DSS Extension for Local Signature Computation Version 1.0

Committee Specification Draft 04 / Public Review Draft 04

22 February 2018

Specification URIs

This version:

http://docs.oasis-open.org/dss-x/localsig/v1.0/csprd04/localsig-v1.0-csprd04.html (Authoritative)
http://docs.oasis-open.org/dss-x/localsig/v1.0/csprd04/localsig-v1.0-csprd04.pdf
http://docs.oasis-open.org/dss-x/localsig/v1.0/csprd04/localsig-v1.0-csprd04.xml

Previous version:

http://docs.oasis-open.org/dss-x/localsig/v1.0/cs02/localsig-v1.0-cs01.html
http://docs.oasis-open.org/dss-x/localsig/v1.0/cs02/localsig-v1.0-cs01.pdf
http://docs.oasis-open.org/dss-x/localsig/v1.0/cs02/localsig-v1.0-cs01.xml

Latest version:

http://docs.oasis-open.org/dss-x/localsig/v1.0/localsig-v1.0.html (Authoritative)
http://docs.oasis-open.org/dss-x/localsig/v1.0/localsig-v1.0.pdf
http://docs.oasis-open.org/dss-x/localsig/v1.0/localsig-v1.0.xml

Technical Committee:

OASIS Digital Signature Services eXtended (DSS-X) TC

Chairs:

Juan Carlos Cruellas (cruellas@ac.upc.edu), UPC-DAC
Stefan Hagen (stefan@dilettant.eu), Individual

Editors:

Ernst Jan van Nigtevecht (ejvn@sonnenglanz.net), Sonnenglanz Consulting BV
Frank Cornelis (frank.cornelis@fedict.be), FedICT
Detlef Hühnlein (detlef.huehnlein@ecsec.de), Individual

Additional artifacts:

This prose specification is one component of a Work Product that also includes:

- XML schemas: http://docs.oasis-open.org/dss-x/localsig/v1.0/csprd04/schemas/

Related work:

This specification is related to:

- *Digital Signature Service Core Protocols, Elements, and Bindings Version 1.0*. Edited by Stefan Drees. 11 April 2007. OASIS Standard. http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-spec-v1.0-os.html.

Copyright © OASIS Open 2018. All Rights Reserved.

Declared XML Namespaces:

http://docs.oasis-open.org/dss-x/ns/localsig

Abstract:

The core OASIS Digital Signature Service webservice [**DSSCore**] supports the creation of signatures on behalf of applications and / or users by utilizing server-based signature keys.

This *Local Signature Computation* profile extends the core functionality such that end users can bring (use) their own (secure) signature-creation device. Examples of such devices are smartcards or usb-tokens but also smartphones, mobile phones, tablets, pc's or laptops with privately held signature keys.

Four solutions are presented to support the varying capabilities of applications and different use cases. The first solution is useful for web-applications where web browsers can access the (qualified) signature-creation device that is available at the desktop (e.g. a smartcard connected via USB). The second solution is useful for applications that can access the (qualified) signature-creation device themselve, for instance desktop applications or smartphone apps. The third solution is useful for any application where the (qualified) signature-creation device can only be accessed via a separate channel, for instance a mobile device, through a third-party. The fourth solution is useful for web-applications, which want to access (qualified) signature-creation devices via a localhost based ChipGateway.

Status:

This document was last revised or approved by the OASIS Digital Signature Services eXtended (DSS-X) TC on the above date. The level of approval is also listed above. Check the "Latest version" location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=dss-x#technical.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committees/dss-x/.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/dss-x/ipr.php).

Note that any machine-readable content (aka Computer Language Definitions) declared Normative for this Work Product is provided in separate plain text files. In the event of a discrepancy between any such plain text file and display content in the Work Product's prose narrative document(s), the content in the separate plain text file prevails.

Citation format:

When referencing this specification the following citation format should be used:

[**localsig-v1.0**]

DSS Extension for Local Signature Computation Version 1.0. Edited by Ernst Jan van Nigtevecht and Frank Cornelis. 22 February 2018. OASIS Committee Specification Draft 04 / Public Review Draft 04. *http://docs.oasis-open.org/dss-x/localsig/v1.0/csprd04/localsig-v1.0-csprd04.html.* Latest version: *http://docs.oasis-open.org/dss-x/localsig/v1.0/localsig-v1.0.html.*

 Copyright © OASIS Open 2018. All Rights Reserved.

Notices

Copyright © OASIS Open 2018. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at http://www.oasis-open.org/who/intellectualproperty.php.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name "OASIS" is a trademark of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see http://www.oasis-open.org/who/trademark.php for above guidance.

Table of Contents

Copyright © OASIS Open 2018. All Rights Reserved.

Appendixes

 Copyright © OASIS Open 2018. All Rights Reserved.

1 Introduction

The OASIS Digital Signature Services specification [**DSSCore**] standardizes a protocol by which (i) a client can send documents (or document digests) to a server and receive back a signature on the documents, or by which (ii) a client can send documents (or document digests) and a signature to a server, and receive back an answer on whether the signature verifies the documents.

These operations can be useful in a variety of contexts, for example they can allow clients to access a single corporate key for signing press releases, with centralized access control, auditing, and archiving of signature requests. They can also allow clients to create signatures without needing complex client software and configuration.

This profile extends the OASIS DSS protocol such that a (qualified) signature-creation device (an QSCD or SCD), under the direct control of the user, is used for the actual computation of the digital signature value. The (qualified) signature-creation device is not part of, nor located at, the server that implements the DSS protocol.

The (qualified) signature-creation device can have limited software and performance capabilities and hence a OASIS DSS compliant service can be used to handle the complexities of the (qualified) signature-creation and document manipulation.

