and new levels for each format of the signature:

<table>
<thead>
<tr>
<th>XAdES</th>
<th>CAdES</th>
<th>PAdES</th>
</tr>
</thead>
<tbody>
<tr>
<td>STANDARD</td>
<td>BASELINE</td>
<td>STANDARD</td>
</tr>
<tr>
<td>XAdES-BES</td>
<td>XAdES-B</td>
<td>CAdES-BES</td>
</tr>
<tr>
<td>XAdES-EPES</td>
<td>CAdES-EPES</td>
<td>CAdES-B</td>
</tr>
<tr>
<td>XAdES-T</td>
<td>XAdES-T</td>
<td>CAdES-T</td>
</tr>
<tr>
<td>XAdES-XL</td>
<td>XAdES-LT</td>
<td>CAdES-XL</td>
</tr>
<tr>
<td>XAdES-A</td>
<td>XAdES-LTA</td>
<td>CAdES-A</td>
</tr>
</tbody>
</table>

Note that the new version (v4) of the DSS framework is compatible with the baseline profiles, it is no longer possible to use the standard profiles for signing purpose. The validation of the signature still takes into account the old profiles.

### The XML Signature (XAdES)

The simplest way to address the digital signature passes through the XAdES format. Indeed, it allows to visualize the content of the signature with a simple text editor. Thus it becomes much easier to make the connection between theoretical concepts and their implementation. Before embarking on the use of the DSS framework, it is advisable to read the following documents:

- XAdES Specifications (cf. [R01])

After reading these documents, it is clear that:

- To electronically sign a document, a signing certificate (that proves the signer’s identity) and the access to its associated private key is needed.
- To electronically validate a signed document the signer’s certificate containing the public key is needed. To give a more colourful example: when a digitally signed document is sent to a given person or organization in order to be validated, the certificate with the public key used to create the signature must also be provided.

### XAdES Profiles

The new ETSI standard defines four conformance levels to address the growing need to protect the validity of the signature in time. Henceforth to denote the level of the signature the word “level” will be used. Follows the list of levels defined in the standard:
• **XAdES-BASELINE-B**: Basic Electronic Signature The lowest and simplest version just containing the SignedInfo, SignatureValue, KeyInfo and SignedProperties. This level combines the old -BES and -EPES levels. This form extends the definition of an electronic signature to conform to the identified signature policy.

• **XAdES-BASELINE-T**: Signature with a timestamp A timestamp regarding the time of signing is added to protect against repudiation.

• **XAdES-BASELINE-LT**: Signature with Long Term Data Certificates and revocation data are embedded to allow verification in future even if their original source is not available. This level is equivalent to the old -XL level.

• **XAdES-BASELINE-LTA**: Signature with Long Term Data and Archive timestamp By using periodical timestamping (e.g. each year) compromising is prevented which could be caused by weakening previous signatures during a long-time storage period. This level is equivalent to the old -A level.

Old levels: -BES, -EPES, -C, -X, -XL, -A are not supported any more when signing.

**XAdES-BASELINE-B**

To start, let's take a simple XML document:

```xml
<?xml version="1.0"?>
<test>Hello World !</test>
```

Since this is an XML document, we will use the XAdES signature and more particularly XAdES-BASELINE-B level, which is the lowest level of protection: just satisfying Directive (cf. [R06]) legal requirements for advanced signature. The normal process of signing wants to sign first with the level -B or level-T, and then later when it becomes necessary to complete the signature with superior levels. However, the framework allows signing directly with any level. When signing data, the resulting signature needs to be linked with the data to which it applies. This can be done either by creating a data set which combines the signature and the data (e.g. by enveloping the data with the signature or including a signature element in the data set) or placing the signature in a separate resource and having some external means for associating the signature with the data. So, we need to define the packaging of the signature, namely ENVELOPED, ENVELOPING, DETACHED or INTERNALLY-DETACHED.

- **ENVELOPED**: when the signature applies to data that surround the rest of the document;
- **ENVELOPING**: when the signed data form a sub-element of the signature itself;
  - Base64 encoded binaries;
  - Embed XML object(s);
  - Embed Manifest object(s)
- **DETACHED**: when the signature relates to the external resource(s) separated from it.
- **INTERNALLY-DETACHED**: when the signature and the related signed data are both included in
a parent element (only XML).

For our example, we will use ENVELOPED packaging.

The DSS framework uses 3 atomic steps to sign a document:

1. Compute the digest to be signed;
2. Sign the digest;
3. Sign the document (add the signed digest).

The DSS fully manages the steps 1 and 3. We need to specify how to do the signature operation. DSS offers some implementations in the dss-token module.

To write our Java code, we still need to specify the type of KeyStore to use for signing our document, more simply, where the private key can be found. In the package "eu.europa.esig.dss.token", we can choose between different connection tokens:

- **Pkcs11SignatureToken**: allows to communicate with SmartCards with the PKCS#11 interface. It requires some installed drivers (dll, sso,...)
- **Pkcs12SignatureToken**: allows to sign with a PKCS#12 keystore (.p12 file).
- **MSCAPISignatureToken**: handles the signature with MS CAPI (the Microsoft interface to communicate with SmartCards).
- **JKSSignatureToken**: allows to sign with a Java Key Store (.jks file).

The DSS also provides the support for MOCCA framework to communicate with the Smartcard with PC/SC, but it involves the installation of the MOCCA and IAIK libraries.

To know more about the use of the different signature tokens, please consult "Management of Signature Tokens" chapter.

In our example the class: "Pkcs12SignatureToken" will be used. A file in PKCS#12 format must be provided to the constructor of the class. It contains an X.509 private key accompanying the public key certificate and protected by symmetrical password. The certification chain can also be included in this file. It is possible to generate dummy certificates and their chains with OpenSSL. Please visit [http://www.openssl.org/](http://www.openssl.org/) for more details.

This is the complete code that allows you to sign our XML document.
Create a XAdES signature

```java
// Preparing parameters for the XAdES signature
XAdESSignatureParameters parameters = new XAdESSignatureParameters();
// We choose the level of the signature (-B, -T, -LT, -LTA).
parameters.setSignatureLevel(SignatureLevel.XAdES_BASELINE_B);
// We choose the type of the signature packaging (ENVELOPED, ENVELOPING, DETACHED).
parameters.setSignaturePackaging(SignaturePackaging.ENVELOPED);
// We set the digest algorithm to use with the signature algorithm. You must use the
// same parameter when you invoke the method sign on the token. The default value is SHA256
parameters.setDigestAlgorithm(DigestAlgorithm.SHA256);

// We set the signing certificate
parameters.setSigningCertificate(privateKey.getCertificate());
// We set the certificate chain
parameters.setCertificateChain(privateKey.getCertificateChain());

// Create common certificate verifier
CommonCertificateVerifier commonCertificateVerifier = new CommonCertificateVerifier();

// Create XAdES service for signature
XAdESService service = new XAdESService(commonCertificateVerifier);

// Get the SignedInfo XML segment that need to be signed.
ToBeSigned dataToSign = service.getDataToSign(toSignDocument, parameters);

// This function obtains the signature value for signed information using the
// private key and specified algorithm
SignatureValue signatureValue = signingToken.sign(dataToSign, parameters .getDigestAlgorithm(), privateKey);

// We invoke the service to sign the document with the signature value obtained in
// the previous step.
DSSDocument signedDocument = service.signDocument(toSignDocument, parameters, signatureValue);
```

What you may notice is that to sign a document we need to:

- Create an object based on SignatureParameters class. The number of specified parameters depends on the type of signature. Generally, the number of specified parameters depends on the profile of signature. This object also defines some default parameters.
- Choose the profile, packaging, signature digest algorithm.
- Indicate the private key entry to be used.
- Instantiate the adequate signature service.
- Carry out the signature process.

The encryption algorithm is determined by the private key and therefore cannot be compelled by
the setter of the signature parameters object. It will cause an inconsistency in the signature making its validation impossible. This setter can be used in a particular context where the signing process is distributed on different machines and the private key is known only to the signature value creation process. See clause "Signing process" for more information. In the case where the private key entry object is not available, it is possible to choose the signing certificate and its certificate chain as in the following example:

```java
// We set the signing certificate
parameters.setSigningCertificate(certificateToken);
// We set the certificate chain
parameters.setCertificateChain(certificateChain);
```

Integrating the certificate chain in the signature simplifies the build of a prospective certificate chain during the validation process.

By default the framework uses the current date time to set the signing date, but in the case where it is necessary to indicate the different time it is possible to use the setter "setSigningDate(Date)" as in the example:

```java
// We set the date of the signature.
parameters.bLevel().setSigningDate(new Date());
```

When the specific service is instantiated a certificate verifier must be set. This object is used to provide four different sources of information:

- the source of trusted certificates (based on the trusted list(s) specific to the context);
- the source of intermediate certificates used to build the certificate chain till the trust anchor. This source is only needed when these certificates are not included in the signature itself;
- the source of OCSP;
- the source of CRL.

In the current implementation this object is only used when profile -LT or -LTA are created.

**Signing process**

Once the parameters of the signature were identified the service object itself must be created. The service used will depend on the type of document to sign. In our case it is an XML file, so we will instantiate a XAdES service. The process of signing takes place in three stages. The first is the "getDataToSign ()" method call, passing as a parameter the document to be signed and the previously selected settings. This step returns the data which is going to be digested and encrypted. In our case it corresponds to the SignedInfo XMLDSig element.
// Create XAdES service for signature
XAdESService service = new XAdESService(commonCertificateVerifier);

// Get the SignedInfo XML segment that need to be signed.
ToBeSigned dataToSign = service.getDataToSign(toSignDocument, parameters);

The next step is a call to the function "sign()" which is invoked on the object token representing the
KeyStore and not on the service. This method takes three parameters. The first is the array of bytes
that must be signed. It is obtained by the previous method invocation. The second is the algorithm
used to create the digest. You have the choice between SHA1, SHA256, and SHA512 (this list is not
exhaustive). And the last one is the private key entry.

DigestAlgorithm digestAlgorithm = parameters.getDigestAlgorithm();
SignatureValue signatureValue = signingToken.sign(dataToSign, digestAlgorithm,
privateKey);

The last step of this process is the integration of the signature value in the signature and linking of
that one to the signed document based on the selected packaging method. This is the method
"signDocument()" on the service. We must pass to it three parameters: again the document to sign,
the signature parameters and the value of the signature obtained in the previous step.

This separation into three steps allows use cases where different environments have their precise
responsibilities: specifically the distinction between communicating with the token and executing
the business logic.

When the breakdown of this process is not necessary than a simple call to only one method can be
done as in the following example:

DSSDocument signedDocument = service.signDocument(toSignDocument, parameters,
signatureValue);

Additional attributes

For this type (XAdES-BASELINE-B) of signature it is possible to identify some additional attributes:

- SignerRole - contains claimed roles assumed by the signer when creating the signature.
- SignatureProductionPlace - contains the indication of the purported place where the signer
claims to have produced the signature.
- CommitmentTypeIndication - identifies the commitment undertaken by the signer in signing (a)
signed data object(s) in the context of the selected signature policy.
- AllDataObjectsTimeStamp - each time-stamp token within this property covers the full set of
references defined in the Signature's SignedInfo element, excluding references of type
"SignedProperties".
- IndividualDataObjectsTimeStamp - each time-stamp token within this property covers selected
signed data objects.
The DSS framework allows to setup the following signed properties: SignerRole, SignatureProductionPlace, CommitmentTypeIndication, AllDataObjectsTimestamp and IndividualDataObjectsTimeStamp.

**XAdES signature with additional signed attributes**

```java
XAdESSignatureParameters parameters = new XAdESSignatureParameters();

// Basic signature configuration
parameters.setSignaturePackaging(SignaturePackaging.ENVELOPED);
parameters.setSignatureLevel(SignatureLevel.XAdES_BASELINE_B);
parameters.setDigestAlgorithm(DigestAlgorithm.SHA512);
parameters.setSigningCertificate(privateKey.getCertificate());
parameters.setCertificateChain(privateKey.getCertificateChain());

// Configuration of several signed attributes like ...
BLevelParameters bLevelParameters = parameters.bLevel();

// claimed signer role(s)
bLevelParameters.setClaimedSignerRoles(Arrays.asList("Manager"));

// signer location
SignerLocation signerLocation = new SignerLocation();
signerLocation.setCountry("BE");
signerLocation.setStateOrProvince("Luxembourg");
signerLocation.setPostalCode("1234");
signerLocation.setLocality("SimCity");
bLevelParameters.setSignerLocation(signerLocation);

// commitment type(s)
List<String> commitmentTypeIndications = new ArrayList<String>();
commitmentTypeIndications.add(CommitmentType.ProofOfOrigin.getUri());
commitmentTypeIndications.add(CommitmentType.ProofOfApproval.getUri());
bLevelParameters.setCommitmentTypeIndications(commitmentTypeIndications);

CommonCertificateVerifier verifier = new CommonCertificateVerifier();
XAdESService service = new XAdESService(verifier);
service.setTspSource(getOnlineTSPSource());

// a content-timestamp (part of the signed attributes)
TimestampToken contentTimestamp = service.getContentTimestamp(toSignDocument, parameters);
parameters.setContentTimestamps(Arrays.asList(contentTimestamp));

// Signature process with its 3 stateless steps
ToBeSigned dataToSign = service.getDataToSign(toSignDocument, parameters);
SignatureValue signatureValue = signingToken.sign(dataToSign, parameters .getDigestAlgorithm(), privateKey);
DSSDocument signedDocument = service.signDocument(toSignDocument, parameters, signatureValue);
```
This code adds the following elements into the signature:

```xml
<xades:SignedProperties Id="xades-id-ea3e16770317bb1a3e97244292931644">
    <xades:SignedSignatureProperties>
        <xades:SigningTime>2018-03-20T08:17:35Z</xades:SigningTime>
        <xades:SigningCertificateV2>
            <xades:Cert>
                <xades:CertDigest>
                    <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                    <ds:DigestValue>2FeANjXziO9x2877SfclRRLVj1E=</ds:DigestValue>
                </xades:CertDigest>
            </xades:Cert>
        </xades:SigningCertificateV2>
        <xades:SignatureProductionPlaceV2>
            <xades:City>SimCity</xades:City>
            <xades:StateOrProvince>Luxembourg</xades:StateOrProvince>
            <xades:PostalCode>1234</xades:PostalCode>
            <xades:CountryName>BE</xades:CountryName>
        </xades:SignatureProductionPlaceV2>
        <xades:SignerRoleV2>
            <xades:ClaimedRoles>
                <xades:ClaimedRole>Manager</xades:ClaimedRole>
            </xades:ClaimedRoles>
        </xades:SignerRoleV2>
    </xades:SignedSignatureProperties>
    <xades:SignedDataObjectProperties>
        <xades:DataObjectFormat ObjectReference="#r-id-1">
            <xades:MimeType>text/xml</xades:MimeType>
        </xades:DataObjectFormat>
        <xades:CommitmentTypeIndication>
            <xades:CommitmentTypeId>
                <xades:Identifier>http://uri.etsi.org/01903/v1.2.2#ProofOfOrigin</xades:Identifier>
            </xades:CommitmentTypeId>
        </xades:CommitmentTypeIndication>
    </xades:SignedDataObjectProperties>
</xades:SignedProperties>
```

http://uri.etsi.org/01903/v1.2.2#ProofOfOrigin
Handling signature policy

With the new standards the policy handling is linked to -B level. The old -EPES level is not used anymore by the framework. This does not alter the structure of the old signature but only modifies how to control the process of its creation.

The DSS framework allows you to reference a signature policy, which is a set of rules for the creation and validation of an electronic signature. It includes two kinds of text:

- In human readable form: It can be assessed to meet the requirements of the legal and contractual context in which it is being applied.
- In a machine processable form: To facilitate its automatic processing using the electronic rules.

If no signature policy is identified then the signature may be assumed to have been generated or verified without any policy constraints, and hence may be given no specific legal or contractual significance through the context of a signature policy.

The signer may reference the policy either implicitly or explicitly. An implied policy means the signer follows the rules of the policy but the signature does not indicate which policy. It is assumed the choice of policy is clear from the context in which the signature is used and SignaturePolicyIdentifier element will be empty. When the policy is not implied, the signature contains an ObjectIdentier that uniquely identifies the version of the policy in use. The signature also contains a hash of the policy document to make sure that the signer and verifier agree on the contents of the policy document.

This example demonstrates an implicit policy identifier. To implement this alternative you must set SignaturePolicyId to empty string.
XAdES with implicit policy

```java
XAdESSignatureParameters parameters = new XAdESSignatureParameters();
parameters.setSignatureLevel(SignatureLevel.XAdES_BASELINE_B);
parameters.setSignaturePackaging(SignaturePackaging.ENVELOPED);
parameters.setDigestAlgorithm(DigestAlgorithm.SHA256);

// We set the signing certificate
parameters.setSigningCertificate(privateKey.getCertificate());
// We set the certificate chain
parameters.setCertificateChain(privateKey.getCertificateChain());

BLevelParameters bLevelParameters = parameters.bLevel();

Policy policy = new Policy();
policy.setId(""");

bLevelParameters.setSignaturePolicy(policy);

// Create common certificate verifier
CommonCertificateVerifier commonCertificateVerifier = new CommonCertificateVerifier();
// Create xadesService for signature
XAdESService service = new XAdESService(commonCertificateVerifier);

// Get the SignedInfo segment that need to be signed.
ToBeSigned dataToSign = service.getDataToSign(toSignDocument, parameters);

// This function obtains the signature value for signed information using the
// private key and specified algorithm
DigestAlgorithm digestAlgorithm = parameters.getDigestAlgorithm();
SignatureValue signatureValue = signingToken.sign(dataToSign, digestAlgorithm, privateKey);

// We invoke the xadesService to sign the document with the signature value obtained
// in
// the previous step.
DSSDocument signedDocument = service.signDocument(toSignDocument, parameters, signatureValue);
```

An XML segment will be added to the signature’s qualified and signed properties:

```
<xades:SignaturePolicyIdentifier>
    <xades:SignaturePolicyId>
        <xades:SignaturePolicyImplied/>
    </xades:SignaturePolicyId>
</xades:SignaturePolicyIdentifier>
```

The next example demonstrates an explicit policy identifier. This is obtained by setting -B profile
signature policy and assigning values to the policy parameters. The Signature Policy Identifier is a
URI or OID that uniquely identifies the version of the policy document. The signature will contain the identifier of the hash algorithm and the hash value of the policy document. The DSS framework does not automatically calculate the hash value; it is to the developer to proceed with the calculation using for example java.security.MessageDigest class (rt.jar). It is important to keep the policy file intact in order to keep the hash constant. It would be wise to make the policy file read-only. See also chapter 7 for further information.
XAdES with explicit policy

XAdESSignatureParameters parameters = new XAdESSignatureParameters();
parameters.setSignatureLevel(SignatureLevel.XAdES_BASELINE_B);
parameters.setSignaturePackaging(SignaturePackaging.ENVELOPED);
parameters.setDigestAlgorithm(DigestAlgorithm.SHA256);

// We set the signing certificate
parameters.setSigningCertificate(privateKey.getCertificate());
// We set the certificate chain
parameters.setCertificateChain(privateKey.getCertificateChain());

BLevelParameters bLevelParameters = parameters.bLevel();

// Get and use the explicit policy
String signaturePolicyId = "http://www.example.com/policy.txt";
DigestAlgorithm signaturePolicyHashAlgo = DigestAlgorithm.SHA256;
String signaturePolicyDescription = "Policy text to digest";
byte[] signaturePolicyDescriptionBytes = signaturePolicyDescription.getBytes();
byte[] digestedBytes = DSSUtils.digest(signaturePolicyHashAlgo,
signaturePolicyDescriptionBytes);

Policy policy = new Policy();
policy.setId(signaturePolicyId);
policy.setDigestAlgorithm(signaturePolicyHashAlgo);
policy.setDigestValue(digestedBytes);

bLevelParameters.setSignaturePolicy(policy);

// Create common certificate verifier
CommonCertificateVerifier commonCertificateVerifier = new CommonCertificateVerifier();
// Create xadesService for signature
XAdESService service = new XAdESService(commonCertificateVerifier);

// Get the SignedInfo segment that need to be signed.
ToBeSigned dataToSign = service.getDataToSign(toSignDocument, parameters);

// This function obtains the signature value for signed information using the
// private key and specified algorithm
DigestAlgorithm digestAlgorithm = parameters.getDigestAlgorithm();
SignatureValue signatureValue = signingToken.sign(dataToSign, digestAlgorithm,
privateKey);

// We invoke the xadesService to sign the document with the signature value obtained in
// the previous step.
DSSDocument signedDocument = service.signDocument(toSignDocument, parameters,
signatureValue);

The following XML segment will be added to the signature qualified & signed properties
XAdES-BASELINE-T

XAdES-BASELINE-T is a signature for which there exists a trusted time associated to the signature. It provides the initial steps towards providing long term validity and more specifically it provides a protection against repudiation. This extension of the signature can be created as well during the generation process as validation process. However, the case when these validation data are not added during the generation process should no longer occur. The XAdES-BASELINE-T trusted time indications must be created before the signing certificate has been revoked or expired and close to the time that the XAdES signature was produced. The XAdES-BASELINE-T form must be built on a XAdES-BASELINE-B form. The DSS framework allows extending the old -BES and -EPES profiles to the new BASELINE-T profile, indeed there is no difference in the structure of the signature.

To implement this profile of signature you must indicate to the service the TSA source, which delivers from each Timestamp Request a Timestamp Response (RFC 3161 (cf. [R07])) containing tokens. Below is the source code that creates a XAdES-BASELINE-T signature. For our example, we will use the Belgian provider and an instance of OnlineTSPSource (see "TSP Sources" chapter for more details).
Create a XAdES-Baseline-T with an OnlineTSPSource

```java
// Preparing parameters for the XAdES signature
XAdESSignatureParameters parameters = new XAdESSignatureParameters();
// We choose the level of the signature (-B, -T, -LT, -LTA).
parameters.setSignatureLevel(SignatureLevel.XAdES_BASELINE_T);
// We choose the type of the signature packaging (ENVELOPED, ENVELOPING, DETACHED).
parameters.setSignaturePackaging(SignaturePackaging.ENVELOPED);
// We set the digest algorithm to use with the signature algorithm. You must use the
// same parameter when you invoke the method sign on the token. The default value is
SHA256
parameters.setDigestAlgorithm(DigestAlgorithm.SHA256);

// We set the signing certificate
parameters.setSigningCertificate(privateKey.getCertificate());
// We set the certificate chain
parameters.setCertificateChain(privateKey.getCertificateChain());

// Create common certificate verifier
CommonCertificateVerifier commonCertificateVerifier = new CommonCertificateVerifier();
// Create XAdES service for signature
XAdESService service = new XAdESService(commonCertificateVerifier);

// Set the Timestamp source
String tspServer = "http://tsa.belgium.be/connect";
OnlineTSPSource onlineTSPSource = new OnlineTSPSource(tspServer);
service.setTspSource(onlineTSPSource);

ToBeSigned dataToSign = service.getDataToSign(toSignDocument, parameters);

SignatureValue signatureValue = signingToken.sign(dataToSign, parameters
 .getDigestAlgorithm(), privateKey);

DSSDocument signedDocument = service.signDocument(toSignDocument, parameters,
signatureValue);
```

If the timestamp source is not set a NullPointerException is thrown.

The SignatureTime Stamp mandated by the XAdES-T form appears as an unsigned property within
the QualifyingProperties:
XAdES-BASELINE-LT

This level has to prove that the certification path was valid, at the time of the validation of the signature, up to a trust point according to the naming constraints and the certificate policy constraints from the "Signature Validation Policy". It will add to the signature the CertificateValues and RevocationValues unsigned properties. The CertificateValues element contains the full set of certificates that have been used to validate the electronic signature, including the signer’s certificate. However, it is not necessary to include one of those certificates, if it is already present in the ds:KeyInfo element of the signature. This is like DSS framework behaves. In order to find a list of all the certificates and the list of all revocation data, an automatic process of signature validation is executed. To carry out this process an object called CertificateVerifier must be passed to the service. The implementer must set some of its properties like for example the source of trusted certificates. The code below shows how to use the default parameters with this object. Please refer to "The Signature Validation" chapter to have the further information. It also includes an example of how to implement this level of signature:

SignXmlXadesLTTest.java

```java
// Preparing parameters for the XAdES signature
XAdESSignatureParameters parameters = new XAdESSignatureParameters();
// We choose the level of the signature (-B, -T, -LT, -LTA).
parameters.setSignatureLevel(SignatureLevel.XAdES_BASELINE_LT);
// We choose the type of the signature packaging (ENVELOPED, ENVELOPING, DETACHED).
parameters.setSignaturePackaging(SignaturePackaging.ENVELOPED);
// We set the digest algorithm to use with the signature algorithm. You must use the same parameter when you invoke the method sign on the token. The default value is SHA256
parameters.setDigestAlgorithm(DigestAlgorithm.SHA256);

// We set the signing certificate
parameters.setSigningCertificate(privateKey.getCertificate());
// We set the certificate chain
parameters.setCertificateChain(privateKey.getCertificateChain());

// Create common certificate verifier
CommonCertificateVerifier commonCertificateVerifier = new CommonCertificateVerifier();
CommonsDataLoader commonsHttpDataLoader = new CommonsDataLoader();
OCSPDataLoader ocspDataLoader = new OCSPDataLoader();
```
KeyStoreCertificateSource keystoreCertificateSource = new KeyStoreCertificateSource (new File("src/main/resources/keystore.p12"), "PKCS12", "dss-password");

TrustedListsCertificateSource tlsCertificateSource = new TrustedListsCertificateSource();

TSLRepository tslRepository = new TSLRepository();
    tslRepository.setTrustedListsCertificateSource(tlsCertificateSource);

TSLValidationJob job = new TSLValidationJob();
    job.setDataLoader(commonsHttpDataLoader);
    job.setOjContentKeyStore(keyStoreCertificateSource);
    job.setLotlCode("EU");
    job.setRepository(tslRepository);
    job.refresh();

commonCertificateVerifier.setTrustedCertSource(tlsCertificateSource);

OnlineCRLSource onlineCRLSource = new OnlineCRLSource();
    onlineCRLSource.setDataLoader(commonsHttpDataLoader);
    commonCertificateVerifier.setCrlSource(onlineCRLSource);

OnlineOCSPSource onlineOCSPSource = new OnlineOCSPSource();
    onlineOCSPSource.setDataLoader(ocspDataLoader);
    commonCertificateVerifier.setOcspSource(onlineOCSPSource);

// For test purpose
// Will request unknown OCSP responder / download untrusted CRL
commonCertificateVerifier.setCheckRevocationForUntrustedChains(true);

// Create XAdES service for signature
XAdESService service = new XAdESService(commonCertificateVerifier);
    service.setTspSource(getOnlineTSPSource());

// Get the SignedInfo XML segment that need to be signed.
    ToBeSigned dataToSign = service.getDataToSign(toSignDocument, parameters);

    // This function obtains the signature value for signed information using the
    // private key and specified algorithm
    SignatureValue signatureValue = signingToken.sign(dataToSign, parameters .getDigestAlgorithm(), privateKey);

    // We invoke the service to sign the document with the signature value obtained in
DSSDocument signedDocument = service.signDocument(toSignDocument, parameters, signatureValue);

The following XML segment will be added to the signature qualified and unsigned properties:

```xml
<CertificateValues>
  <EncapsulatedX509Certificate>
    MIIFNTCCBB2gAwIBAgIBATANB...
  </EncapsulatedX509Certificate>
  <EncapsulatedX509Certificate>
    MIIFsjCCBJgAwIBAgIDAMoBM...
  </EncapsulatedX509Certificate>
  <EncapsulatedX509Certificate>
    MIIFRjCCBC6gAwIBAgIBATANB...
  </EncapsulatedX509Certificate>
</CertificateValues>
<RevocationValues>
  <OCSPValues>
    <EncapsulatedOCSPValue>
      MIIGzAoBAKCCBSUwggBBgkr...
    </EncapsulatedOCSPValue>
  </OCSPValues>
</RevocationValues>
```

The use of online sources can significantly increase the execution time of the signing process. For testing purpose you can create your own source of data.

In last example the CommonsHttpDataLoader is used to provide the communication layer for HTTP protocol. Each source which need to go through the network to retrieve data need to have this component set.

**XAdES-BASELINE-LTA**

When the cryptographic data becomes weak and the cryptographic functions become vulnerable the auditor should take steps to maintain the validity of the signature. The XAdES-BASELINE-A form uses a simple approach called "archive validation data". It adds additional time-stamps for archiving signatures in a way that they are still protected, but also to be able to prove that the signatures were validated at the time when the used cryptographic algorithms were considered safe. The time-stamping process may be repeated every time the protection used becomes weak. Each time-stamp needs to be affixed before either the signing key or the algorithms used by the TSA are no longer secure. XAdES-A form adds the ArchiveTimestamp element within the UnsignedSignatureProperties and may contain several ArchiveTimestamp elements.

Below is an example of the implementation of this level of signature (but in practice, we will rather extend the signature to this level when there is a risk that the cryptographic functions become vulnerable or when one of certificates arrives to its expiration date):
parameters.setSignatureLevel(SignatureLevel.XAdES_BASELINE_LTA);

The following XML segment will be added to the signature qualified and unsigned properties:

```xml
<ns4:ArchiveTimeStamp
  Id="time-stamp-22b92602-2670-410e-888f-937c5777c685">
  <ds:CanonicalizationMethod
    Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
  <EncapsulatedTimeStamp
    Id="time-stamp-token-0bd5aaf3-3850-4911-a22d-c98dcaca5cea">MIAGCSqGSDHAqCAM…
  </EncapsulatedTimeStamp>
</ns4:ArchiveTimeStamp>
```

Various settings

Trust anchor inclusion policy

It is possible to indicate to the framework if the certificate related to the trust anchor should be included to the signature or not. The setter #setTrustAnchorBPPolicy of the BLevelParameters class should be used for this purpose.

This rule applies as follows: when -B level is constructed the trust anchor is not included, when -LT level is constructed the trust anchor is included.

- when trust anchor baseline profile policy is defined only the certificates previous to the trust anchor are included when -B level is constructed.

Multiple signatures

In everyday life, there are many examples where it is necessary to have multiple signatures covering the same document, such as a contract to purchase a vehicle. Independent signatures are parallel signatures where the ordering of the signatures is not important. The computation of these signatures is performed on exactly the same input but using different private keys.

The XML Signature Extension (XAdES)

The -B level contains immutable signed properties. Once this level is created, these properties cannot be changed.

The levels -T/-LT/-LTA add unsigned properties to the signature. This means that the properties of these levels could be added afterwards to any AdES signature. This addition helps to make the signature more resistant to cryptographic attacks on a longer period of time. The extension of the signature is incremental, i.e. when you want to extend the signature to the level -LT the lower level
(-T) will also be added. The whole extension process is implemented by reusing components from signature production. To extend a signature we proceed in the same way as in the case of a signature, except that you have to call the function "extendDocument" instead of the "sign" function. Note that when the document is signed with several signatures then they are all extended.

**XAdES-BASELINE-T**

The XAdES-BASELINE-T trusted time indications have to be created before a certificate has been revoked or expired and close to the time that the XAdES signature was produced. It provides a protection against repudiation. The framework adds the timestamp only if there is no timestamp or there is one but the creation of a new extension of the level-T is deliberate (using another TSA). It is not possible to extend a signature which already incorporates higher level as -LT or -LTA. In the theory it would be possible to add another -T level when the signature has already reached level -LT but the framework prevents this operation. Note that if the signed document contains multiple signatures, then all the signatures will be extended to level -T. It is also possible to sign a document directly at level -T.

Here is an example of creating an extension of type T:

*Extend a XAdES signature*

```java
DSSDocument document = new FileDocument("src/test/resources/signedXmlXadesB.xml");
XAdESSignatureParameters parameters = new XAdESSignatureParameters();
parameters.setSignatureLevel(SignatureLevel.XAdES_BASELINE_T);
CommonCertificateVerifier certificateVerifier = new CommonCertificateVerifier();
XAdESService xadesService = new XAdESService(certificateVerifier);
xadesService.setTspSource(getOnlineTSPSource());
DSSDocument extendedDocument = xadesService.extendDocument(document, parameters);
```

Here is the result of adding a new extension of type-T to an already existing -T level signature:
XAdES-BASELINE-LT and -LTA

For these types of extensions, the procedure to follow is the same as the case of the extension of type T. Please refer to the chapter XAdES Profiles (XAdES) to know specific parameters for each level of signature and which must be positioned.

XAdES and specific schema version

Some signatures may have been created with an older version of XAdES standard using different schema definition. To take into account the validation of such signatures the class eu.europa.esig.dss.xades.validation.XPathQueryHolder was created. This class includes all XPath queries which are used to explore the elements of the signature. It is now easy to extend this class in order to define specific queries to a given schema. The DSS framework proposes in standard the class eu.europa.esig.dss.xades.validation.XAdES111XPathQueryHolder that defines the XPath queries for the version "http://uri.etsi.org/01903/v1.1.1#" of XAdES standard.

When carrying out the validation process of the signature, the choice of query holder to be used is taken by invoking the method:

eu.europa.esig.dss.xades.validation.XPathQueryHolder#canUseThisXPathQueryHolder

This choice is made based on the namespace. If the namespace is: http://uri.etsi.org/01903/v1.3.2# then the default query holder is used, if the namespace is http://uri.etsi.org/01903/v1.1.1# the XAdES111XPathQueryHolder is used. The element used to choose the namespace is "QualifyingProperties".

To implement another query holder the class XPathQueryHolder must be extended, new XPath queries defined and the method canUseThisXPathQueryHolder overridden.

In case there is a need to use only a specific query holder the following steps should be followed:

- Call: eu.europa.esig.dss.xades.validation.XMLDocumentValidator#clearQueryHolders
The signature validation

Generally and following ETSI standard, the validation process of an electronic signature must provide one of the three following status: TOTAL-FAILED, TOTAL-PASSED or INDETERMINATE. A TOTAL-PASSED response indicates that the signature has passed verification and it complies with the signature validation policy. An TOTAL_FAILED response indicates that either the signature format is incorrect or that the digital signature value fails verification. An INDETERMINATE validation response indicates that the format and digital signature verifications have not failed but there is insufficient information to determine if the electronic signature is valid. For each of the validation checks, the validation process must provide information justifying the reasons for the resulting status indication as a result of the check against the applicable constraints. In addition, the ETSI standard defines a consistent and accurate way for justifying statuses under a set of sub-indications.

Validation Process

Since version 4.7 of the DSS framework the validation process is based on the latest ETSI standard [R08]. It is driven by the validation policy and allows long term signature validation. It not only verifies the existence of certain data and their validity, but it also checks the temporal dependences between these elements. The signature check is done following basic building blocks. On the simplified diagram below, showing the process of the signature validation, you can follow the relationships between each building block which represents a logic set of checks used in validation process.
Note that the current version of the framework during the validation process does not indicate what part of the document was signed. However, in the case of the XAdES signature XPath transformations present in the signature will be applied, in the case of CAdES or PAdES signature the whole document must be signed.

At the end of the validation process, three reports are created. They contain different details level concerning the validation result. They provide three kinds of visions of validation process: macroscopic, microscopic and input data. For more information about these reports, please refer to "Simple Report" chapter.

Below is the simplest example of the validation of the signature of a document. The first thing to do is instantiating an object named validator, which orchestrates the verification of the different rules. To perform this it is necessary to invoke a static method fromDocument() on the abstract class SignedDocumentValidator. This method returns the object in question whose type is chosen dynamically based on the type of source document. The DSS framework provides five types of validators:

- XMLDocumentValidator,
- CMSDocumentValidator,
- PDFDocumentValidator,
- ASiCContainerWithXAdESValidator,
- ASiCContainerWithCAdESValidator.