1.1 Terminology

The key words *MUST*, *MUST NOT*, *REQUIRED*, *SHALL*, *SHALL NOT*, *SHOULD*, *SHOULD NOT*, *RECOMMENDED*, *MAY*, and *OPTIONAL* are to be interpreted as described in [**RFC 2119**].

These keywords are capitalized when used to unambiguously specify requirements over protocol features and behavior that affect the interoperability and security of implementations. When these words are not capitalized, they are meant in their natural-language sense.

This specification uses the following typographical conventions in text: `<ns:Element>`, `Attribute`, **Datatype**, `OtherCode`.

An **input document** that has to be signed can be any piece of data that can be used as input to a signature calculation, according to the [**DSSCore**] specification, Section 2.4. Such a document can even be a signature (for example, a pre-existing signature can be counter-signed) or timestamp.

A **digital signature value** is a basic form of a signature, conformant to a `<ds:Signature>` or `<dss:Base64Signature>` within a `<dssSignatureObject>` element according to the [**DSSCore**] specification, Section 2.5. The digital signature value is computed by the (secure) signature-creation device for a given digest. The validity of the corresponding certificate is not checked by the device.

An **advanced electronic signature** is an enriched digital signature value and can include for instance a time-stamp (to specify the time of signing) and/or revocation information (to specify the validity of the corresponding certificate and trusted roots and/or intermediate certificates). Additionally, the advanced electronic signature will only be created by the Digital Signature Service if the corresponding certificate is valid (not revoked).

1.2 Abbreviations

- *SCD*: Signature-Creation Device. A device that is capable of creating a digital signature value using a private key that is stored in the device.
- *QSCD*: Qualified Signature-Creation Device. A signature-creation device which meets the requirements laid down in Annex II of the Reglation (EU) No. 910/2014 [**eIDAS**].
- *LSCD*: Local Signature-Creation Device. A (qualified) signature-creation device that is owned by and in the proximity of an end user.

Copyright © OASIS Open 2018. All Rights Reserved.

- *RSCD*: Remote Signature-Creation Device. A (qualified) signature-creation device that is owned by, but not in the proximity of, an end user. Nonetheless, the usage of the device is (by some other means) under the control of the end user.
- *Client*: A requester of a particular resource or service that is provided by a server.
- *Server*: A provider of a resource or service that is used by a client.

1.3 Normative References

[**DSSCore**]
S. Drees et al., *Digital Signature Service Core Protocols and Elements*, OASIS, April 2007, *http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-spec-v1.0-os.pdf*

[**LocalSigXSD**]
E.J. van Nigtevecht et al., *DSS-X Local Signature Computation Profile XML Schema Definition*, OASIS, 20 February 2017, *http://docs.oasis-open.org/dss-x/localsig/v1.0/cs02/schemas/localsig-v1.0.xsd*

[**Excl-C14N**]
J. Boyer et al., *Exclusive XML Canonicalization Version 1.0*, World Wide Web Consortium, July 2002, *http://www.w3.org/TR/xml-exc-c14n/*

[**HTML401**]
D. Raggett et al., *HTML 4.01 Specification*, World Wide Web Consortium, December 1999, *http://www.w3.org/TR/html4*

[**RFC 2119**]
S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, *http://www.ietf.org/rfc/rfc2119.txt* IETF (Internet Engineering Task Force) RFC 2119, March 1997.

[**RFC 2616**]
R. Fielding et al., *Hypertext Transfer Protocol - HTTP/1.1*, *http://www.ietf.org/rfc/rfc2616.txt* IETF (Internet Engineering Task Force) RFC 2616, June 1999.

[**RFC 3061**]
M. Mealling, *A URN Namespace of Object Identifiers*, *http://tools.ietf.org/rfc/rfc3061.txt* IETF (Internet Engineering Task Force), February 2001.

[**SAMLCore**]
S. Cantor et al., *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*, OASIS, March 2005, *http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf*

[**XHTML**]
XHTML 1.0 The Extensible HyperText Markup Language (Second Edition), World Wide Web Consortium Recommendation, August 2002, *http://www.w3.org/TR/xhtml1/ [http://www.w3.org/TR/xhtml1/]*

[**XMLSig**]
D. Eastlake et al., *XML-Signature Syntax and Processing*, W3C Recommendation, June 2008, *http://www.w3.org/TR/xmldsig-core/*

[**XML-ns**]
T. Bray, D. Hollander, A. Layman, *Namespaces in XML*, W3C Recommendation, January 1999, *http://www.w3.org/TR/1999/REC-xml-names-19990114*

[**XML-Schema**]
H. S. Thompson et al., *XML Schema Part 1: Structures*, W3C Recommendation, May 2001, *http://www.w3.org/TR/xmlschema-1/*

Copyright © OASIS Open 2018. All Rights Reserved.