The next step is to create an object that will check the status of a certificate using the Trusted List model (see "Trusted Lists of Certification Service Provider" for more information). In our example,
this object is instantiated from the TrustedListCertificateVerifier class. In turn, this object needs an OCSP and/or CRL source and a TSL source (which defines how the certificates are retrieved from the Trusted Lists). See chapter "Management of CRL and OCSP Sources" for more information concerning sources.
Validation of a signature

```java
// First, we need a Certificate verifier
CertificateVerifier cv = new CommonCertificateVerifier();

// We can inject several sources. eg: OCSP, CRL, AIA, trusted lists

// Capability to download resources from AIA
cv.setDataLoader(new CommonsDataLoader());

// Capability to request OCSP Responders
cv.setOcspSource(new OnlineOCSPSource());

// Capability to download CRL

cv.setCrlSource(new OnlineCRLSource());

// We now add trust anchors (trusted list, keystore,...)
cv.setTrustedCertSource(trustedCertSource);

// We also can add missing certificates
cv.setAdjunctCertSource(adjunctCertSource);

// Here is the document to be validated (any kind of signature file)
DSSDocument document = new FileDocument(new File("src/test/resources/signedXmlXadesLT.xml");

// We create an instance of DocumentValidator
// It will automatically select the supported validator from the classpath
SignedDocumentValidator documentValidator = SignedDocumentValidator.fromDocument(document);

// We add the certificate verifier (which allows to verify and trust certificates)
documentValidator.setCertificateVerifier(cv);

// Here, everything is ready. We can execute the validation (for the example, we use
the default and embedded
// validation policy)
Reports reports = documentValidator.validateDocument();

// We have 3 reports
// The diagnostic data which contains all used and static data
DiagnosticData diagnosticData = reports.getDiagnosticData();

// The detailed report which is the result of the process of the diagnostic data and
the validation policy
DetailedReport detailedReport = reports.getDetailedReport();

// The simple report is a summary of the detailed report (more user-friendly)
SimpleReport simpleReport = reports.getSimpleReport();
```
When using the TrustedListsCertificateSource class, for performance reasons, consider creating a single instance of this class and initialize it only once.

In general, the signature must cover the entire document so that the DSS framework can validate it. However, for example in the case of a XAdES signature, some transformations can be applied on the XML document. They can include operations such as canonicalization, encoding/decoding, XSLT, XPath, XML schema validation, or XInclude. XPath transforms permit the signer to derive an XML document that omits portions of the source document. Consequently those excluded portions can change without affecting signature validity.

EU Trusted Lists of Certification Service Providers

On 16 October 2009 the European Commission adopted a Decision setting out measures facilitating the use of procedures by electronic means through the "points of single contact" under the Services Directive. One of the measures adopted by the Decision consisted in the obligation for Member States to establish and publish by 28.12.2009 their Trusted List of supervised/accredited certification service providers issuing qualified certificates to the public. The objective of this obligation is to enhance cross-border use of electronic signatures by increasing trust in electronic signatures originating from other Member States. The Decision was updated several times since 16.10.2009; the last amendment was made on 01.02.2014. The consolidated version is available here for information.

In order to allow access to the trusted lists of all Member States in an easy manner, the European Commission has published a central list with links to national "trusted lists". This central list will now be designated in the document under the abbreviation LOTL.

The LOTL is published by the EU at the following URL: https://ec.europa.eu/information_society/policy/esignature/trusted-list/tl-mp.xml. This XML file contains the list of the trusted list. This file must be signed by an allowed certificate. To know who has the permission to sign / publish the LOTL, we need to refer to the Official Journal Of the Union (http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.C_.2016.233.01.0001.01.ENG).

The signature format of the LOTL and TL should be XAdES-BASELINE-B. If the LOTL signature is valid, its content can be trusted. The LOTL contains for each country some information: urls of the XML/PDF files, the allowed certificates to sign, ...

So, we trusted the LOTL, we can process each trusted list. If they are valid, we can trust the service providers and its certificates.

To build the source of trust, DSS requires:

- the LOTL url (XML);
- the LOTL country code;
- a trust store which contains allowed certificates (extracted from the OJ).

Below, you can find a complete example to load the LOTL and its linked TLs.
Configuration to load the LOTL and related TLs

```java
TSLRepository tslRepository = new TSLRepository();

TrustedListsCertificateSource certificateSource = new TrustedListsCertificateSource();
 tslRepository.setTrustedListsCertificateSource(certificateSource);

TSLValidationJob job = new TSLValidationJob();
 job.setDataLoader(new CommonsDataLoader());
 job.setCheckLOTLSignature(true);
 job.setCheckTSLSignatures(true);
 job.setLotlCode("EU");

// This information is needed to be able to filter the LOTL pivots

// The keystore contains certificates referenced in the Official Journal Link (OJ URL)
 KeyStoreCertificateSource keyStoreCertificateSource = new KeyStoreCertificateSource( new File("src/main/resources/keystore.p12"), "PKCS12",
  "dss-password");
 job.setOjContentKeyStore(keyStoreCertificateSource);

 job.setRepository(tslRepository);

 job.refresh();
```

The TrustedListsCertificateSource is updated with the trusted certificates.

To generate the trust store, there’s an utility class CreateKeyStoreApp in the dss-cookbook module.

**Non-European trusted lists support**

Additionally, DSS can load external trusted lists. These trusted lists are checked against their trust store (keystore which contains the authorized TL signers).
Handling of Non-European trusted list(s)

TSLValidationJob job = new TSLValidationJob();
   // ...

   // Configuration to load the peruvian trusted list.
   // DSS requires the country code, the URL and allowed signing certificates
   OtherTrustedList peru = new OtherTrustedList();
   peru.setCountryCode("PE");
   peru.setUrl("https://iofe.indecopi.gob.pe/TSL/tsl-pe.xml");
   peru.setTrustStore(getTrustStore());

   job.setOtherTrustedLists(Arrays.asList(peru));

Validation Result Materials

The result of the validation process consists of three elements:

- the simple report,
- the detailed report and,
- the diagnostic data.

All these reports are encoded using XML, which allows the implementer to easily manipulate and extract information for further analysis. For each report, XML Schema and JAXB model are available as maven dependencies.

DSS also provides XSLT to able to generate PDF or HTML reports (simple and detailed reports).

You will find below a detailed description of each of these elements.

Simple Report

This is a sample of the simple validation report.
Policy

PolicyName="QES AdES QC TL based"
PolicyDescription="Validate electronic signatures and indicates whether they are Advanced electronic Signatures (AdES), AdES supported by a Qualified Certificate (AdES/QC) or a Qualified electronic Signature (QES). All certificates and their related chains supporting the signatures are validated against the EU Member State Trusted Lists (this includes signer’s certificate and certificates used to validate certificate validity status services - CRLs, OCSP, and time-stamps)."
ValidationTime="2018-02-23T05:50:46"
DocumentName="testdocument.pdf"
ValidSignaturesCount="1"
SignaturesCount="1"
SignatureId="f67ab9c96a6fe80ec9f619f8e9b4505b66b078c7a2b6e151ec2bbb"
SignatureFormat="PAdES-BASE-LTA"
SigningTime="2017-10-27T11:54:56"
BestSignatureTime="2017-10-27T11:55:08"
SignedBy="E Van R (Signature)"
CertificateChain
  Certificate
    id="021EC5069EB8A903B62B6769EDDFE439DFA90A720C9A362D5FE76B9C31D0302"
    qualifiedName="E Van R (Signature)"
  Certificate
    id="80AC352930875BA0A0FE7F70DD389130C8E1E7BEFFDC96477356AD2A9E003AD2B"
    qualifiedName="Citizen CA"
  Certificate
    id="702DD5C1A093CF0A9D71FADD9BF9A7C5857D89FB73B716E86722B83C2EBB968F"
    qualifiedName="Belgium Root CA4"
SignatureScopeScope="PdfByteRangeSignatureScope" The document byte range: [0, 9258, 32602, 495939]"
The result of the validation process is based on very complex rules. The purpose of this report is to make as simple as possible the information while keeping the most important elements. Thus the end user can, at a glance, have a synthetic view of the validation. To build this report the framework uses some simple rules and the detailed report as input.

**Detailed Report**

This is a sample of the detailed validation report. Its structure is based on the ETSI standard [R08] and is built around Basic Building Blocks, Basic Validation Data, Timestamp Validation Data, AdES-T Validation Data and Long Term Validation Data. Some segments were deleted to make reading easier. They are marked by three dots:

```
    <Signatures Id="id-2056753e8b67ab9c96a6fe80ec9f619f8e9b4505b66b078c7a2b6e151edc2bbb">
        <ValidationProcessBasicSignatures BestSignatureTime="2018-02-23T05:50:46">
            <Constraint Id="id-2056753e8b67ab9c96a6fe80ec9f619f8e9b4505b66b078c7a2b6e151edc2bbb">
                <Name NameId="ADEST_ROBVPIIC">Is the result of the Basic Validation Process conclusive?</Name>
                <Status>OK</Status>
            </Constraint>
        </ValidationProcessBasicSignatures>
    </Signatures>

    <ValidationProcessTimestamps Id="F8B72B754450E84664DE8B950BC781CA2528CF5B39ABB3E1F93947B752BE79F" Type="SIGNATURE_TIMESTAMP" ProductionTime="2017-10-27T11:55:08">
        <Constraint Id="F8B72B754450E84664DE8B950BC781CA2528CF5B39ABB3E1F93947B752BE79F">
            <Name NameId="ADEST_ROTVPIIC">Is the result of the timestamps validation process conclusive?</Name>
            <Status>OK</Status>
        </Constraint>
    </ValidationProcessTimestamps>

    <ValidationProcessTimestamps Id="7F9B88A8161CC87905298FF8E0CD80516452F8A1480DD6AFAE388B70D18E2A18" Type="ARCHIVE_TIMESTAMP" ProductionTime="2017-10-27T11:55:08">
        <Constraint Id="7F9B88A8161CC87905298FF8E0CD80516452F8A1480DD6AFAE388B70D18E2A18">
            <Name NameId="ADEST_ROTVPIIC">Is the result of the timestamps validation process conclusive?</Name>
            <Status>OK</Status>
        </Constraint>
    </ValidationProcessTimestamps>
</DetailedReport>
```
<ISC>
  <Constraint>
    ...
  </Constraint>
  <Conclusion>
    <Indication>PASSED</Indication>
  </Conclusion>
</ISC>

<CV>
  <Constraint>
    ...
  </Constraint>
  <Conclusion>
    <Indication>PASSED</Indication>
  </Conclusion>
</CV>

<SAV>
  <Constraint>
    ...
  </Constraint>
  <Conclusion>
    <Indication>PASSED</Indication>
  </Conclusion>
</SAV>

<XCV>
  <Constraint>
    ...
  </Constraint>
  <Conclusion>
    <Indication>INDETERMINATE</Indication>
    <SubIndication>OUT_OF_BOUNDS_NO_POE</SubIndication>
    <Errors NameId="BBB_XCV_ICTIVRSC_ANS">The current time is not in the validity range of the signer's certificate.</Errors>
  </Conclusion>
  <SubXCV Id="CB217219BADFC13B4FEA3EFA43882E9FECE49E542DCDBA83428DC6854499A35F" TrustAnchor="false">
    ...
  </SubXCV>
  <SubXCV Id="80AC352930875BA0AFE7F70DD389130CBE1E7BEFFDC9477356AD2A9E003AD2B" TrustAnchor="false">
    ...
  </SubXCV>
  <SubXCV Id="702DD5C1A093CF0A9D71FADD9BF9A7C5857D89FB73B716E867228B3C2EBB968F" TrustAnchor="true">
    ...
  </SubXCV>
</XCV>

<CertificateChain>
  ...
</CertificateChain>
The current time is not in the validity range of the signer's certificate.
For example the Basic Building Blocks are divided into seven elements:

- FC - Format Checking
- ISC - Identification of the Signing Certificate
- VCI - Validation Context Initialization
- RFC - Revocation Freshness Checker
- XCV - X.509 certificate validation
- CV - Cryptographic Verification
- SAV - Signature Acceptance Validation

The following additional elements also can be execute in case of validation in the past:

- PCV - Past Certificate Validation
- VTS - Validation Time Sliding process
- POE extraction - Proof Of Existence extraction
- PSV - Past Signature Validation

Each block contains a number of rules that are executed sequentially. The rules are driven by the constraints defined in the validation policy. The result of each rule is OK or NOT OK. The process is stopped when the first rule fails. Each block also contains a conclusion. If all rules are met then the conclusion node indicates PASSED. Otherwise FAILED or INDETERMINATE indication is returned depending on the ETSI standard definition.

### Diagnostic Data

This is a data set constructed from the information contained in the signature itself, but also from information retrieved dynamically as revocation data and information extrapolated as the mathematical validity of a signature. All this information is independent of the applied validation policy. Two different validation policies applied to the same diagnostic data can lead to different results.

This is an example of the diagnostic data for a PAdES signature. Certain fields and certain values were trimmed or deleted to make reading easier:

```xml
<DiagnosticData xmlns="http://dss.esig.europa.eu/validation/diagnostic"/>
```
<EncodedSignedDataDigestValue>kS17/pxPekLwE8UcpnhxyZd/8gkQ8IAhho9KI+2yuh25qyB4qS7ozZ7L85YsCy66ByRGweAG/mwK8RfxjoA=/</EncodedSignedDataDigestValue>
<MessageImprintDataFound>true</MessageImprintDataFound>
<MessageImprintDataIntact>true</MessageImprintDataIntact>
<BasicSignature>
  <EncryptionAlgoUsedToSignThisToken>RSA</EncryptionAlgoUsedToSignThisToken>
  <KeyLengthUsedToSignThisToken>2048</KeyLengthUsedToSignThisToken>
  <DigestAlgoUsedToSignThisToken>SHA512</DigestAlgoUsedToSignThisToken>
  <ReferenceDataFound>true</ReferenceDataFound>
  <ReferenceDataIntact>true</ReferenceDataIntact>
  <SignatureIntact>true</SignatureIntact>
  <SignatureValid>true</SignatureValid>
</BasicSignature>
<SigningCertificate_Id="EE3C22E06087BFEC213709AD3E7F2DDA9CE9D19CE238DCA81A6433E9070A9FBE"/>
<CertificateChain>
  <ChainItem_Id="EE3C22E06087BFEC213709AD3E7F2DDA9CE9D19CE238DCA81A6433E9070A9FBE">
    <Source>TIMESTAMP</Source>
  </ChainItem>
  <ChainItem_Id="702DD5C1A093CF09ADD9BF9A7C5857D89FB73B716E867228B3C2BEB968F">
    <Source>TRUSTED_LIST</Source>
  </ChainItem>
</CertificateChain>
<TimestampedObjects>
  <TimestampedObject_Id="id-2056753e8b67ab9c96a6fe80ec9f619f8e9b4505b66b078c7a2b6e151edc2bb" Category="SIGNATURE">
    <TimestampedObject Category="CERTIFICATE">
      <DigestAlgoAndValue>
        <DigestMethod>SHA256</DigestMethod>
        <DigestValue>Ah7FBp64q0Q9YrZ2nt3+Q536kKcgyaNi1f52ucMdAwI=</DigestValue>
      </DigestAlgoAndValue>
    </TimestampedObject>
  </TimestampedObject>
</TimestampedObjects>
<DigestValue>Ah7FBp64q0Q9YrZ2nt3+Q536kKcgyaNi1f52ucMdAwI=</DigestValue>
<Timestamp_Id="7f9b88a8161cc8705298ff8e0cd080516452fba1480dd6afae38b7dd18e2a18" Type="ARCHIVE_TIMESTAMP">
  ....
</Timestamp_Id>
</Timestamps>
<SignatureScopes>
  <SignatureScope name="PDF previous version #1" scope="PdfByteRangeSignatureScope">
The document byte range: [0, 9258, 32602,
<Certificate Id="021EC5069EBBA003B0D2867696DDEFE439DA90A720C9A36D56E76B9C31D0302">

<SubjectDistinguishedName Format="CANONICAL">
2.5.4.5=#130b3837303533303236323231,2.5.4.42=#130b456c696e65204765726461,2.5.4.4=#130d56616e205261656d6466e6636b, cn=E van R (signature), c=be</SubjectDistinguishedName>

<SubjectDistinguishedName Format="RFC2253">
2.5.4.5=#130b3837303533303236323231,2.5.4.42=#130b456c696e65204765726461,2.5.4.4=#130d56616e205261656d6466f6636b, CN=E Van R (Signature), C=BE</SubjectDistinguishedName>

<IssuerDistinguishedName Format="CANONICAL">
2.5.4.5=#130b38373031373130,cn=citizen
ca,o=http://repository.eid.belgium.be/, c=be</IssuerDistinguishedName>

<IssuerDistinguishedName Format="RFC2253">
2.5.4.5=#1306323031373130, CN=Citizen
CA,O=http://repository.eid.belgium.be/, C=BE</IssuerDistinguishedName>

<SerialNumber>21267647932559290630671294378886251870</SerialNumber>

<CommonName>E Van R (Signature)</CommonName>

<CountryName>BE</CountryName>

<GivenName>E G</GivenName>

<Surname>Van R</Surname>

<AuthorityInformationAccessUrls>
</AuthorityInformationAccessUrls>

<CRLDistributionPoints>
    <Url>http://crl.eid.belgium.be/eidc201710.crl</Url>
</CRLDistributionPoints>

<OCSPAccessUrls>
    <Url>http://ocsp.eid.belgium.be/2</Url>
</OCSPAccessUrls>

<DigestAlgoAndValues>
    <DigestAlgoAndValue>
        <DigestMethod>SHA256</DigestMethod>
        <DigestValue>Ah7FBp64q0Q09YrZ2nt3+Q536kKcgyanIi1f52ueMdAwI</DigestValue>
    </DigestAlgoAndValue>
    <DigestAlgoAndValue>
        <DigestMethod>SHA1</DigestMethod>
        <DigestValue>WWUnOSgChkeevrP7omdQuS/plaNQ</DigestValue>
    </DigestAlgoAndValue>
</DigestAlgoAndValues>

<NotAfter>2027-03-13T23:59:59</NotAfter>


<PublicKeySize>2048</PublicKeySize>

<PublicKeyEncryptionAlgo>RSA</PublicKeyEncryptionAlgo>

<KeyUsageBits>
    <KeyUsage>nonRepudiation</KeyUsage>
</KeyUsageBits>
<ExtendedKeyUsages />
<IdPkixOcspNoCheck>false</IdPkixOcspNoCheck>

<BasicSignature>
  <EncryptionAlgoUsedToSignThisToken>
    RSA
  </EncryptionAlgoUsedToSignThisToken>
  <KeyLengthUsedToSignThisToken>4096</KeyLengthUsedToSignThisToken>
  <DigestAlgoUsedToSignThisToken>SHA256</DigestAlgoUsedToSignThisToken>
  <ReferenceDataFound>true</ReferenceDataFound>
  <ReferenceDataIntact>true</ReferenceDataIntact>
  <SignatureIntact>true</SignatureIntact>
  <SignatureValid>true</SignatureValid>
</BasicSignature>

<SigningCertificate Id="80AC3529308758A0AFE7F70DD389130CBE1E7BEFFDC96477356AD2A9E003AD2B"/>

<CertificateChain>
  <ChainItem Id="80AC3529308758A0AFE7F70DD389130CBE1E7BEFFDC96477356AD2A9E003AD2B">
    <Source>UNKNOWN</Source>
  </ChainItem>
  <ChainItem Id="702DD5C1A093CF0A9D71FADD9BF9A7C5857D89FB73B716E867228B3C2BE968F">
    <Source>TRUSTED_LIST</Source>
  </ChainItem>
</CertificateChain>

<Trusted>false</Trusted>
<SelfSigned>false</SelfSigned>

<CertificatePolicies>
  <certificatePolicy cpsUrl="http://repository.eid.belgium.be">
    2.16.56.12.1.1.2.1</certificatePolicy>
  </CertificatePolicies>
</QCStatementIds>

<QCTypes/>

<TrustedServiceProviders>
  <TrustedServiceProvider>
    <TSPName>Certipost n.v./s.a.</TSPName>
    <TSPRegistrationIdentifier>VATBE-0475396406</TSPRegistrationIdentifier>
    <CountryCode>BE</CountryCode>
    <TrustedServices>
      <TrustedService>
        <ServiceName>CN=Belgium Root CA4, C=BE</ServiceName>
        <ServiceType>http://uri.etsi.org/TrstSvc/Svctype/CA/QC</ServiceType>
      </TrustedService>
    </TrustedServices>
  </TrustedServiceProvider>
</TrustedServiceProviders>

>Status>http://uri.etsi.org/TrstSvc/TrustedList/Svctype/QC</Status>
  <StartDate>2016-06-30T22:00:00</StartDate>
  <CapturedQualifiers>
<Qualifier>http://uri.etsi.org/TrstSvc/TrustedList/SvcInfoExt/QCQSCDStatusAsInCert</Qualifier>
</CapturedQualifiers>
<AdditionalServiceInfoUris>
<URI>http://uri.etsi.org/TrstSvc/TrustedList/SvcInfoExt/RootCA-QC</URI>
</AdditionalServiceInfoUris>
<URI>http://uri.etsi.org/TrstSvc/TrustedList/SvcInfoExt/ForeSignatures</URI>
</TrustedService>
</TrustedServices>
</TrustedServiceProvider>
</TrustedServiceProviders>
<Revocations>
<Revocation Id="021ec5069eb8a903bd62b6769eddfe439dfa90a720c9a362d5fe76b9c31d0302bfc08d553a774d1f440ab36525a3290e2cc23b46d0ac954ea4d8201faff0d91a"/>
<Origin>SIGNATURE</Origin>
<source>OCSPToken</source>
>Status>true</Status>
<ProductionDate>2017-10-27T11:55:08</ProductionDate>
<thisUpdate>2017-10-27T11:55:08</thisUpdate>
<NextUpdate>2017-10-27T11:56:08</nextUpdate>
<DigestAlgoAndValues>
<DigestAlgoAndValue>
<DigestMethod>SHA256</DigestMethod>
<DigestValue>v8CNVTp3TR9ECrNlJaMpDizCO0bQrJVoNggh6/w2Ro=</DigestValue>
</DigestAlgoAndValue>
<DigestAlgoAndValue>
<DigestMethod>SHA1</DigestMethod>
<DigestValue>MRhWbZTCsnogtBv4KZ5Gze2imWA=</DigestValue>
</DigestAlgoAndValue>
</DigestAlgoAndValues>
</BasicSignature>
<EncryptionAlgoUsedToSignThisToken>RSA</EncryptionAlgoUsedToSignThisToken>
<KeyLengthUsedToSignThisToken>2048</KeyLengthUsedToSignThisToken>
</DigestAlgoUsedToSignThisToken>
<ReferenceDataFound>true</ReferenceDataFound>
<ReferenceDataIntact>true</ReferenceDataIntact>
<SignatureIntact>true</SignatureIntact>
<SignatureValid>true</SignatureValid>
</BasicSignature>
</BasicSignature>
</CertificateChain>
<ChainItem_Id="CB217219BADFC13B4FEA3EFA43882E9FECE49E542DCDBA83428DC6854499A35F">"
<Source>OCSP_RESPONSE</Source>

</ChainItem>

<ChainItem Id="80AC3529308758A0AFE7F70DD389130C8E1E7BEFFDC96477356AD2A9E003AD2B">
  <Source>UNKNOWN</Source>
</ChainItem>

<ChainItem Id="702DD5C1A093CF0A9D71FADD9BF9A7C5B57D89FB73B716E867228B3C2BE8968F">
  <Source>TRUSTED_LIST</Source>
  <CertificateChain>
    <Info/>
    <Revocation>
      <Info/>
      <Revocations>
        <Info>
          <Message Id="0">No CRL info found!</Message>
        </Info>
      </Revocations>
    </Info>
  </CertificateChain>
</ChainItem>

<Info/>
</Revocation>
</Revocations>
</Info>
</Certificate>

<Certificate Id="80AC3529308758A0AFE7F70DD389130C8E1E7BEFFDC96477356AD2A9E003AD2B">
  ...
</Certificate>

<Certificate Id="702DD5C1A093CF0A9D71FADD9BF9A7C5B57D89FB73B716E867228B3C2BE8968F">
  ...
</Certificate>

<Certificate Id="EE3C22E6087BFEC213709AD3E7F2DDA9CE9D19CE238DCA81A6433E9070A9FBE">
  ...
</Certificate>

<Certificate Id="CB217219BADFC13B4FEA3EFA43882E9FEC49E542DCDBA83428DC6854499A35F">
  ...
</Certificate>
</UsedCertificates>

<TrustedLists>
  <TrustedList>
    <CountryCode>BE</CountryCode>
    <Url>https://tsl.belgium.be/tsl-be.xml</Url>
    <SequenceNumber>36</SequenceNumber>
    <Version>5</Version>
    <LastLoading>2018-02-23T05:15:00</LastLoading>
    <IssueDate>2018-02-08T00:00:00</IssueDate>
    <NextUpdate>2018-07-30T00:00:00</NextUpdate>
    <WellSigned>true</WellSigned>
  </TrustedList>
</TrustedLists>

<TrustedLists>
  <TrustedList>
    <CityCode>EU</CityCode>
  </TrustedList>
</TrustedLists>
Customised Validation Policy

The validation process may be driven by a set of constraints that are contained in the XML file constraint.xml.

**constraint.xml (default policy is provided in validation-policy module)**

```xml
<ConstraintsParameters Name="QES AdESQC TL based" xmlns="http://dss.esig.europa.eu/validation/policy">
    <Description>Validate electronic signatures and indicates whether they are Advanced electronic Signatures (AdES), AdES supported by a Qualified Certificate (AdES/QC) or a Qualified electronic Signature (QES). All certificates and their related chains supporting the signatures are validated against the EU Member State Trusted Lists (this includes signer’s certificate and certificates used to validate certificate validity status services - CRLs, OCSP, and time-stamps).
    </Description>
    <ContainerConstraints>
        <AcceptableContainerTypes Level="FAIL">
            <Id>ASiC-S</Id>
            <Id>ASiC-E</Id>
        </AcceptableContainerTypes>
        <!--        <ZipCommentPresent Level="WARN" /> -->
        <!--        <AcceptableZipComment Level="WARN"> -->
        <!--            <Id>mimetype=application/vnd.etsi.asic-s+zip</Id> -->
        <!--            <Id>mimetype=application/vnd.etsi.asic-e+zip</Id> -->
        <!--        </AcceptableZipComment> -->
        <MimeTypeFilePresent Level="FAIL"/>
        <AcceptableMimeTypeFileContent Level="WARN">
            <Id>application/vnd.etsi.asic-s+zip</Id>
            <Id>application/vnd.etsi.asic-e+zip</Id>
        </AcceptableMimeTypeFileContent>
        <ManifestFilePresent Level="FAIL"/>
        <AllFilesSigned Level="WARN"/>
    </ContainerConstraints>
    <SignatureConstraints>
        <AcceptablePolicies Level="FAIL">
            <Id>ANY_POLICY</Id>
            <Id>NO_POLICY</Id>
        </AcceptablePolicies>
    </SignatureConstraints>
</ConstraintsParameters>
```
<PolicyAvailable Level="FAIL" />
<PolicyHashMatch Level="FAIL" />
<AcceptableFormats Level="FAIL">
  <Id>*</Id>
</AcceptableFormats>
<BasicSignatureConstraints>
  <ReferenceDataExistence Level="FAIL" />
  <ReferenceDataIntact Level="FAIL" />
  <SignatureIntact Level="FAIL" />
  <ProspectiveCertificateChain Level="FAIL" />
</BasicSignatureConstraints>
<!--            <TrustedServiceTypeIdentifier Level="WARN"> -->
<!--                <Id>http://uri.etsi.org/TrstSvc/Svctype/CA/QC</Id> -->
<!--            </TrustedServiceTypeIdentifier> -->
<!--            <TrustedServiceStatus Level="FAIL"> -->
<!--                <Id>http://uri.etsi.org/TrstSvc/TrustedList/Svcstatus/undersupervision</Id> -->
<!--                <Id>http://uri.etsi.org/TrstSvc/TrustedList/Svcstatus/accredited</Id> -->
<!--                <Id>http://uri.etsi.org/TrstSvc/TrustedList/Svcstatus/supervisionincessation</Id> -->
<!--                <Id>http://uri.etsi.org/TrstSvc/TrustedList/Svcstatus/granted</Id> -->
<!--                <Id>http://uri.etsi.org/TrstSvc/TrustedList/Svcstatus/withdrawn</Id> -->
<!--            </TrustedServiceStatus> -->
<SigningCertificate>
  <Recognition Level="FAIL" />
  <Signature Level="FAIL" />
  <NotExpired Level="FAIL" />
  <AuthorityInfoAccessPresent Level="WARN" />
  <RevocationInfoAccessPresent Level="WARN" />
  <RevocationDataAvailable Level="FAIL" />
  <RevocationDataNextUpdatePresent Level="WARN" />
  <RevocationDataFreshness Level="WARN" />
  <KeyUsage Level="WARN">
    <Id>nonRepudiation</Id>
  </KeyUsage>
  <SerialNumberPresent Level="WARN" />
  <NotRevoked Level="FAIL" />
  <NotOnHold Level="FAIL" />
  <NotSelfSigned Level="WARN" />
  <Qualification Level="WARN" /> -->
  <SupportedByQSCD Level="WARN" /> -->
  <IssuedToNaturalPerson Level="INFORM" /> -->
  <IssuedToLegalPerson Level="INFORM" /> -->
  <UsePseudonym Level="INFORM" />
  <Cryptographic Level="FAIL">
    <AcceptableEncryptionAlgo>
      <Algo>RSA</Algo>
      <Algo>DSA</Algo>
      <Algo>ECDSA</Algo>
    </AcceptableEncryptionAlgo>
  </Cryptographic>
</SigningCertificate>
<AcceptableEncryptionAlgo>
  <MiniPublicKeySize>
    <Algo Size="128">DSA</Algo>
    <Algo Size="1024">RSA</Algo>
    <Algo Size="192">ECDSA</Algo>
  </MiniPublicKeySize>
</AcceptableEncryptionAlgo>

<AcceptableDigestAlgo>
  <Algo>SHA1</Algo>
  <Algo>SHA224</Algo>
  <Algo>SHA256</Algo>
  <Algo>SHA384</Algo>
  <Algo>SHA512</Algo>
  <Algo>SHA3-224</Algo>
  <Algo>SHA3-256</Algo>
  <Algo>SHA3-384</Algo>
  <Algo>SHA3-512</Algo>
  <Algo>RIPEMD160</Algo>
</AcceptableDigestAlgo>

</Cryptographic>
</SigningCertificate>

<CACertificate>
  <Signature Level="FAIL" />
  <NotExpired Level="FAIL" />
  <RevocationDataAvailable Level="FAIL" />
  <RevocationDataNextUpdatePresent Level="WARN" />
  <RevocationDataFreshness Level="WARN" />
  <NotRevoked Level="FAIL" />
  <NotOnHold Level="FAIL" />
  <Cryptographic Level="FAIL">
    <AcceptableEncryptionAlgo>
      <Algo>RSA</Algo>
      <Algo>DSA</Algo>
      <Algo>ECDSA</Algo>
    </AcceptableEncryptionAlgo>
    <MiniPublicKeySize>
      <Algo Size="128">DSA</Algo>
      <Algo Size="1024">RSA</Algo>
      <Algo Size="192">ECDSA</Algo>
    </MiniPublicKeySize>
    <AcceptableDigestAlgo>
      <Algo>SHA1</Algo>
      <Algo>SHA224</Algo>
      <Algo>SHA256</Algo>
      <Algo>SHA384</Algo>
      <Algo>SHA512</Algo>
      <Algo>SHA3-224</Algo>
      <Algo>SHA3-256</Algo>
      <Algo>SHA3-384</Algo>
      <Algo>SHA3-512</Algo>
      <Algo>RIPEMD160</Algo>
    </AcceptableDigestAlgo>
  </Cryptographic>
</CACertificate>
Cryptographic
</CACertificate>

<Cryptographic Level="FAIL">
    <AcceptableEncryptionAlgo>
        <Algo>RSA</Algo>
        <Algo>DSA</Algo>
        <Algo>ECDSA</Algo>
    </AcceptableEncryptionAlgo>
    <MiniPublicKeySize>
        <Algo Size="128">DSA</Algo>
        <Algo Size="1024">RSA</Algo>
        <Algo Size="192">ECDSA</Algo>
    </MiniPublicKeySize>
    <AcceptableDigestAlgo>
        <Algo>SHA1</Algo>
        <Algo>SHA224</Algo>
        <Algo>SHA256</Algo>
        <Algo>SHA384</Algo>
        <Algo>SHA512</Algo>
        <Algo>RIPEMD160</Algo>
    </AcceptableDigestAlgo>
</Cryptographic>
</BasicSignatureConstraints>

<SignedAttributes>
    <SigningCertificatePresent Level="FAIL" />
    <SigningCertificateSigned Level="FAIL" />
    <CertDigestPresent Level="FAIL" />
    <CertDigestMatch Level="FAIL" />
    <IssuerSerialMatch Level="WARN" />
    <SigningTime Level="FAIL" />
    <!--
        <ContentType Level="FAIL" value="1.2.840.113549.1.7.1" />
        <ContentHints Level="FAIL" value="*" />
        <CommitmentTypeIndication Level="FAIL">
            <Id>1.2.840.113549.1.9.16.6.1</Id>
            <Id>1.2.840.113549.1.9.16.6.4</Id>
            <Id>1.2.840.113549.1.9.16.6.5</Id>
            <Id>1.2.840.113549.1.9.16.6.6</Id>
        </CommitmentTypeIndication>
        <SignerLocation Level="FAIL" />
        <ContentTimeStamp Level="FAIL" />
    -->
</SignedAttributes>