1.4 Non-Normative References

[**BSI-TR-02102**]

eID-Client, Technical Guideline Nr. 03124, Part 1-2 , Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik, BSI), February 2017, *https://www.bsi.bund.de/EN/Publications/TechnicalGuidelines/tr02102/tr02102_node.html*

[**BSI-TR-03124**]

eID-Client, Technical Guideline Nr. 03124, Part 1-2 , Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik, BSI), February 2015, *http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-spec-v1.0-os.pdf*

[**CGW-Contribution**]

LuxTrust S.A., ecsec GmbH *ChipGateway Protocol - OASIS Contribution* *https://www.oasis-open.org/committees/download.php/60049/ChipGateway-Specification-OASIS.pdf*

[**eIDAS**]

Regulation (EU) No. 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC *https://www.eid.as http://data.europa.eu/eli/reg/2014/910/oj*

[**ECC**]

CEN *CEN-TS 15480 / CEN/TC 224 - Personal identification, electronic signature and cards and their related systems and operations*

[**M-COMM**]

ETSI *Mobile Commerce (M-COMM); Mobile Signatures; Business and Functional Requirements*, ETSI Technical Report 102 203 V1.1.1, May 2003

[**ISO 24727**]

Identification Cards — Integrated Circuit Cards Programming Interfaces — Part 1-6, ISO/IEC, International Standards 2008-2014

[**JCA-Names**]

Java™ Cryptography Architecture Standard Algorithm Name Documentation, *http://docs.oracle.com/javase/7/docs/technotes/guides/security/StandardNames.html* Oracle.

[**RFC 3852**]

R. Housley, *Cryptographic Message Syntax (CMS)*, *https://www.ietf.org/rfc/rfc3852.txt* IETF (Internet Engineering Task Force), July 2004.

[**RFC 5246**]

T. Dierks, E. Rescorla, *The Transport Layer Security (TLS) Protocol*, *https://www.ietf.org/rfc/rfc5246.txt* IETF (Internet Engineering Task Force), August 2008.

[**RFC 5280**]

D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, *https://www.ietf.org/rfc/rfc5280.txt* IETF (Internet Engineering Task Force), May 2008.

[**RFC 6931**]

D. Eastlake, *Additional XML Security Uniform Resource Identifiers (URIs)*, *https://www.ietf.org/rfc/rfc6931.txt* IETF (Internet Engineering Task Force), April 2013.

[**RFC 7516**]

M. Jones, J. Hildebrand, *JSON Web Encryption (JWE)*, *https://www.ietf.org/rfc/rfc7516.txt* IETF (Internet Engineering Task Force), May 2015.

[**RFC 7517**]

M. Jones, *JSON Web Key (JWK))*, *https://www.ietf.org/rfc/rfc7517.txt* IETF (Internet Engineering Task Force), May 2015.

Copyright © OASIS Open 2018. All Rights Reserved.

[**PKCS#1 version 2.1**]
Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1, IETF RFC 3447, February 2003. *http://tools.ietf.org/html/rfc3447*

[**PKCS#1 version 2.2**]
PKCS#1 v2.2: RSA Cryptography Standard, RSA Laboratories, October 27, 2012. *http://www.rsa.com/rsalabs/pkcs/files/h11300-wp-pkcs-1v2-2-rsa-cryptography-standard.pdf*

[**PKCS#11-CM**]
PKCS #11 Cryptographic Token Interface Current Mechanisms Specification, Susan Gleeson and Chris Zimman (ed.), *http://docs.oasis-open.org/pkcs11/pkcs11-curr/v2.40/pkcs11-curr-v2.40.pdf* OASIS Standard, December 23, 2014.

[**draft-larmouth-oid-iri-04**]
J. Larmouth, *An IRI/URI Namespace for International Object Identifiers (OIDs)*, *http://tools.ietf.org/id/draft-larmouth-oid-iri-04.txt* IETF (Internet Engineering Task Force) DRAFT, July 24, 2005.

[**XML-Algs**]
W3C: XML Security Algorithm Cross-Reference, *https://www.w3.org/TR/xmlsec-algorithms/* W3C Working Group Note 11, April 2013.

1.5 Namespaces

The structures described in this specification are contained in the schema file which is part of [**LocalSigXSD**]. The xml schema definitions present within this document are copy of the XML schema file and must be considered as informative text, and that in case of discrepancy, definitions within the XML schema prevail. The schema is associated with the following XML namespace:

```
http://docs.oasis-open.org/dss-x/ns/localsig
```

Conventional XML namespace prefixes are used in this document:

- The prefix `xs:` stands for the W3C XML Schema namespace [**XML-Schema**].
- The prefix `ds:` stands for the W3C XML Signature namespace [**XMLSig**].
- The prefix `dss:` stands for the OASIS DSS core namespace [**DSSCore**].
- The prefix `localsig:` stands for the OASIS DSS-X local signature computation profile namespace.

Applications MAY use different namespaces, and MAY use whatever namespace defaulting/scoping conventions they desire, as long as they are compliant with the Namespace in XML specification [**XML-ns**].

1.6 Requirements (Non-Normative)

This section is informative. The overall goal of the local signature computation profile is to extend the OASIS Digital Signature Service (DSS) protocol such that an electronic signature can be created by means of a (qualified) signature-creation device under the direct control of an end user. This section lists the requirements for the local signature computation profile.

- It is possible to use a (qualified) signature-creation device (using protocols such as [ISO/IEC 7816], [**ISO 24727**] and [CEN 15480]) at a different location from the OASIS DSS server.
- It is possible to specify a digest method (algorithm) to be used in the (qualified) signature-creation process.
- It is possible to obtain the digest for a given input documents (and document type) and given digest method (algorithm).