<UnsignedAttributes>
    <!--
        <CounterSignature Level="IGNORE" /> check presence -->
</UnsignedAttributes>
</SignatureConstraints>
<Timestamp>
    <TimestampDelay Level="FAIL" Unit="DAYS" Value="0" />
</Timestamp>
<MessageImprintDataFound Level="FAIL" />
<MessageImprintDataIntact Level="FAIL" />
<RevocationTimeAgainstBestSignatureTime Level="FAIL" />
<BestSignatureTimeBeforeIssuanceDateOfSigningCertificate Level="FAIL" />
<SigningCertificateValidityAtBestSignatureTime Level="FAIL" />
<AlgorithmReliableAtBestSignatureTime Level="FAIL" />
<Coherence Level="WARN" />
<BASICSignatureConstraints>
  <ReferenceDataExistence Level="FAIL" />
  <ReferenceDataIntact Level="FAIL" />
  <SignatureIntact Level="FAIL" />
  <ProspectiveCertificateChain Level="WARN" />
</BASICSignatureConstraints>
<SigningCertificate>
  <Recognition Level="FAIL" />
  <Signature Level="FAIL" />
  <NotExpired Level="FAIL" />
  <RevocationDataAvailable Level="FAIL" />
  <RevocationDataNextUpdatePresent Level="WARN" />
  <RevocationDataFreshness Level="WARN" />
  <NotRevoked Level="FAIL" />
  <NotOnHold Level="FAIL" />
  <NotSelfSigned Level="WARN" />
  <CRYPTOGRAPHIC Level="FAIL">
    <AcceptableEncryptionAlgo>
      <Algo>RSA</Algo>
      <Algo>DSA</Algo>
      <Algo>ECDSA</Algo>
    </AcceptableEncryptionAlgo>
    <MiniPublicKeySize>
      <Algo Size="128">DSA</Algo>
      <Algo Size="1024">RSA</Algo>
      <Algo Size="192">ECDSA</Algo>
    </MiniPublicKeySize>
    <AcceptableDigestAlgo>
      <Algo>SHA1</Algo>
      <Algo>SHA224</Algo>
      <Algo>SHA256</Algo>
      <Algo>SHA384</Algo>
      <Algo>SHA512</Algo>
      <Algo>SHA3-224</Algo>
      <Algo>SHA3-256</Algo>
      <Algo>SHA3-384</Algo>
      <Algo>SHA3-512</Algo>
      <Algo>RIPEMD160</Algo>
    </AcceptableDigestAlgo>
  </CRYPTOGRAPHIC>
</SigningCertificate>
<CACertificate>
  <Signature Level="FAIL" />
  <NotExpired Level="FAIL" />
  <RevocationDataAvailable Level="WARN" />
</CACertificate>
<RevocationDataNextUpdatePresent Level="WARN" />
<RevocationDataFreshness Level="WARN" />
<NotRevoked Level="FAIL" />
<NotOnHold Level="FAIL" />
<Cryptographic Level="FAIL">
  <AcceptableEncryptionAlgo>
    <Algo>RSA</Algo>
    <Algo>DSA</Algo>
    <Algo>ECDSA</Algo>
  </AcceptableEncryptionAlgo>
  <MiniPublicKeySize>
    <Algo Size="128">DSA</Algo>
    <Algo Size="1024">RSA</Algo>
    <Algo Size="192">ECDSA</Algo>
  </MiniPublicKeySize>
  <AcceptableDigestAlgo>
    <Algo>SHA1</Algo>
    <Algo>SHA224</Algo>
    <Algo>SHA256</Algo>
    <Algo>SHA384</Algo>
    <Algo>SHA512</Algo>
    <Algo>SHA3-224</Algo>
    <Algo>SHA3-256</Algo>
    <Algo>SHA3-384</Algo>
    <Algo>SHA3-512</Algo>
    <Algo>RIPEMD160</Algo>
  </AcceptableDigestAlgo>
</Cryptographic>
</CACertificate>
<Cryptographic Level="FAIL">
  <AcceptableEncryptionAlgo>
    <Algo>RSA</Algo>
    <Algo>DSA</Algo>
    <Algo>ECDSA</Algo>
  </AcceptableEncryptionAlgo>
  <MiniPublicKeySize>
    <Algo Size="128">DSA</Algo>
    <Algo Size="1024">RSA</Algo>
    <Algo Size="192">ECDSA</Algo>
  </MiniPublicKeySize>
  <AcceptableDigestAlgo>
    <Algo>SHA1</Algo>
    <Algo>SHA224</Algo>
    <Algo>SHA256</Algo>
    <Algo>SHA384</Algo>
    <Algo>SHA512</Algo>
    <Algo>SHA3-224</Algo>
    <Algo>SHA3-256</Algo>
    <Algo>SHA3-384</Algo>
    <Algo>SHA3-512</Algo>
    <Algo>RIPEMD160</Algo>
  </AcceptableDigestAlgo>
</Cryptographic>
<NotRevoked Level="FAIL" />
<NotOnHold Level="FAIL" />
<Cryptographic Level="FAIL">
  <AcceptableEncryptionAlgo>
    <Algo>RSA</Algo>
    <Algo>DSA</Algo>
    <Algo>ECDSA</Algo>
  </AcceptableEncryptionAlgo>
  <MiniPublicKeySize>
    <Algo Size="128">DSA</Algo>
    <Algo Size="1024">RSA</Algo>
    <Algo Size="192">ECDSA</Algo>
  </MiniPublicKeySize>
  <AcceptableDigestAlgo>
    <Algo>SHA1</Algo>
    <Algo>SHA224</Algo>
    <Algo>SHA256</Algo>
    <Algo>SHA384</Algo>
    <Algo>SHA512</Algo>
    <Algo>SHA3-224</Algo>
    <Algo>SHA3-256</Algo>
    <Algo>SHA3-384</Algo>
    <Algo>SHA3-512</Algo>
    <Algo>RIPEMD160</Algo>
  </AcceptableDigestAlgo>
</Cryptographic>
</CACertificate>
<Cryptographic Level="FAIL">
  <AcceptableEncryptionAlgo>
    <Algo>RSA</Algo>
    <Algo>DSA</Algo>
    <Algo>ECDSA</Algo>
  </AcceptableEncryptionAlgo>
  <MiniPublicKeySize>
    <Algo Size="128">DSA</Algo>
    <Algo Size="1024">RSA</Algo>
    <Algo Size="192">ECDSA</Algo>
  </MiniPublicKeySize>
  <AcceptableDigestAlgo>
    <Algo>SHA1</Algo>
    <Algo>SHA224</Algo>
    <Algo>SHA256</Algo>
    <Algo>SHA384</Algo>
    <Algo>SHA512</Algo>
    <Algo>SHA3-224</Algo>
    <Algo>SHA3-256</Algo>
    <Algo>SHA3-384</Algo>
    <Algo>SHA3-512</Algo>
    <Algo>RIPEMD160</Algo>
  </AcceptableDigestAlgo>
</Cryptographic>
CAdES signature (CMS)

To familiarise yourself with this type of signature it is advisable to read the following document:

- CAdES Specifications (cf. [R02])

To implement this form of signature you can use the XAdES examples. You only need to instantiate the CAdES object service and change the SignatureLevel parameter value. Below is an example of the CAdES-Baseline-B signature:
// Preparing parameters for the CAdES signature
cAdESSignatureParameters parameters = new CAdESSignatureParameters();
// We choose the level of the signature (-B, -T, -LT, -LTA).
parameters.setSignatureLevel(SignatureLevel.CAdES_BASELINE_B);
// We choose the type of the signature packaging (ENVELOPING, DETACHED).
parameters.setSignaturePackaging(SignaturePackaging.ENVELOPING);
// We set the digest algorithm to use with the signature algorithm. You must use the // same parameter when you invoke the method sign on the token. The default value is // SHA256
parameters.setDigestAlgorithm(DigestAlgorithm.SHA256);

// We set the signing certificate
parameters.setSigningCertificate(privateKey.getCertificate());
// We set the certificate chain
parameters.setCertificateChain(privateKey.getCertificateChain());

// Create common certificate verifier
CommonCertificateVerifier commonCertificateVerifier = new CommonCertificateVerifier();
// Create CAdES xadesService for signature
CAdESService service = new CAdESService(commonCertificateVerifier);

// Get the SignedInfo segment that need to be signed.
ToBeSigned dataToSign = service.getDataToSign(toSignDocument, parameters);

// This function obtains the signature value for signed information using the // private key and specified algorithm
DigestAlgorithm digestAlgorithm = parameters.getDigestAlgorithm();
SignatureValue signatureValue = signingToken.sign(dataToSign, digestAlgorithm, privateKey);

// We invoke the xadesService to sign the document with the signature value obtained in // the previous step.
DSSDocument signedDocument = service.signDocument(toSignDocument, parameters, signatureValue);

**PAdES signature (PDF)**

The standard ISO 32000-1 (cf. [R05]) allows defining a file format for portable electronic documents. It is based on PDF 1.7 of Adobe Systems. Concerning the digital signature it supports three operations:

- Adding a digital signature to a document,
- Providing a placeholder field for signatures,
- Checking signatures for validity.
PAdES defines eight different profiles to be used with advanced electronic signature in the meaning of European Union Directive 1999/93/EC (cf. [R06]):

- PAdES Basic - PDF signature as specified in ISO 32000-1 (cf. [R05]). The profile is specified in ETSI EN 319 142 (cf. [R03]).
- PAdES-BES Profile - based upon CAdES-BES as specified in ETSI EN 319 122 (cf. [R02]) with the option of a signature time-stamp (CAdES-T).
- PAdES-EPES profile - based upon CAdES-EPES as specified in ETSI EN 319 122 (cf. [R02]). This profile is the same as the PAdES - BES with the addition of a signature policy identifier and optionally a commitment type indication.
- PAdES-LTV Profile - This profile supports the long term validation of PDF Signatures and can be used in conjunction with the above-mentioned profiles.
- Four other PAdES profiles for XML Content.

To familiarise yourself with this type of signature it is advisable to read the documents referenced above.

Below is an example of code to perform a PAdES-BASELINE-B type signature:
// Preparing parameters for the PAdES signature
PAdESSignatureParameters parameters = new PAdESSignatureParameters();
// We choose the level of the signature (-B, -T, -LT, -LTA).
parameters.setSignatureLevel(SignatureLevel.PAdES_BASELINE_B);
// We set the digest algorithm to use with the signature algorithm. You must use the
// same parameter when you invoke the method sign on the token. The default value is
// SHA256
parameters.setDigestAlgorithm(DigestAlgorithm.SHA256);

// We set the signing certificate
parameters.setSigningCertificate(privateKey.getCertificate());
// We set the certificate chain
parameters.setCertificateChain(privateKey.getCertificateChain());

// Create common certificate verifier
CommonCertificateVerifier commonCertificateVerifier = new CommonCertificateVerifier();
// Create PAdESService for signature
PAdESService service = new PAdESService(commonCertificateVerifier);

// Get the SignedInfo segment that need to be signed.
ToBeSigned dataToSign = service.getDataToSign(toSignDocument, parameters);

// This function obtains the signature value for signed information using the
// private key and specified algorithm
DigestAlgorithm digestAlgorithm = parameters.getDigestAlgorithm();
SignatureValue signatureValue = signingToken.sign(dataToSign, digestAlgorithm, privateKey);

// We invoke the xadesService to sign the document with the signature value obtained in
// the previous step.
DSSDocument signedDocument = service.signDocument(toSignDocument, parameters, signatureValue);

To add the timestamp to the signature (PAdES-T or LTA), please provide TSP source to the service.

To create PAdES-BASELINE-B level with additional options: signature policy identifier and
optionally a commitment type indication, please observe the following example in code 5.

All these parameters are optional.

- **SignaturePolicyOID** : The string representation of the OID of the signature policy to use when
  signing.
- **SignaturePolicyHashValue** : The value of the hash of the signature policy, computed the same
  way as in clause 5.2.9 of CAdES (ETSI EN 319 122 (cf. [R02])).
- **SignaturePolicyHashAlgorithm** : The hash function used to compute the value of the
  SignaturePolicyHashValue entry. Entries must be represented the same way as in table 257 of
ISO 32000-1 (cf. [R05]).

- **SignaturePolicyCommitmentType**: If the SignaturePolicyOID is present, this array defines the commitment types that can be used within the signature policy. An empty string can be used to indicate the default commitment type.

If the SignaturePolicyOID is absent, the three other fields defined above will be ignored. If the SignaturePolicyOID is present but the SignaturePolicyCommitmentType is absent, all commitments defined by the signature policy will be used.

The extension of a signature of the level PAdES-BASELINE-B up to PAdES-BASELINE-LTA profile will add the following features:

- Addition of validation data to an existing PDF document which may be used to validate earlier signatures within the document (including PDF signatures and time-stamp signatures).
- Addition of a document time-stamp which protects the existing document and any validation data.
- Further validation data and document time-stamp may be added to a document over time to maintain its authenticity and integrity.

**PAdES Visible Signature**

The framework also allows to create PDF files with visible signature as specified in ETSI EN 319 142 (cf. [R03]). In the SignatureParameters object, there’s a special attribute named ImageParameters. This parameter let you custom the visual signature (with text, with image or with image and text). Below is an example of code to perform a PADES-BASELINE-B type signature with a visible signature:
Add a visible signature to a PDF document

```java
// Preparing parameters for the PAdEs signature
PAdESSignatureParameters parameters = new PAdESSignatureParameters();
// We choose the level of the signature (-B, -T, -LT, -LTA).
parameters.setSignatureLevel(SignatureLevel.PAdES_BASELINE_B);

// We set the signing certificate
parameters.setSigningCertificate(privateKey.getCertificate());
// We set the certificate chain
parameters.setCertificateChain(privateKey.getCertificateChain());

// Initialize visual signature
SignatureImageParameters imageParameters = new SignatureImageParameters();
// the origin is the left and top corner of the page
imageParameters.setxAxis(200);
imageParameters.setyAxis(500);

// Initialize text to generate for visual signature
SignatureImageTextParameters textParameters = new SignatureImageTextParameters();
textParameters.setFont(new Font("serif", Font.PLAIN, 14));
textParameters textColor(Color.BLUE);
textParameters setText("My visual signature");
imageParameters setTextParameters(textParameters);
parameters.setSignatureImageParameters(imageParameters);

// Create common certificate verifier
CommonCertificateVerifier commonCertificateVerifier = new CommonCertificateVerifier();
// Create PAdESService for signature
PAdESService service = new PAdESService(commonCertificateVerifier);

// Get the SignedInfo segment that need to be signed.
ToBeSigned dataToSign = service.getDataToSign(toSignDocument, parameters);

// This function obtains the signature value for signed information using the
// private key and specified algorithm
DigestAlgorithm digestAlgorithm = parameters.getDigestAlgorithm();
SignatureValue signatureValue = signingToken.sign(dataToSign, digestAlgorithm, privateKey);

// We invoke the xadesService to sign the document with the signature value obtained in
// the previous step.
DSSDocument signedDocument = service.signDocument(toSignDocument, parameters, signatureValue);
```

Additionally, DSS also allows to insert a visible signature in an existing field:
parameters.setSignatureFieldId("field-id");

**ASiC signature (containers)**

When creating a digital signature, the user must choose between different packaging elements, namely enveloping, enveloped or detached. This choice is not obvious, because in one case the signature will alter the signed document and in the other case it is possible to lose the association between the signed document and its signature. That's where the standard ETSI EN 319 162 (cf. [R04]) offers a standardized use of container forms to establish a common way for associating data objects with advanced signatures or time-stamp tokens.

A number of application environments use ZIP based container formats to package sets of files together with meta-information. ASiC technical specification is designed to operate with a range of such ZIP based application environments. Rather than enforcing a single packaging structure, ASiC describes how these package formats can be used to associate advanced electronic signatures with any data objects.

The standard defines two types of containers; the first (ASiC-S) allows you to associate one or more signatures with a single data element. In this case the structure of the signature can be based (in a general way) on a single CAdES signature or on multiple XAdES signatures or finally on a single TST; the second is an extended container (ASiC-E) that includes multiple data objects. Each data object may be signed by one or more signatures which structure is similar to ASiC-S. This second type of container is compatible with OCF, UCF and ODF formats.

For the moment the DSS framework has some restrictions on the containers you can generate, depending on the input file. If the input file is already an ASiC container, the output container must be the same type of container based on the same type of signature. If the input is any other file, the output does not have any restriction.

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASiC-S CAdES</td>
<td>ASiC-S CAdES</td>
</tr>
<tr>
<td>ASiC-S XAdES</td>
<td>ASiC-S XAdES</td>
</tr>
<tr>
<td>ASiC-E CAdES</td>
<td>ASiC-E CAdES</td>
</tr>
<tr>
<td>ASiC-E XAdES</td>
<td>ASiC-E XAdES</td>
</tr>
<tr>
<td>Binary</td>
<td>ASiC-S CAdES, ASiC-S XAdES, ASiC-E CAdES, ASiC-E XAdES</td>
</tr>
</tbody>
</table>

This is an example of the source code for signing a document using ASiCS-S based on XAdES-B:
// Preparing parameters for the AsicS signature
ASiCWithXAdESSignatureParameters parameters = new ASiCWithXAdESSignatureParameters();
// We choose the level of the signature (-B, -T, -LT, LTA).
parameters.setSignatureLevel(SignatureLevel.XAdES_BASELINE_B);
// We choose the container type (ASiC-S or ASiC-E)
parameters.asiC().setContainerType(ASiCContainerType.ASiC_S);

// We set the digest algorithm to use with the signature algorithm. You must use the
// same parameter when you invoke the method sign on the token. The default value is
// SHA256
parameters.setDigestAlgorithm(DigestAlgorithm.SHA256);

// We set the signing certificate
parameters.setSigningCertificate(privateKey.getCertificate());
// We set the certificate chain
parameters.setCertificateChain(privateKey.getCertificateChain());

// Create common certificate verifier
CommonCertificateVerifier commonCertificateVerifier = new CommonCertificateVerifier();
// Create ASiC service for signature
ASiCWithXAdESService service = new ASiCWithXAdESService(commonCertificateVerifier);

// Get the SignedInfo segment that need to be signed.
ToBeSigned dataToSign = service.getDataToSign(toSignDocument, parameters);

// This function obtains the signature value for signed information using the
// private key and specified algorithm
DigestAlgorithm digestAlgorithm = parameters.getDigestAlgorithm();
SignatureValue signatureValue = signingToken.sign(dataToSign, digestAlgorithm, privateKey);

// We invoke the xadesService to sign the document with the signature value obtained
// in
// the previous step.
DSSDocument signedDocument = service.signDocument(toSignDocument, parameters, signatureValue);

This is another example of the source code for signing multiple documents using ASiCS-E based on CADES:
Sign multiple files within an ASiC-E container

// Preparing the documents to be embedded in the container and signed
List<DSSDocument> documentsToBeSigned = new ArrayList<DSSDocument>();
documentsToBeSigned.add(new FileDocument("src/main/resources/hello-world.pdf"));
documentsToBeSigned.add(new FileDocument("src/main/resources/xml_example.xml"));

// Preparing parameters for the ASiC-E signature
ASiCWithCAdESSignatureParameters parameters = new ASiCWithCAdESSignatureParameters();

// We choose the level of the signature (-B, -T, -LT or -LTA).
parameters.setSignatureLevel(SignatureLevel.CAdES_BASELINE_B);
// We choose the container type (ASiC-S or ASiC-E)
parameters.aSiC().setContainerType(ASiCContainerType.ASiC_E);

// We set the digest algorithm to use with the signature algorithm. You
// must use the
// same parameter when you invoke the method sign on the token. The
// default value is
// SHA256
parameters.setDigestAlgorithm(DigestAlgorithm.SHA256);

// We set the signing certificate
parameters.setSigningCertificate(privateKey.getCertificate());
// We set the certificate chain
parameters.setCertificateChain(privateKey.getCertificateChain());

// Create common certificate verifier
CommonCertificateVerifier commonCertificateVerifier = new CommonCertificateVerifier();
// Create ASiC service for signature
ASiCWithCAdESService service = new ASiCWithCAdESService(commonCertificateVerifier);

// Get the SignedInfo segment that need to be signed.
ToBeSigned dataToSign = service.getDataToSign(documentsToBeSigned, parameters);

// This function obtains the signature value for signed information
// using the
// private key and specified algorithm
DigestAlgorithm digestAlgorithm = parameters.getDigestAlgorithm();
SignatureValue signatureValue = signingToken.sign(dataToSign, digestAlgorithm, privateKey);

// We invoke the xadesService to sign the document with the signature
// value obtained in
// the previous step.
DSSDocument signedDocument = service.signDocument(documentsToBeSigned, parameters, signatureValue);

Please note that you need to pass only few parameters to the service. Other parameters, although
are positioned, will be overwritten by the internal implementation of the service. Therefore, the
obtained signature is always based on CAdES and of DETACHED packaging.

It is also possible with the framework DSS to make an extension of an ASiC container to the level XAdES-BASELINE-T or -LT.

Available implementations of DSSDocument

DSS allows to create different kinds of DSSDocument:

- **InMemoryDocument**: fully loads in memory. This type of DSSDocument can be instantiated with an array of bytes, an InputStream,...
- **FileDocument**: refers an existing File
- **DigestDocument**: only contains pre-computed digest values for a given document. That allows to avoid sending the full document (detached signatures).

```java
DigestDocument

// Firstly, we load a basic DSSDocument (FileDocument or InMemoryDocument)  

// After that, we create a DigestDocument  
DigestDocument digestDocument = new DigestDocument();  
digestDocument.setName(fileDocument.getName());

// We add needed digest value(s). Eg: for a SHA-256 based signature  
digestDocument.addDigest(DigestAlgorithm.SHA256, fileDocument.getDigest(DigestAlgorithm.SHA256));
```

Management of signature tokens

The DSS framework is able to create signatures from PKCS#11, PKCS#12 and MS CAPI. Java 6 is inherently capable of communicating with these kinds of KeyStores. To be independent of the signing media, DSS framework uses an interface named SignatureTokenConnection to manage different implementations of the signing process. The base implementation is able to sign a stream of the data in one step. That means that all the data to be signed needs to be sent to the SSCD. This is the case for MS CAPI. As to the PKCS#11 and PKCS#12, which give to the developer a finer control in the signature operation, the DSS framework implements the AsyncSignatureTokenConnection abstract class that permits to execute the digest operation and signature operation in two different threads or even two different hardwares.

This design permits also other card providers/adopters to create own implementations. For example, this can be used for a direct connection to the Smartcard through Java 6 PC/SC.

PKCS#11

PKCS#11 is widely used to access smart cards and HSMs. Most commercial software uses PKCS#11
to access the signature key of the CA or to enrol user certificates. In the DSS framework, this standard is encapsulated in the class Pkcs11SignatureToken.

**Pkcs11SignatureToken usage**

```java
try (Pkcs11SignatureToken token = new Pkcs11SignatureToken("C:\Windows\System32\beidpkcs11.dll")) {
    List<DSSPrivateKeyEntry> keys = token.getKeys();
    for (DSSPrivateKeyEntry entry : keys) {
        System.out.println(entry.getCertificate().getCertificate());
    }

    ToBeSigned toBeSigned = new ToBeSigned("Hello world".getBytes());
    SignatureValue signatureValue = token.sign(toBeSigned, DigestAlgorithm.SHA256, keys.get(0));

    System.out.println("Signature value : " + Utils.toBase64(signatureValue.getValue()));
}
```

**PKCS#12**

This standard defines a file format commonly used to store the private key and corresponding public key certificate protecting them by password.

In order to use this format with the DSS framework you have to go through the class Pkcs12SignatureToken.

**Pkcs12SignatureToken usage**

```java
try (Pkcs12SignatureToken token = new Pkcs12SignatureToken("src/main/resources/user_a_rsa.p12", new PasswordProtection("password".toCharArray())) {
    List<DSSPrivateKeyEntry> keys = token.getKeys();
    for (DSSPrivateKeyEntry entry : keys) {
        System.out.println(entry.getCertificate().getCertificate());
    }

    ToBeSigned toBeSigned = new ToBeSigned("Hello world".getBytes());
    SignatureValue signatureValue = token.sign(toBeSigned, DigestAlgorithm.SHA256, keys.get(0));

    System.out.println("Signature value : " + Utils.toBase64(signatureValue.getValue()));
}
```
If the middleware for communicating with an SSDC provides a CSP based on MS CAPI specification, then to sign the documents you can use MSCAPISignatureToken class.

MSCAPISignatureToken usage

```java
try (MSCAPISignatureToken token = new MSCAPISignatureToken()) {
    List<DSSPrivateKeyEntry> keys = token.getKeys();
    for (DSSPrivateKeyEntry entry : keys) {
        System.out.println(entry.getCertificate().getCertificate());
    }
    ToBeSigned toBeSigned = new ToBeSigned("Hello world".getBytes());
    SignatureValue signatureValue = token.sign(toBeSigned, DigestAlgorithm.SHA256, keys.get(0));
    System.out.println("Signature value : " + Utils.toBase64(signatureValue.getValue()));
}
```

Other Implementations

As you can see, it is easy to add another implementation of the SignatureTokenConnection, thus enabling the framework to use other API than the provided three (PKCS#11, PKCS#12 and MS CAPI). For example, it is likely that in the future PC/SC will be the preferred way of accessing a Smartcard. Although PKCS#11 is currently the most used API, DSS framework is extensible and can use PC/SC. For our design example we propose to use PC/SC to communicate with the Smartcard.

Management of certificates sources

The validation of a certificate requires the access to some other certificates from multiple sources like trusted lists, trust store, the signature itself: certificates can be contained inside or any other source. Within the framework, an X509 certificate is modelled through the class:

- eu.europa.esig.dss.x509.CertificateToken

This encapsulation helps make certificate handling more suited to the needs of the validation in the context of trust. Each certificate is unambiguously identified by its issuer DN and serial number. The framework associates a unique internal identifier to each certificate but this identifier is not calculated on the data contained in the certificate and therefore varies from one application to another. However, it is independent of its source. It allows to easily comparing certificates issued by different sources. Certificate tokens are grouped into pools. A certificate token can be declared in several pools. The class that models a pool is called:

This class allows keeping only one occurrence of the certificate in the given context (i.e. validation).

The CertificateSource interface provides abstraction for accessing a certificate, regardless of the source. However, each source has its own type:

- eu.europa.esig.dss.x509.CertificateSourceType

This information is used, for example, to distinguish between the certificate from a trusted source and the others. A source has one and only one type, but a certificate token can be found in multiple sources. The DSS framework supplies some standard implementations, but also gives the possibility to implement owner solutions. Among the standard solutions you can find:


This is the superclass of almost of the certificate sources. It implements the common method CommonCertificateSource#get returns the list of CertificateToken(s) corresponding to the given subject distinguished name. Note that the content of the encapsulated certificates pool can be different from the content of the source. Only CertificateToken(s) present in the source are taken into account. It exposes also the method CommonCertificateSource#addCertificate which gives the possibility to add manually any X509Certificate as a part of this source and as a part of the encapsulated pool. If the certificate is already present in the pool its source type is associated to the token.

- eu.europa.esig.dss.x509.SignatureCertificateSource

Some certificate sources are based on data encapsulated within the signature. That means that the set of certificates is available and the software only needs to find the certificate using its subject name. This class adds also new methods to obtain specialized list of certificates contained in the source:

- SignatureCertificateSource#getKeyInfoCertificates
- SignatureCertificateSource#getEncapsulatedCertificates
  - eu.europa.esig.dss.tsl.TrustedListsCertificateSource

Certificates coming from the list of Trusted Lists. This class gives the mechanism to define the set of trusted certificates (trust anchors). They are used in the validation process to decide if the prospective certificate chain has a trust anchor. See chapter 5.2 to know more about EU Trusted Lists.

### Management of CRL and OCSP sources

A CRL is a time-stamped list identifying revoked certificates. It is signed by a Certificate Authority (CA) and made freely available in a public repository. Each revoked certificate is identified in a CRL by its certificate serial number.

The Online Certificate Status Protocol (OCSP) is an Internet protocol used for obtaining the revocation status of an unique X.509 digital certificate.

For every certificate, the validity has to be checked via CRL or OCSP responses. The information...
may originate from different CRLSources or OCSPSources: For easing the usage of such sources, DSS implements a CRLSource and OCSPSource interfaces (which inherit from RevocationSource), which offer a generic, uniform way of accessing CRL and OCSP sources. Furthermore, a caching mechanism can be easily attached to those sources, optimizing the access time to revocation information by reducing network connections to online servers.

The interface CRLSource defines the method which returns CRLToken for the given certificate/issuer certificate couple:

*CRLSource usage*

```java
CRLToken crlToken = crlSource.getRevocationToken(certificateToken, issuerCertificateToken);
```

The interface OCSPSource defines the method which returns OCSPToken for the given certificate/issuer certificate couple:

*OCSPSource usage*

```java
OCSPToken ocspToken = ocspSource.getRevocationToken(certificateToken, issuerCertificateToken);
```

We use these classes during the certificate validation process through "validationContext" object (based on ValidationContext class) which is a "cache" for one validation request that contains every object retrieved so far. This object in turn instantiates a "verifier" based on CSPAndCRLCertificateVerifier class whose role is to fetch revocation data by querying an OCSP server first and then a CRL server if no OCSP response could be retrieved. In general we can distinguish three main sources:

- Offline sources;
- Online sources;
- Sources with the cache mechanism.

**Other implementations of CRL and OCSP Sources**

Such sources find the status of a certificate either from a list stored locally or using the information contained in the advanced signature or online way. Here is the list of sources already implemented in the DSS framework:

- CRL sources
  - JdbcCacheCRLSource: Retrieves information from a JDBC datasource
  - OfflineCRLSource: This class that implements in a generic way the findCrl method that operates on the different CRLs implemented in children classes.
    - ListCRLSource: This source maintains a list of CRLToken.
    - SignatureCRLSource: The advanced signature contains a list of CRL that was needed to validate the signature. This class is a basic skeleton that is able to retrieve the needed
CRL from a list. The child needs to retrieve the list of wrapped CRLs.

- CAdESCRLSource: Retrieves information from a CAdES signature.
- PAdESCRLSource: Retrieves information from a PAdES signature.
- XAdESCRLSource: Retrieves information from a XAdES signature.
- ExternalResourcesCRLSource: A class that can instantiate a list of certificate revocation lists from a directory where should be the individual lists (each individual list file must end with the extension ".crl").

- OnlineCRLSource: This is a representation of an Online CRL repository. This implementation will contact using HTTP protocol the CRL Responder to download the CRLs from the given URI. Note that certificate’s Authority Information Access (AIA) extension is used to find issuer’s resources location like CRT file and/or Online Certificate Status Protocol (OCSP). The URIs of CRL server will be extracted from this property (OID value: 1.3.6.1.5.5.7.48.1.3).

OCSP sources

- OfflineOCSPSource: An abstract class that helps to implement OCSPSource with an already loaded list of OCSPToken. It implements in a generic way the getOCSPResponse method that operates on the different OCSP implementations in children classes.
  - ListOCSPSource: Implements an OCSPSource from a list of OCSPToken.
  - SignatureOCSPSource: The advanced signature contains a list of OCSPResp that was needed to validate the signature. This class is a basic skeleton that is able to retrieve the needed OCSPResp from a list. The children need to retrieve the list of wrapped OCSPResp.
    - CAdESOCSPSource: Retrieves information from a CAdES signature.
    - PAdESOCSPSource: Retrieves information from a PAdES signature.
    - XAdESOCSPSource: Retrieves information from a XAdES signature.
    - ExternalResourcesOCSPSource: A class that can instantiate a list of OCSPToken from a directory where should be the individual DER Encoded X509 certificates files (each individual file must end with the extension ".der").

- OnlineOCSPSource: This is a representation of an Online OCSP repository. This implementation will contact using HTTP protocol the OCSP Responder to retrieve the OCSP response. Note that certificate’s Authority Information Access (AIA) extension is used to find issuer's resources location like CRT file and/or Online Certificate Status Protocol (OCSP). The URIs of OCSP server will be extracted from this property (OID value: 1.3.6.1.5.5.7.48.1).

CertificateVerifier configuration

The CertificateVerifier and its implementation CommonCertificateVerifier determine how DSS accesses to external resources and how it should react in some occasions. This configuration is used in both extension and validation mode.
CertificateVerifier usage

CertificateVerifier cv = new CommonCertificateVerifier();

// This data loader is used to collect certificates from external resources
// (AIA)
cv.setDataLoader(dataLoader);

// This certificate source is used to provide missing intermediate certificates
// (not trusted certificates)
cv.setAdjunctCertSource(adjunctCertSource);

// This certificate source is used to provide trusted certificates (the trust
// anchors where the certificate chain building should stop)
cv.setTrustedCertSource(trustedCertSource);

// The CRL Source to be used for external accesses (can be configured with a
// cache,...)
cv.setCrlSource(crlSource);

// The OCSP Source to be used for external accesses (can be configured with a
// cache,...)
cv.setOcspSource(ocspSource);

// Define the behavior to be followed by DSS in case of revocation checking for
// certificates issued from an unsure source (DSS v 5.4+)
// Default : revocation check is disabled for unsure sources (security reasons)
cv.setCheckRevocationForUntrustedChains(false);

// DSS v 5.4+ : The 3 below configurations concern the extension mode (LT/LTA
// extension)

// DSS throws an exception by default in case of missing revocation data
// Default : true
 cv.setExceptionOnMissingRevocationData(true);

// DSS throws an exception if a TSU certificate chain is not covered with a
// revocation data (timestamp generation time > CRL/OCSP production time).
// Default : false
 cv.setExceptionOnUncoveredPOE(true);

// DSS interrupts by default the extension process if a revoked certificate is
// present
// Default : true
 cv.setExceptionOnRevokedCertificate(true);

// DSS stops the extension process if an invalid timestamp is met
// Default : true
 cv.setExceptionOnInvalidTimestamp(true);
TSP Sources

The Time Stamp Authority by creating time-stamp tokens provides independent and irrefutable proof of time for business transactions, e-documents and digital signatures. The TSA must comply with the IETF RFC 3161 specifications (cf. [R07]). A time-stamp is obtained by sending the digest value of the given data and digest algorithm to the Time Stamp Authority. The returned time-stamp is a signed data that contains the digest value, the identity of the TSA, and the time of stamping. This proves that the given data existed before the time of stamping. The DSS framework proposes TSPSource interface to implement the communication with TSA. The class OnlineTSPSource is the default implementation of TSP using HTTP(S) communication layer. The following bit of Java code illustrates how you might use this class:

```
final String tspServer = "http://tsa.belgium.be/connect";
OnlineTSPSource tspSource = new OnlineTSPSource(tspServer);

final DigestAlgorithm digestAlgorithm = DigestAlgorithm.SHA256;
final byte[] toDigest = "Hello world".getBytes("UTF-8");
final byte[] digestValue = DSSUtils.digest(digestAlgorithm, toDigest);
final TimeStampToken tsr = tspSource.getTimeStampResponse(digestAlgorithm, digestValue);

System.out.println(DSSUtils.toHex(tsr.getEncoded()));
```

Time-stamp policy

A time-stamp policy is a "named set of rules that indicates the applicability of a time-stamp token to a particular community and/or class of application with common security requirements". A TSA may define its own policy which enhances the policy defined in RFC 3628. Such a policy shall incorporate or further constrain the requirements identified in RFC 3628. A time-stamp policy may be defined by the user of times-stamp services.

Composite TSP Source

Sometimes, timestamping servers may encounter interruptions (restart,...). To avoid failing signature extension, DSS allows to configure several TSP Sources. DSS will try source by source until getting an usable timestamp token.
Configuration of a CompositeTSPSource

```java
// Create a map with several TSPSources
Map<String, TSPSource> tspSources = new HashMap<String, TSPSource>();
tspSources.put("Poland", new OnlineTSPSource("http://time.certum.pl/");
tspSources.put("Belgium", new OnlineTSPSource("http://tsa.belgium.be/connect");

// Instantiate a new CompositeTSPSource and set the different sources
CompositeTSPSource tspSource = new CompositeTSPSource();
tspSource.setTspSources(tspSources);

final DigestAlgorithm digestAlgorithm = DigestAlgorithm.SHA256;
final byte[] toDigest = "Hello world".getBytes("UTF-8");
final byte[] digestValue = DSSUtils.digest(digestAlgorithm, toDigest);

// DSS will request the tsp sources (one by one) until getting a valid token.
// If none of them succeed, a DSSException is thrown.
final TimeStampToken tsr = tspSource.getTimeStampResponse(digestAlgorithm, digestValue);

System.out.println(DSSUtils.toHex(tsr.getEncoded()));
```

Supported algorithms

DSS supports several signature algorithms (combination of an encryption algorithm and a digest algorithm). Below, you can find the supported combinations. The support of the algorithms depends of the registered OID (ASN1) or URI (XML).

In the next table, XAdES also applies to ASiC with embedded XAdES signatures and CAdES also concerns PAdES and ASiC with embedded CAdES signatures.

SmartCards/HSMs don't allow to sign with all digest algorithms. Please refer to your SmartCard/HSM provider.

<table>
<thead>
<tr>
<th></th>
<th>SHA-1</th>
<th>SHA-224</th>
<th>SHA-256</th>
<th>SHA-384</th>
<th>SHA-512</th>
<th>SHA3-224</th>
<th>SHA3-256</th>
<th>SHA3-384</th>
<th>SHA3-512</th>
<th>MD2</th>
<th>MD5</th>
<th>RIPE MD160</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>XAdES</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>CAdES</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>RSA-PSS</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>XAdES</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>CAdES</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>ECDSA</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>XAdES</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>
Multi-threading

DSS can be used in multi-threaded environments but some points need to be considered like resources sharing and caching. All operations are stateless and this fact requires to be maintained. Some resources can be shared, others are proper to an operation.

For each provided operation, DSS requires a CertificateVerifier object. This object is responsible to provide certificates and accesses to external resources (AIA, CRL, OCSP,…). At the beginning of all operation, a new internal CertificatePool is created and all available certificates are copied. Throughout the signature/validation process, the CertificatePool content evolves. Certificates are added/updated from the signature, timestamp(s), revocation data,… Revocation data / issuer certificates are collected and added to the certificate. Certificate status are updated to give as much as possible information. For these reasons, integrators need to be careful about the CertificateVerifier configuration.

Resource sharing

The trusted certificates can be shared between multiple threads because these certificates are static. This means they don’t require more analysis. Their status won’t evolve. For these certificates, DSS doesn’t need to collect issuer certificate and/or their revocation data.