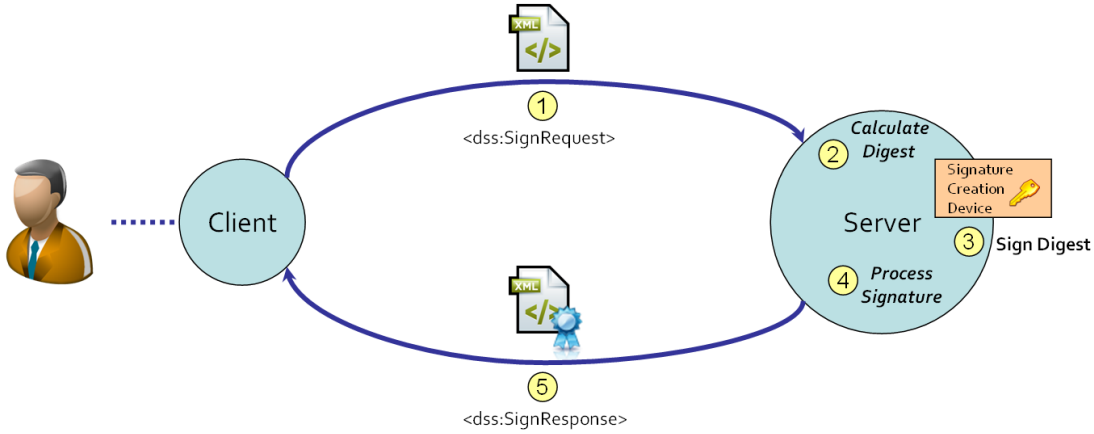
Copyright © OASIS Open 2018. All Rights Reserved.

1.7 Design Rationale (Non-Normative)

1.7.1 Introduction

This section is informative. The DSS protocol assumes a client-server relationship. The client initiates a SignRequest (1) and the server creates the electronic signature for the input document (2, 3 and 4). The resulting electronic signature (and whenever requested the document) is sent back to the client in the SignResponse (5). This is shown in the figure below.

Figure 1. A client and server that implement the OASIS DSS protocol



Note that the signature-creation device (SCD) is (by default) part of the server that implements the OASIS DSS protocol.

Such an architecture is applicable in case the end users do not own a signature-creation device (SCD). However, large-scale signing token deployments increase the use of (qualified) signature-creation devices that are owned by and in the proximity of an end user. Examples are:

- National eID cards or European Citizen Card [**ECC**], containing a qualified signature-creation device ([**eIDAS**]).
- Mobile devices, where the SIM card can be used as a qualified signature-creation device (QSCD) [**M-COMM**].

In such scenarios it is still interesting to keep a OASIS DSS in place for several reasons:

- Despite the fact that every person owns a token with signing capability, he/she might not have the appropriate software installed on the system for the creation of electronic signatures. It might be easier to maintain a lightweight solution, for instance by means of an applet, instead of a full blown token middleware that has to be installed on every participating client's system. The diversity among the client platforms is also easier to manage from a centralized service instead of distributing token middleware to all participating client systems. Furthermore, managing the configuration of the signature policy to be used for the creation of electronic signatures within a certain context might be easier using a centralized service.
- Transforming a business workflow that is based on paper-documents into a digital equivalent, requires a sub-process for the creation (and validation) of electronic signatures on digital documents. Such a sub-process might be available as a service that can be easily integrated with a business application.
- From a technical point of view it might be easier to maintain different OASIS Digital Signature Services, each specialized in handling a specific signature and token types. E.g. tokens per vendor, or per country.

Copyright © OASIS Open 2018. All Rights Reserved.

This profile extends the OASIS DSS protocol such that end users can present their own (qualified) signature-creation device. Consequently, the device itself is not located at the server that implements the DSS protocol.

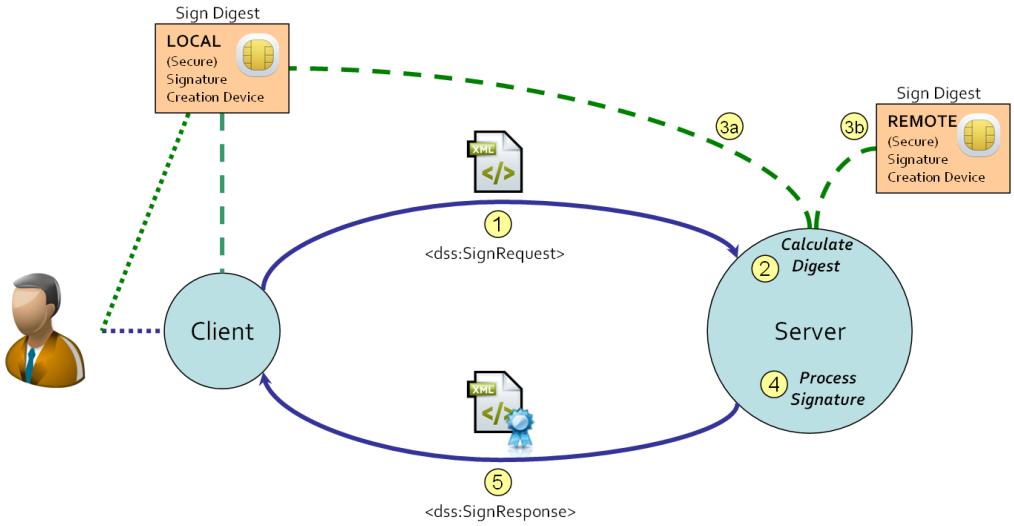
Although the (qualified) signature-creation device is under the control of the end user, the location of the device can be local or remote. The following terminology is used:

- if the (qualified) signature-creation device is in the proximity of the end user, it is referred to as a *local* (qualified) signature-creation device, abbreviated to *LSCD*;
- if the (qualified) signature-creation device is owned by the end user and stored at a remote site (still under the direct control of the end user, but not in the proximity of the end user), it is referred to as a *remote* (qualified) signature-creation device, abbreviated to *RSCD*.

Note that from the viewpoint of the Digital Signature Service, both LSCD and RSCD have to be treated as remote devices.

The following diagram visualizes the (logical) relationship between the LSCD (or RSCD) and the Digital Signature Service.