In opposition, the adjunct certificates cannot be shared. These certificates concern a specific signature/validation operation. This parameter is used to provide missing certificate(s). When DSS is unable to build the complete certificate path with the provided certificates (as signature parameters or embedded within a signature), it is possible to inject not present certificates. These certificates are not necessarily trusted and may require future “modifications” like revocation data collection,…

Caching

In case of multi-threading usage, we strongly recommend to cache external resources. All external resources can be cached (AIA, CRL, OCSP) to improve performances and to avoid requesting too much time the same resources. FileCacheDataLoader and JdbcCacheCRLSource can help you in this way.
Additional features

Certificate validation

DSS offers the possibility to validate a certificate. For a given certificate, the framework builds a certificate path until a known trust anchor (trusted list, keystore,...), validates each found certificate (OCSP / CRL) and determines its European "qualification".

To determine the certificate qualification, DSS follows the draft standard ETSI TS 119 172-4 (R09). It analyses the certificate properties (QCStatements, Certificate Policies,...) and applies possible overrules from the related trusted list ("catched" qualifiers from a trust service). More information about qualifiers can be found in the standard ETSI TS 119 612 (R10).

DSS always computes the status at 2 different times: certificate issuance and signing/validation time. The certificate qualification can evolve in the time, its status is not immutable (eg: a trust service provider lost its granted status). The eIDAS regulation (R11) clearly defines these different times in the Article 32 and related Annex I.

Validate a certificate and retrieve its qualification level

```java
// Firstly, we load the certificate to be validated
CertificateToken token = DSSUtils.loadCertificate( 
  new File("src/main/resources/keystore/ec.europa.eu.1.cer"));

// We need a certificate verifier and configure it (see specific chapter about the CertificateVerifier configuration)
CertificateVerifier cv = new CommonCertificateVerifier();

// We create an instance of the CertificateValidator with the certificate
CertificateValidator validator = CertificateValidator.fromCertificate(token);
validator.setCertificateVerifier(cv);

// We execute the validation
CertificateReports certificateReports = validator.validate();

// We have 3 reports
// The diagnostic data which contains all used and static data
DiagnosticData diagnosticData = certificateReports.getDiagnosticData();

// The detailed report which is the result of the process of the diagnostic data and the validation policy
DetailedReport detailedReport = certificateReports.getDetailedReport();

// The simple report is a summary of the detailed report or diagnostic data (more user-friendly)
SimpleCertificateReport simpleReport = certificateReports.getSimpleReport();
```
**Extract the signed data from a signature**

DSS is able to retrieve the original data from a valid signature.

**Retrieve original data from a signed document**

```java
// We have our signed document, we want to retrieve the original/signed data
DSSDocument signedDocument = new FileDocument("src/test/resources/signedXmlXadesB.xml");

// We create an instance of DocumentValidator. DSS automatically selects the validator depending of the
// signature file
SignedDocumentValidator documentValidator = SignedDocumentValidator.fromDocument(signedDocument);

// We set a certificate verifier. It handles the certificate pool, allows to check the certificate status,...
documentValidator.setCertificateVerifier(new CommonCertificateVerifier());

// We retrieve the found signatures
List<AdvancedSignature> signatures = documentValidator.getSignatures();

// We select the wanted signature (the first one in our current case)
AdvancedSignature advancedSignature = signatures.get(0);

// We call get original document with the related signature id (DSS unique ID)
List<DSSDocument> originalDocuments = documentValidator.getOriginalDocuments(advancedSignature.getId());

// We can have one or more original documents depending of the signature (ASiC, PDF,...)
DSSDocument original = originalDocuments.get(0);

original.save("target/original.xml");
```

**REST Services**

DSS offers some REST and SOAP web services. The documentation will covers the REST calls. Additionally, we also provide a SOAP-UI project and Postman samples in the cookbook module.

The different webservice are:

- Signature webservices (dss-soap / dss-rest and their clients): they expose methods to allow to sign or extend a signature from a client.
- Server-signing webservice (dss-server-signing-soap / dss-server-signing-rest and their clients): they expose method to retrieve keys from a server (PKCS#11, PKCS#12, HSM,...) and to sign the digest on the server side.
• Validation webservice (dss-validation-soap / dss-validation-rest and their client) : they expose methods to allow validate a signature, with an optional detached file and an optional validation policy.

The data structure in webservice is similar in REST and SOAP.

REST signature service

This service exposes 3 methods for one or more document(s):

• getDataToSign : computes the digest to be signed
• signDocument : adds the signature value in the document
• extendDocument : extends an existing signature

Get data to sign

The method allows to retrieve the data to be signed. The user sends the document to be signed, the parameters (signature level,...) and the certificate chain.

⚠️ The parameters in getDataToSign and signDocument MUST be the same (especially the signing date).

Request

POST /services/rest/signature/one-document/getDataToSign HTTP/1.1
Accept: application/json, application/javascript, text/javascript, text/json
Content-Type: application/json; charset=UTF-8
Host: localhost:8080
Content-Length: 2433

```json
{
    "parameters" : {
        "signWithExpiredCertificate" : false,
        "generateTBSWithoutCertificate" : false,
        "signatureLevel" : "CAdES_BASELINE_B",
        "signaturePackaging" : "ENVELOPING",
        "signatureAlgorithm" : "RSA_SHA256",
        "encryptionAlgorithm" : "RSA",
        "digestAlgorithm" : "SHA256",
        "referenceDigestAlgorithm" : null,
        "maskGenerationFunction" : null,
        "contentTimestampParameters" : {
            "digestAlgorithm" : "SHA256",
            "canonicalizationMethod" : "http://www.w3.org/2001/10/xml-exc-c14n#"
        },
        "signatureTimestampParameters" : {
            "digestAlgorithm" : "SHA256",
            "canonicalizationMethod" : "http://www.w3.org/2001/10/xml-exc-c14n#"
        }
    }
}
```
"archiveTimestampParameters": {
  "digestAlgorithm": "SHA256",
  "canonicalizationMethod": "http://www.w3.org/2001/10/xml-exc-c14n#"
},
"signingCertificate": {
  "encodedCertificate": "MIIC6jCCAdKgAwIBAgIGLtYU17tXMA0GCSqGSIb3DQEBCwUA..."
},
"certificateChain": [
],
"detachedContents": null,
"asicContainerType": null,
"blevelParams": {
  "trustAnchorBPOnolicy": true,
  "signingDate": 1542794107033,
  "claimedSignerRoles": null,
  "signaturePolicy": null,
  "commitmentTypeIndications": null,
  "signerLocation": null
},
"toSignDocument": {
  "bytes": "SGVsbG8=",
  "digestAlgorithm": null,
  "name": "RemoteDocument",
  "mimeType": null
}
Response

HTTP/1.1 200 OK
Date: Wed, 21 Nov 2018 09:55:07 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Content-Length: 392

{
  "bytes": "MYIBETAYBgkqhkiG9w0BCQMxCwYJKoZiHvcNAQcBMkwGCSqGSIb3DQEJBTEFw0xODExMjEwOTU1MDdaMC0GCSqGSIb3DQEJNDEgMB4wDQYJKoZIhvcNAQELBQAwLwYJKoZiHvcNAQkEMSIEIBfjbjMcSicf41GWg/MJLhZDwCuwTtpRgAfRdKsBpM1W9Cyg6Sib3DQEJEAIvMNgwzjBkMIEIALz6BoBYydCU7yAnSdQjdQ5DfjmsGwARXeFhW3cMD4wNKQyNDAxGzAZBgnVBAHME1v3RTZwYxmuU2lnbmVkRmFrZTERMA8GA1UECgwIFNTLXRlc3QCBi7WFNe7Vw=="
}

Sign document

The method allows to generate the signed document with the received signature value.

⚠️ The parameters in getDataToSign and signDocument MUST be the same (especially the signing date).

Request

POST /services/rest/signature/one-document/signDocument HTTP/1.1
Accept: application/json, application/javascript, text/javascript, text/json
Content-Type: application/json; charset=UTF-8
Host: localhost:8080
Content-Length: 2522

{
  "parameters": {
    "signWithExpiredCertificate": false,
    "generateTBSWithoutCertificate": false,
    "signatureLevel": "CAdES_BASELINE_B",
    "signaturePackaging": "ENVELOPING",
    "signatureAlgorithm": "RSA_SHA256",
    "encryptionAlgorithm": "RSA",
    "digestAlgorithm": "SHA256",
    "referenceDigestAlgorithm": null,
    "maskGenerationFunction": null,
    "contentTimestampParameters": {
      "digestAlgorithm": "SHA256",
      "canonicalizationMethod": "http://www.w3.org/2001/10/xml-exc-c14n#"
    },
    "signatureTimestampParameters": {
      "digestAlgorithm": "SHA256",
      "canonicalizationMethod": "http://www.w3.org/2001/10/xml-exc-c14n#"
  }
}
The method allows to extend an existing signature to a stronger level.

```
POST /services/rest/signature/one-document/extendDocument HTTP/1.1
Accept: application/json, application/javascript, text/javascript, text/json
Content-Type: application/json; charset=UTF-8
Host: localhost:8080
Content-Length: 8677

{
  "toExtendDocument": {

  }
}
```
null
REST server signature service

This service also exposed 3 methods:

- **getKeys**: retrieves available keys on server side
- **getKey**: retrieves a key on the server side by its alias
- **sign**: signs the digest value with the given key

Get keys

This method allows to retrieve all available keys on the server side (PKCS#11, PKCS#12, HSM,...). All keys will have an alias, a signing certificate and its chain. The alias will be used in following steps.

Request

```
GET /services/rest/server-signing/keys HTTP/1.1
Accept: application/json, application/javascript, text/javascript, text/json
Host: localhost:8080
```
HTTP/1.1 200 OK
Date: Wed, 21 Nov 2018 09:55:06 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Content-Length: 2189

[ {
  "alias": "certificate",
  "encryptionAlgo": "RSA",
  "certificate": {
    "encodedCertificate": "MIIC6jCCAdKgAwIBAgIGLtyU17txMA0GCSqGSIb3DQEBCwUAJMAxGzAZBgNVBAMME1vb3RTZwxmu21nbmvKr
mFrZTERMA8GA1UEcgwIRFNTLX1c3QwhcNMcwnJ4bMTExj3AxWhcNMDcwnAzO0M0dc1NzOWj0aMoRMQd9YDVQQ
DDApTaWduZXJGYh2L1MrdWydVYQQDAhEuU1mt6GvzcDDCASIwDQYJKoZIhvcNAQEBBQADgEPADCCAQoCggEBA
MI3kZhnipi+iZH29ax8FlfE50w/cfwBTFaE831ZQUp6/BQnBt70qO0XWbte9qkv7/UDdcB3XPV5QWS5AyMPH
pqQ75Hijsjq/Fzu8EhtkKfPizcxg6aBZd0k8h4rSrt01Kj0s0Rd5DqWsgkeVCCN90k0ZsZ0ENy+I80QxSmzt
sStkyYUdlqzw24JEXW9vz64eTbde4vQbJ6pJG4Jf1gQNEc2ZxzhmI/prXLYsWNqC71L27PUZTrdegABTuzYCR
JKwBmPmq4o6L0N49c9QDq4a5k3GowHegL2nAQ128x0M3A59TC+ieNc6VFiPSTLPuw7PNI2VstX5juausC1
weEAAaM5MBawgYDVR0PAQH/BAQDAgEGMA0CSqGSIb3DQEBCwUAA4IBAQQK6LGA01TR+rmlU86y8hA40kDN2b1
dbIL8181CFyolCXX8xq3ubZC0xqh1X2jg0wzr0b0/MUix00IuVnBf0xAW7PBZKIDnm6lsckRxs1U32sC
9d1L0He35KBNB6GZALT1ewj7h5bWfjtmcoVq6eVGA5cvfu/2/TkJyKV/NT39HnXrdspdygvwypEtXjetz
D7U793nuw3cXv8VIFtVlwhf9LjU7h+UJmKX9G15eYr3sZLum6vy0I08bv14PwtsWGL0Q0SRvIBBrP4adc6GT
gjgjolkTC0BB8Fkrr+8lkHuc6bpl4LUIuOk5ILXsiEeE9WAqmxQx0"
  },
  "certificateChain": [
    {
      "encodedCertificate": "MIIC6jCCAdKgAwIBAgIGLtyU17txMA0GCSqGSIb3DQEBCwUAJMAxGzAZBgNVBAMME1vb3RTZwxmu21nbmvKr
mFrZTERMA8GA1UEcgwIRFNTLX1c3QwhcNMcwnJ4bMTExj3AxWhcNMDcwnAzO0M0dc1NzOWj0aMoRMQd9YDVQQ
DDApTaWduZXJGYh2L1MrdWydVYQQDAhEuU1mt6GvzcDDCASIwDQYJKoZIhvcNAQEBBQADgEPADCCAQoCggEBA
MI3kZhnipi+iZH29ax8FlfE50w/cfwBTFaE831ZQUp6/BQnBt70qO0XWbte9qkv7/UDdcB3XPV5QWS5AyMPH
pqQ75Hijsjq/Fzu8EhtkKfPizcxg6aBZd0k8h4rSrt01Kj0s0Rd5DqWsgkeVCCN90k0ZsZ0ENy+I80QxSmzt
sStkyYUdlqzw24JEXW9vz64eTbde4vQbJ6pJG4Jf1gQNEc2ZxzhmI/prXLYsWNqC71L27PUZTrdegABTuzYCR
JKwBmPmq4o6L0N49c9QDq4a5k3GowHegL2nAQ128x0M3A59TC+ieNc6VFiPSTLPuw7PNI2VstX5juausC1
weEAAaM5MBawgYDVR0PAQH/BAQDAgEGMA0CSqGSIb3DQEBCwUAA4IBAQQK6LGA01TR+rmlU86y8hA40kDN2b1
dbIL8181CFyolCXX8xq3ubZC0xqh1X2jg0wzr0b0/MUix00IuVnBf0xAW7PBZKIDnm6lsckRxs1U32sC
9d1L0He35KBNB6GZALT1ewj7h5bWfjtmcoVq6eVGA5cvfu/2/TkJyKV/NT39HnXrdspdygvwypEtXjetz
D7U793nuw3cXv8VIFtVlwhf9LjU7h+UJmKX9G15eYr3sZLum6vy0I08bv14PwtsWGL0Q0SRvIBBrP4adc6GT
gjgjolkTC0BB8Fkrr+8lkHuc6bpl4LUIuOk5ILXsiEeE9WAqmxQx0"
    }
  ]
},

Get key

This method allows to retrieve key informations for a given alias.
GET /services/rest/server-signing/key/certificate HTTP/1.1
Accept: application/json, application/javascript, text/javascript, text/json
Host: localhost:8080

HTTP/1.1 200 OK
Date: Wed, 21 Nov 2018 09:55:06 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Content-Length: 2185

{
    "alias": "certificate",
    "encryptionAlgo": "RSA",
    "certificate": {
        "encodedCertificate": "MIIC6jCCAgIBAgIGLTY1t7tXMA0GCSqGSIb3DQEBCwUAMDAxgAZBgNVBAMME1Jvb3RTZWxmU2lnbmkVRmFr7TERMA8GA1UECwIFNTLX3c3QwHcMCwNzA0MDc1Nzi0VgAoMRMwEQYDVQQDDApTaWduZXJGWhEwDQYJKoZIhvcNAQEBQjAGMDAwE/DQYJK0ZIhvcNAQcDQwMCQ0rMRswEAYDVR0gBEsDAKCCgghcQgAgAAwIBAgIE
        "certificateChain": [
            {
                "encodedCertificate": "MIIC6jCCAgIBAgIGLTY1t7tXMA0GCSqGSIb3DQEBCwUAMDAxgAZBgNVBAMME1Jvb3RTZWxmU2lnbmkVRmFr7TERMA8GA1UECwIFNTLX3c3QwHcMCwNzA0MDc1Nzi0VgAoMRMwEQYDVQQDDApTaWduZXJGWhEwDQYJKoZIhvcNAQEBQjAGMDAwE/DQYJK0ZIhvcNAQcDQwMCQ0rMRswEAYDVR0gBEsDAKCCgghcQgAgAAwIBAgIE
        }
    },
    "chain": [
        {
            "encodedCertificate": "MIIC6jCCAgIBAgIGLTY1t7tXMA0GCSqGSIb3DQEBCwUAMDAxgAZBgNVBAMME1Jvb3RTZWxmU2lnbmkVRmFr7TERMA8GA1UECwIFNTLX3c3QwHcMCwNzA0MDc1Nzi0VgAoMRMwEQYDVQQDDApTaWduZXJGWhEwDQYJKoZIhvcNAQEBQjAGMDAwE/DQYJK0ZIhvcNAQcDQwMCQ0rMRswEAYDVR0gBEsDAKCCgghcQgAgAAwIBAgIE
        }
    ]
}
Sign

This method allows to sign the given digest with a server side certificate.

Request

```
POST /services/rest/server-signing/sign/certificate/SHA256 HTTP/1.1
Accept: application/json, application/javascript, text/javascript, text/json
Content-Type: application/json; charset=UTF-8
Host: localhost:8080
Content-Length: 24

{
    "bytes": "AQID"
}
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 21 Nov 2018 09:55:06 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Content-Length: 395

{
    "algorithm": "RSA_SHA256",
    "value": "AZgLVQQLPQkPgRlfTNfTg3QlcDa0JTb0lS6kSteHxHLvjTmtKnRfYTVPZ0bupdPMVQIfuBt4OQvZzVtTbor+k
jtu7Baae050mXB8OMvo93F/ZmHPIf8VduPASq17xc4TN73I6KoAn6ouYT0jux1uQa9r79yvGo/qhoUw9R/j6f0fGPKNhG
VDqnG1rHX0qEWPKlYxetiTLnaIZGxuZ9p2vDzZRoEaTs0UwcFu8Yln9Xk8fe6hSxAQOncBXwXQX8LKAmZH4/QLsGuJwr+2Fh
snC4s1X17dXPzAlqLU38gmamK+OjqMTIPmQioLq2WLVhLye59dHvgvDChkTW3IZA=
}
```

REST validation service

Validate a document

This service allows to validate signature (all formats/types) against a validation policy.