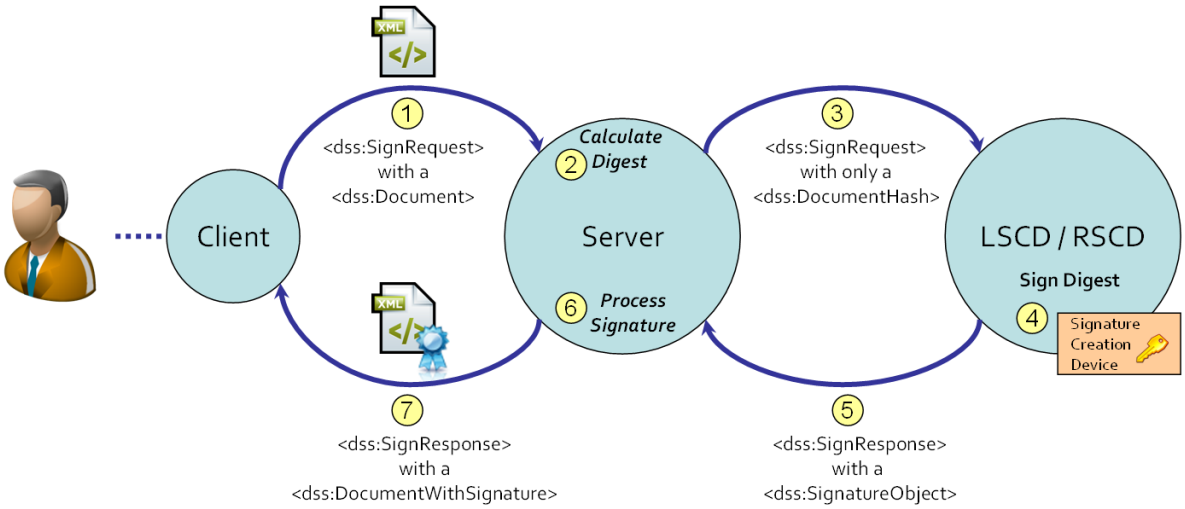
Figure 2. Local and remote device for signature creation



To re-use the existing DSS core protocol functionality as much as possible, the Digital Signature Service can delegate the computation of the digital signature value as depicted below. It assumes that the LSCD or RSCD implements a basic DSS protocol to serve the sign requests of a digest; only the basic signature operations are assumed (no on-line certificate validation, timestamp requests, etcetera).

Copyright © OASIS Open 2018. All Rights Reserved.

Figure 3. Delegation of the signature creation to the LSCD or RSCD



In general, the LSCD or RSCD have limited software and performance capabilities and hence will be supported by a OASIS DSS compliant service to handle the complexities of the (qualified) signature-creation, electronic signature processing and document manipulation. Furthermore, in general, the LSCD or RSCD might not be accessible through a DSS core protocol.

The LSCD or RSCD will be able to serve a request to sign a given digest. It is assumed that the interface to the actual (S)SCD is accessed through one of the possible standards, such as the APDU (ISO 7816) or the IFD-Client (ISO/IEC 24727 / CEN 15480) standard. It shows that the profile does not depend on the actual interface of the (S)SCD. It is assumed that there will be some middleware that abstracts from the vendor-specific implementation of the (S)SCD.

The introduction of an LSCD or RSCD can have different use cases, depending on the technology as well. For instance, is the LSCD integrated in a mobile device and is the DSS client running on different platform? Or is the LSCD connected to the DSS client? The next section present a number of use cases that are considered for this profile.

1.7.2 Use Cases

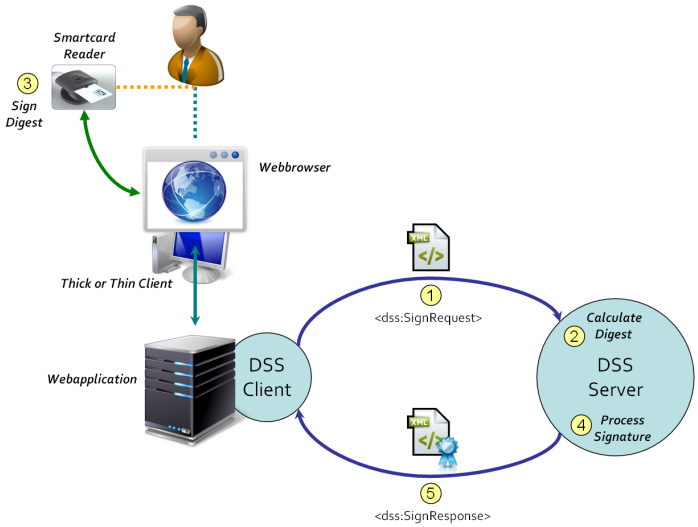
The basic sequence within the use cases is as follows. A user initiates a signing action by means of a client application. The client application initiates a `SignRequest` to the Digital Signature Service (1). The Digital Signature Service calculates the digest (2), obtains a digital signature value by some mechanism (3), creates the electronic signature from the digital signature value and processes it according to the request (4). The resulting electronic signature (and whenever requested the document) is sent back to the client in the SignResponse (5).

1.7.2.1 Use Case 1

This use case assumes a thick or thin client platform. The (qualified) signature-creation device can be a smartcard connected to the client platform by means of a smartcard reader. A web browser is used to let the end user access the web application, for instance a Document Management System (DMS). The web application (for instance a DMS) has implemented the DSS protocol to allow its users to sign documents. To sign (a) document(s), the user selects a document from wihin the web application (for instance a DMS) and the web application sends the document(s) to the DSS server; the DSS server obtains (somehow) the digital signature value computed by the smartcard. Note that the DSS server cannot access the smartcard reader without additional mechanisms; proposals to solve this are presented in Section Section 1.7.3, “Proposed Solutions”.

Copyright © OASIS Open 2018. All Rights Reserved.

Figure 4. A smartcard connected to a desktop

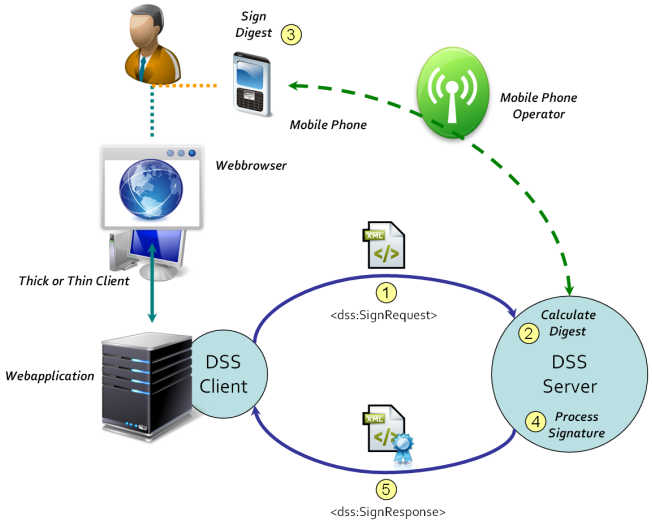


1.7.2.2 Use Case 2

This use case assumes the use of a mobile phone that contains a (qualified) signature-creation device. The mobile phone is connected to the mobile operator infrastructure. A web browser is used to let the end user access the client application. The client application has implemented the DSS protocol. To sign (a) document(s), the client sends the document(s) to the DSS server; the DSS server obtains the digital signature value computed by the smartcard in the mobile phone. The DSS server connects to the mobile phone which requests the user to sign (the provided digest).

This use case resembles the ETSI standard for Mobile Commerce (M-COMM) or Mobile Signature Service [**M-COMM**].

Figure 5. A mobile phone for signature creation

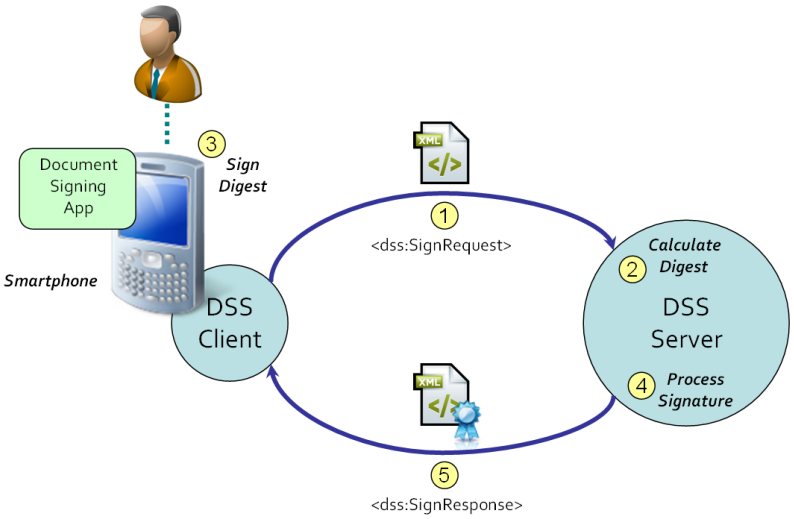


1.7.2.3 Use Case 3

This use case assumes the use of a smartphone that contains a (qualified) signature-creation device. The smartphone contains an app to sign (a) document(s), although the actual electronic signature processing functionality is delegated to a DSS server. The app has implemented the DSS protocol and the use of the (qualified) signature-creation device in the smartphone. Note that the DSS server cannot access the (qualified) signature-creation device without additional mechanisms; proposals to solve this are presented in Section Section 1.7.3, “Proposed Solutions”.

Copyright © OASIS Open 2018. All Rights Reserved.

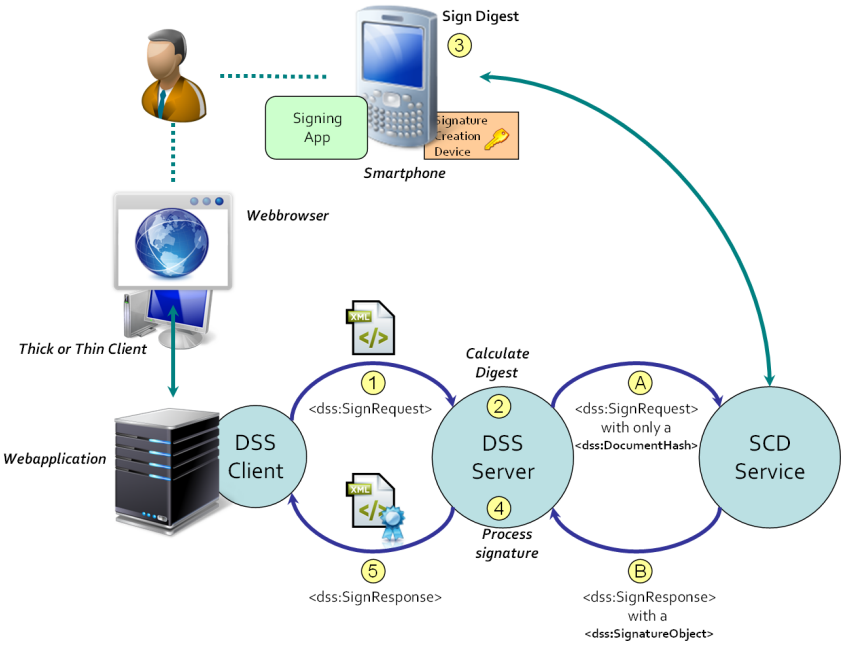
Figure 6. A smartphone as a DSS client



1.7.2.4 Use Case 4

This use case is a generalisation of use case 2. It assumes the use of a mobile device that contains a (qualified) signature-creation device, in the proximity of the end user. If the mobile device does not contain a (qualified) signature-creation device it is assumed that it is located at the "SCD Service". The mobile device can have more capabilities than a mobile phone. For instance, a smartphone with an internet connection and an app to interact with the "SCD Service".