Request

```
POST /services/rest/validation/validateSignature HTTP/1.1
Accept: application/json, application/javascript, text/javascript, text/json
Content-Type: application/json; charset=UTF-8
Host: localhost:8080
Content-Length: 7495

{
}
```
HTTP/1.1 200 OK
Date: Wed, 21 Nov 2018 09:55:06 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Content-Length: 31957

{
  "diagnosticData": {
    "documentName": "xades-detached.xml",
    "validationDate": 1542794106070,
    "containerInfo": null,
    "signatures": [
      {
        "signatureFilename": "xades-detached.xml",
        "parentId": null,
        "errorMessage": null,
        "dateTime": 1506596944000,
        "signatureFormat": "XAdES-BASELINE-B",
        "structuralValidation": {
          "valid": true,
          "message": null
        },
        "digestMatchers": [
          {
            "digestMethod": "SHA256",
            "digestValue": "kcDHOZjwZhVfuDhuhCeCERRmYPht4Jj4RmfVVi31Q9g=",
            "dataFound": true,
            "dataIntact": true,
            "type": "REFERENCE",
            "name": "r-id-1"
          },
          {
            "digestMethod": "SHA256",
            "digestValue": "DztwNTmRomOAm6/lMI8Rym5xZPz1vLYDzn/ebYYkPsr4=",
            "dataFound": true,
            "dataIntact": true,
            "type": "SIGNED_PROPERTIES",
            "name": "#xades-id-afde782436468dd74eeb181f7ce110e1"
          }
        ],
        "basicSignature": {
          "encryptionAlgoUsedToSignThisToken": "RSA",
          "keyLengthUsedToSignThisToken": "2048",
          "digestAlgoUsedToSignThisToken": "SHA256",
          "maskGenerationFunctionUsedToSignThisToken": null,
          "signatureIntact": true,
          "signatureValid": true
        },
        "signingCertificate": {
          "attributePresent": true,
          "digestValuePresent": true,
          "digestValueMatch": true,
          "issuerSerialMatch": true
        }
      }
    ]
  }
}
"id" : "F0FF0B4514D316304F2817DBA008FB05DEDB98527C0E47C73E8D8FD9F8E16B267E",
"certificateChain" : [ { "source" : "SIGNATURE",
"id" : "F0FF0B4514D316304F2817DBA008FB05DEDB98527C0E47C73E8D8FD9F8E16B267E" }, { "source" : "SIGNATURE",
"id" : "6F35DE3965B9A69BC3661D1A355B0AE60907ADB741CC1911EFD0F3BE72D6A6E9" } ],
"contentType" : "text/xml",
"contentIdentifier" : null,
"contentHints" : null,
"signatureProductionPlace" : null,
"commitmentTypeIndication" : [ ],
"claimedRoles" : [ ],
"certifiedRoles" : [ ],
"policy" : null,
"timestamps" : [ ],
"signatureScopes" : [ { "value" : "Full document",
"name" : "sample.xml",
"scope" : "FULL"
} ],
"id" : "id-afde7f8243646bd4dd74eeb1f7ce110e1",
"counterSignature" : null
"usedCertificates" : [ { "subjectDistinguishedName" : [ { "value" : "c=lu,ou=pki-test,o=nowina solutions,cn=good-ca",
"format" : "CANONICAL"
} ],
"value" : "C=LU,OU=PKI-TEST,O=Nowina Solutions,CN=good-ca",
"format" : "RFC2253"
} ],
"issuerDistinguishedName" : [ { "value" : "c=lu,ou=pki-test,o=nowina solutions,cn=root-ca",
"format" : "CANONICAL"
} ],
"value" : "C=LU,OU=PKI-TEST,O=Nowina Solutions,CN=root-ca",
"format" : "RFC2253"
} ],
"serialNumber" : 4,
"commonName" : "good-ca",
"locality" : null,
"state" : null,
"countryName" : "LU",
"organizationName" : "Nowina Solutions",
"givenName" : null,
"organizationalUnit" : "PKI-TEST",
"surname" : null,
"pseudonym" : null,
"value": "c=lu,ou=pki-test,o=nowina solutions,cn=root-ca",
"format": "CANONICAL"
},
"value": "C=LU,OU=PKI-TEST,O=Nowina Solutions,CN=root-ca",
"format": "RFC2253"
],
"issuerDistinguishedName": [
"value": "c=lu,ou=pki-test,o=nowina solutions,cn=root-ca",
"format": "CANONICAL"
],
"value": "C=LU,OU=PKI-TEST,O=Nowina Solutions,CN=root-ca",
"format": "RFC2253"
],
"serialNumber": 1,
"commonName": "root-ca",
"locality": null,
"state": null,
"countryName": "LU",
"organizationName": "Nowina Solutions",
"givenName": null,
"organizationalUnit": "PKI-TEST",
"surname": null,
"pseudonym": null,
"email": null,
"authorityInformationAccessUrls": [],
"digestAlgoAndValues": [],
"notAfter": 1571924603000,
"notBefore": 1508852603000,
"publicKeySize": 2048,
"publicKeyEncryptionAlgo": "RSA",
"keyUsageBits": ["keyCertSign", "crlSign"],
"extendedKeyUsages": [],
"idPkixOcspNoCheck": false,
"basicSignature": {
"encryptionAlgoUsedToSignThisToken": "RSA",
"keyLengthUsedToSignThisToken": "2048",
"digestAlgoUsedToSignThisToken": "SHA512",
"maskGenerationFunctionUsedToSignThisToken": null,
"signatureIntact": true,
"signatureValid": true
},
"signingCertificate": null,
"certificateChain": [],
"trusted": false,
"selfSigned": true,
"certificatePolicies": [],
"trustedServiceProviders": [],
"revocations": [],
"base64Encoded": "MIIDVzCCAj+gAwIBAgIBATANBgkqhkiG9w0BAQ0FADBNMRAwDgYDVQQLDAhQS0ktVEVTVDELMAkGA1UEBhMCTFUhMCkGA1UEAwwHhcNMTcxMDI0MTM0MzIzWhc"
"extendedKeyUsages" : [ ],
"idPkixOcspNoCheck" : false,
"basicSignature" : { 
    "encryptionAlgoUsedToSignThisToken" : "RSA",
    "keyLengthUsedToSignThisToken" : "2048",
    "digestAlgoUsedToSignThisToken" : "SHA256",
    "maskGenerationFunctionUsedToSignThisToken" : null,
    "signatureIntact" : true,
    "signatureValid" : true,
},

"signingCertificate" : { 
    "attributePresent" : null,
    "digestValuePresent" : null,
    "digestValueMatch" : null,
    "issuerSerialMatch" : null,
    "id" : "6F35DE3965B9A69B3C661D1A355B0AE60907ADB741CC1911EFD0F3BE72D6A6E9",
},

"certificateChain" : [ 
    { 
        "source" : "SIGNATURE",
        "id" : "6F35DE3965B9A69B3C661D1A355B0AE60907ADB741CC1911EFD0F3BE72D6A6E9",
    }
],

"trusted" : false,
"selfSigned" : false,
"certificatePolicies" : [ ],
"trustedServiceProviders" : [ ],

"ocspaccessUrls" : [ "http://dss.nowina.lu/pki-factory/ocsp/good-ca" ],
"crldistributionPoints" : [ ],
"qcstatementIds" : [ ]
}
"simpleReport": {
  "policy": {
    "policyName": "QES AdES QC TL based",
    "policyDescription": "Validate electronic signatures and indicates whether they are Advanced electronic Signatures (AdES), AdES supported by a Qualified Certificate (AdES/QC) or a Qualified electronic Signature (QES). All certificates and their related chains supporting the signatures are validated against the EU Member State Trusted Lists (this includes the signer's certificate and certificates used to certify the validity status services - CRLs, OCSP, and time-stamps).
  ",
  "qctypes": [ ],
  "ocspaccessUrls": [ ],
  "crldistributionPoints": [ "http://dss.nowina.lu/pki-factory/crl/root-ca.crl" ],
  "qcstatementIds": [ ]
},
"trustedLists": [ ],
"listOfTrustedLists": null
}
"bestSignatureTime": 1542794106070,
"signedBy": "good-user",
"certificateChain": {
  "certificate": [
    {
      "id": "F0FF0B4514D316304F2817DA0BF05DEDB98527C0E47C73E8D8FDFE16DF267E",
      "qualifiedName": "good-user"
    },
    {
      "id": "6F35DE3965B9A698C3661D1A355B0AE60907AD741CC1911EF0F3BE72D6A6E9",
      "qualifiedName": "good-ca"
    }
  ],
  "signatureLevel": {
    "value": "NA",
    "description": "Not applicable"
  },
  "indication": "INDETERMINATE",
  "subIndication": "NO_CERTIFICATE_CHAIN_FOUND",
  "errors": [ "The certificate path is not trusted!", "The result of the LTV validation process is not acceptable to continue the process!" ],
  "warnings": [ "The signature/seal is an INDETERMINATE AdES!" ],
  "infos": [ ]
},
"signatureScope": [ {
  "value": "Full document",
  "name": "sample.xml",
  "scope": "FULL"
} ],
"id": "id-afde782436468dd74eeb181f7ce110e1",
"counterSignature": null,
"parentId": null,
"signatureFormat": "XAdES-BASELINE-B"
},
"detailedReport": {
  "signatures": [ {
    "validationProcessBasicSignatures": {
      "constraint": [ {
        "name": {
          "value": "Is the result of the Basic Validation Process conclusive?",
          "nameId": "ADEST_ROBVPIC"
        },
        "status": "NOT_OK",
        "error": [ {
          "value": "The result of the Basic validation process is not conclusive!",
          "nameId": "ADEST_ROBVPIC_ANS"
        },
        "warning": null,
        "info": null,
        "additionalInfo": null,
        "id": "id-afde782436468dd74eeb181f7ce110e1"
      } ]
    },
    "conclusion": {
      "name": "sample.xml",
      "scope": "FULL"
    }
  } ]
}
"indication" : "INDETERMINATE",
"subIndication" : "NO_CERTIFICATE_CHAIN_FOUND",
"errors" : [ {
    "value" : "The certificate chain for signature is not trusted, there is no trusted anchor.",
    "nameId" : "BBB_XCV_CCCBB_SIG_ANS"
} ],
"warnings" : [ ],
"infos" : [ ]
},
"bestSignatureTime" : 1542794106070
},
"validationProcessTimestamps" : [ ],
"validationProcessLongTermData" : {
    "constraint" : [ {
        "name" : {
            "value" : "Is the result of the Basic Validation Process acceptable?",
            "nameId" : "LTV_ASVB"
        },
        "status" : "NOT_OK",
        "error" : {
            "value" : "The result of the Basic validation process is not acceptable to continue the process!",
            "nameId" : "LTV_ASVB_ANS"
        },
        "warning" : null,
        "info" : null,
        "additionalInfo" : null,
        "id" : null
    } ],
    "conclusion" : {
        "indication" : "INDETERMINATE",
        "subIndication" : "NO_CERTIFICATE_CHAIN_FOUND",
        "errors" : [ {
            "value" : "The certificate chain for signature is not trusted, there is no trusted anchor.",
            "nameId" : "BBB_XCV_CCCBB_SIG_ANS"
        } ],
        "warnings" : [ ],
        "infos" : [ ]
    },
    "bestSignatureTime" : 1542794106070
},
"validationProcessArchivalData" : {
    "constraint" : [ {
        "name" : {
            "value" : "Is the result of the LTV validation process acceptable?",
            "nameId" : "ARCH_LT"V"
        },
        "status" : "NOT_OK",
        "error" : {
            "value" : "The result of the LTV validation process is not acceptable!",
            "nameId" : "ARCH_LT"V_ANS"
        }
    } ],
    "conclusion" : {
        "indication" : "INDETERMINATE",
        "subIndication" : "NO_CERTIFICATECHAIN FOUND",
        "errors" : [ {
            "value" : "The certificate chain for signature is not trusted, there is no trusted anchor.",
            "nameId" : "BBB_XCV_CCCBB_SIG_ANS"
        } ],
        "warnings" : [ ],
        "infos" : [ ]
    },
    "bestSignatureTime" : 1542794106070
}
"value": "The result of the LTV validation process is not acceptable to continue the process!",
"nameId": "ARCH_LTVV_ANS"
],
"warning": null,
"info": null,
"additionalInfo": null,
"id": null
}
],
"conclusion": {
"indication": "INDETERMINATE",
"subIndication": "NO_CERTIFICATE_CHAIN_FOUND",
"errors": [
{
"value": "The certificate chain for signature is not trusted, there is no trusted anchor.",
"nameId": "BBB_XCV_CCCBB_SIG_ANS"
}],
"warnings": [],
"infos": []
},
"bestSignatureTime": 1542794106070
},
"validationSignatureQualification": {
"constraint": [
{
"name": {
"value": "Is the signature/seal an acceptable AdES (ETSI EN 319 102-1) 
?",
"nameId": "QUAL_IS_ADES"
},
"status": "WARNING",
"error": null,
"warning": {
"value": "The signature/seal is an INDETERMINATE AdES!",
"nameId": "QUAL_IS_ADES_IND"
},
"info": null,
"additionalInfo": null,
"id": null
},
{
"name": {
"value": "Is the certificate path trusted?",
"nameId": "QUAL_TRUSTED_CERT_PATH"
},
"status": "NOT_OK",
"error": {
"value": "The certificate path is not trusted!",
"nameId": "QUAL_TRUSTED_CERT_PATH_ANS"
},
"warning": null,
"info": null,
"additionalInfo": null,
"id": null
}]
}
"id" : null
},
"conclusion" : {
"indication" : "FAILED",
"subIndication" : null,
"errors" : [ {
"value" : "The certificate path is not trusted!",
"nameId" : "QUAL_TRUSTED_CERT_PATH_ANS"
}, {
"value" : "The certificate path is not trusted!",
"nameId" : "QUAL_TRUSTED_CERT_PATH_ANS"
} ],
"Warnings" : [ {
"value" : "The signature/seal is an INDETERMINATE AdES!",
"nameId" : "QUAL_IS_ADES_IND"
} ],
"infos" : [ ]
},
"validationCertificateQualification" : [ ],
"id" : null,
"signatureQualification" : "NA"
},
"id" : "id-afde782436468dd74eeb181f7ce110e1",
"counterSignature" : null
},
"certificate" : null,
"basicBuildingBlocks" : [ {
"fc" : {
"constraint" : [ {
"name" : {
"value" : "Is the expected format found?",
"nameId" : "BBB_FC_IEFF"
},
"status" : "OK",
"error" : null,
"warning" : null,
"info" : null,
"additionalInfo" : null,
"id" : null
} ],
"conclusion" : {
"indication" : "PASSED",
"subIndication" : null,
"errors" : [ ],
"warnings" : [ ],
"infos" : [ ]
}
},
"isc" : {
"constraint" : [ {
"name" : {
"value" : "Is the expected format found?",
"nameId" : "BBB_FC_IEFF"
}
} ]
}
"value": "Is there an identified candidate for the signing certificate?",
"nameId": "BBB_ICS_ISCI"
},
"status": "OK",
"error": null,
"warning": null,
"info": null,
"additionalInfo": null,
"id": null
}, {
"name": {
"value": "Is the signed attribute: 'signing-certificate' present?",
"nameId": "BBB_ICS_ISASCP"
},
"status": "OK",
"error": null,
"warning": null,
"info": null,
"additionalInfo": null,
"id": null
}, {
"name": {
"value": "Is the signed attribute: 'cert-digest' of the certificate present?",
"nameId": "BBB_ICS_ISACDP"
},
"status": "OK",
"error": null,
"warning": null,
"info": null,
"additionalInfo": null,
"id": null
}, {
"name": {
"value": "Is the certificate's digest value valid?",
"nameId": "BBB_ICS_ICDVV"
},
"status": "OK",
"error": null,
"warning": null,
"info": null,
"additionalInfo": null,
"id": null
}, {
"name": {
"value": "Are the issuer distinguished name and the serial number equal?",
"nameId": "BBB_ICS_AIDNASNE"
},
"status": "OK",
"error": null,
"warning": null,
"info": null,
"additionalInfo": "Reference : r-id-1",
"id": null
},
{
"name": {
"value": "Is the reference data object intact?",
"nameId": "BBB_CV_IRDOI"
},
"status": "OK",
"error": null,
"warning": null,
"info": null,
"additionalInfo": "Reference : r-id-1",
"id": null
},
{
"name": {
"value": "Is the reference data object found?",
"nameId": "BBB_CV_IRDOF"
},
"status": "OK",
"error": null,
"warning": null,
"info": null,
"additionalInfo": "Reference : #xades-id-afde782436468dd74eeb181f7ce110e1",
"id": null
},
{
"name": {
"value": "Is the signature intact?",
"nameId": "BBB_CV_ISI"
},
"status": "OK",
"error": null,
"warning": null,
"info": null,
"additionalInfo": null,
"id": null
}
],
"conclusion": {
"indication": "PASSED",
"value": null,
"status": null,
"error": null,
"warning": null,
"info": null,
"additionalInfo": null,
"id": null
}
"subIndication" : null,
"errors" : [ ],
"warnings" : [ ],
"infos" : [ ]
}
,"sav" : {
"constraint" : [ {
"name" : {
"value" : "Is signed qualifying property: 'signing-time' present?",
"nameId" : "BBB_SAV_ISQPSTP"
},
"status" : "OK",
"error" : null,
"warning" : null,
"info" : null,
"additionalInfo" : null,
"id" : null
}, {
"name" : {
"value" : "Are signature cryptographic constraints met?",
"nameId" : "ASCCM"
},
"status" : "OK",
"error" : null,
"warning" : null,
"info" : null,
"additionalInfo" : "Validation time : 2018-11-21 09:55",
"id" : null
} ]
},
"conclusion" : {
"indication" : "PASSED",
"subIndication" : null,
"errors" : [ ],
"warnings" : [ ],
"infos" : [ ]
}
},
"xcv" : {
"constraint" : [ {
"name" : {
"value" : "Can the certificate chain be built till the trust anchor?",
"nameId" : "BBB_XCV_CCCBB"
},
"status" : "NOT_OK",
"error" : {
"value" : "The certificate chain for signature is not trusted, there is no trusted anchor.",
"nameId" : "BBB_XCV_CCCBB_SIG_ANS"
},
"warning" : null,
"id" : null
} ]
}
Retrieve original document(s)

This service returns the signed data for a given signature.
Response

HTTP/1.1 200 OK
Date: Wed, 21 Nov 2018 09:55:06 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Content-Length: 101

[ {
  "bytes": "aGVsbG8=",
  "digestAlgorithm": null,
  "name": null,
  "mimeType": null
} ]