Figure 7. Signature creation delegated to another service



To sign (a) document(s), the client sends the document(s) to the DSS server. The DSS server delegates the actual computation of the digital signature value to another service: the digest is sent to the "SCD Service", with possibly (but not necessarily) a <dss:SignRequest> (A). The "SCD Service" is able to deliver a digital signature value, based on the interaction with the mobile device by means of a certain protocol.

If the smartphone contains a (qualified) signature-creation device, it receives or retrieves the digest from the "SCD Service" and computes the digital signature value for the digest (3), after which the digital

Copyright © OASIS Open 2018. All Rights Reserved.

signature value is returned to the "SCD Service". The "SCD Service" responds with possibly (but not necessarily) a <dss:SignResponse> (B) to the DSS server.

Note that some secure correlation is required between the initial user request and the interaction with the smartphone.

This use case can make use of the ETSI standard for Mobile Commerce (M-COMM) or Mobile Signature Service [**M-COMM**]

1.7.3 Proposed Solutions

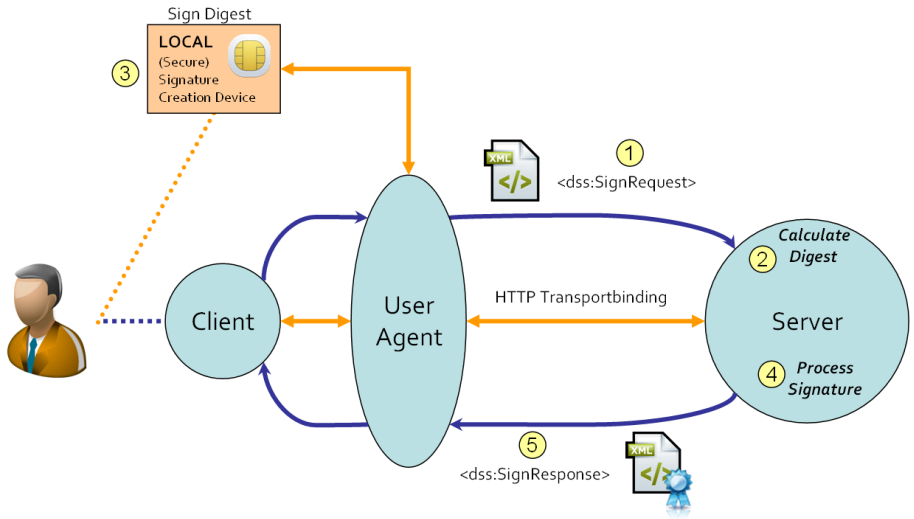
This section describes three possible technical solutions for different use cases.

- A User Agent at the client is introduced for use case 1 to access the (qualified) signature-creation device (connected to the client platform) for the computation of the digital signature value. This case is similar to the special case in which the client is a web-application and the User Agent is a localhost-based ChipGateway.
- A two-step approach is introduced for use case 1 and use case 3: the Digital Signature Service returns the document digest in the first step. After the client has computed the digital signature value for the document digest, it is returned to the Digital Signature Service in the second step (to enrich the digital signature value, for instance with a time-stamp and/or revocation information).
- A third-party is introduced for use case 2 and use case 4: the Digital Signature Service delegates the computation of the digital signature value to a third-party. The third-party has some means to contact the (secure) signature-creation device of the end user.

1.7.3.1 Introduction of a User Agent

To provide the Digital Signature Service access to a LSCD, a User Agent is introduced. The User Agent will provide access to the LSCD. The DSS client communicates with the DSS server via the User Agent; there is no direct communication between the DSS client and DSS server. The figure below illustrates how this is done.

Figure 8. A LSCD used by the DSS server



The order of actions is as follows:

- The DSS client sends the SignRequest to the User Agent.
- The User Agent sends the SignRequest to the DSS server (1). The DSS server has obtained a session (connection) with the User Agent.

Copyright © OASIS Open 2018. All Rights Reserved.

- The DSS server calculates the digest (2) for the input document.
- In order to have the digest signed, an interaction between the User Agent and the LSCD (3) is required. This interaction will be initated by the DSS server, by means of the session (connection) that has been obtained earlier. This profile does not specify how the User Agent obtains the digital signature value: it is left to the implementers.
- Once the digital signature value has been obtained, the DSS server creates and processes (4) the electronic signature according the the request, and the SignResponse is built. (If requested, the electronic signature is embedded into the document.)
- The DSS server sends the SignResponse to the User Agent (5)
- The User Agent sends the SignResponse to the DSS client.

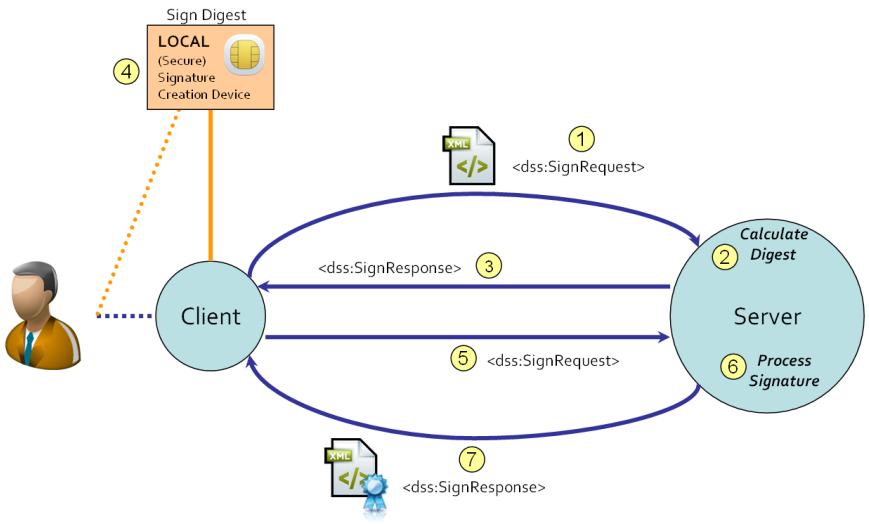
For achieving this User Agent approach a transport binding needs to be introduced:

- A HTTP transport binding is defined for the use of a User Agent and HTML forms. The HTML form initiates a HTTP POST (from within the User Agent) to the specified action location (either the DSS server or the DSS client). This transport binding is referred to as a *WEB FORM transport binding* and can be found in Section 4.1, "WEB FORM Transport Binding".

1.7.3.2 Introduction of a Two-Step Approach

The OASIS DSS protocol could be split into two related request/response pairs, such that the client can obtain a digest of the document in the FIRST request/response. The client is able to access the LSCD. The client interacts with the LSCD to compute the digital signature value and sends it back to the DSS server by means of the SECOND request/response.

Figure 9. An LSCD used by the DSS client.



The order of actions is as follows:

- The first step consists of a SignRequest (1) - SignResponse (3) pair where the SignResponse only contains the digest, calculated by the Digital Signature Service (2).
- This digest is used by the client application to compute the digital signature value (4) by means of the (secure) signature-creation device at the client.
- The second step consist of a SignRequest (5) - SignResponse (7) pair where the SignRequest contains the digital signature value. The Digital Signature Service builts and processes (6) the

Copyright © OASIS Open 2018. All Rights Reserved.

electronic signature based on the digital signature value and the request received. The result is sent to the client with the SignResponse (7).

Note that no assumptions are made about the actual connection or communication between the client and LSCD. (Although the core DSS protocol could be used to send a `<dss:DocumentHash>` to the LSCD and to obtain a `<SignatureObject>`.) The only assumption is that the client that implements the (client-side of the) DSS protocol is capable of using the LSCD.

For achieving this two-step approach some OptionalInputs and OptionalOutputs elements need to be introduced for the SignRequest or SignResponse:

1. An optional output `<ds:DocumentHash>` element in the SignResponse. After the DSS server has received a SignRequest it calculates the digest of the document. The SignResponse contains the `<dss:DocumentHash>` value.

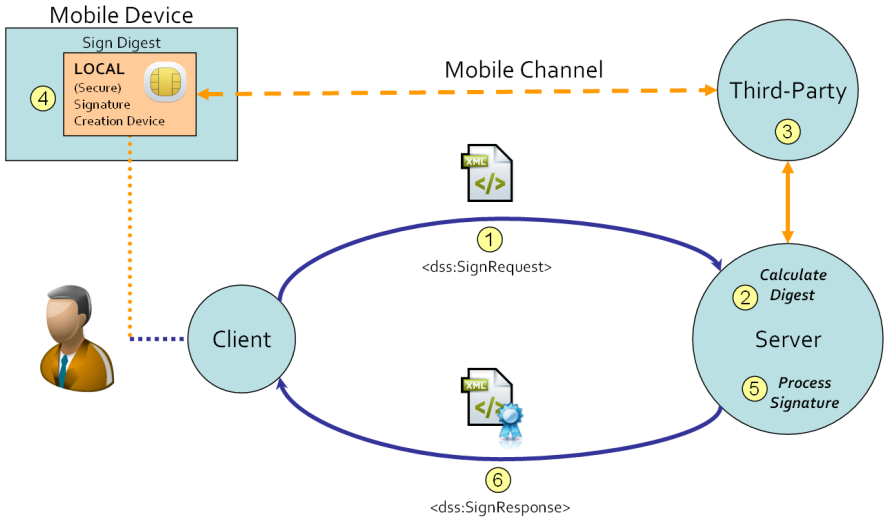
2. An optional input `<dss:SignatureObject>` element in the second SignRequest. After the DSS client has received the digest as a result of the first SignRequest, the LSCD computes the digital signature value and the client incorporates it to the aforementioned SignRequest using the `<dss:SignatureObject>` element. The DSS server will perform the necessary operations for building the final electronic signature, for incorporating it within the corresponding document if required, and for returning it within the corresponding SignResponse.

3. An optional output (resp. input) `<localsig:CorrelationID>` element in the first SignResponse (resp. second SignRequest). The correlation identifier is used to relate the first and second SignRequest/SignResponse pairs.

1.7.3.3 Introduction of a Third-Party

The Digital Signature Service delegates the computation of the digital signature value to a third-party for users with a mobile device that contains a (secure) signature creation device (in that case, the mobile device contains the LSCD). If the mobile device does not contain a (secure) signature-creation device it is assumed that the third-party provides access to the RSCD on behalf of the end user by means of the mobile device. See for instance Section 1.7.2.2, “Use Case 2” and Section 1.7.2.4, “Use Case 4” .

Figure 10. Signature creation delegated to a third-party.



Because the end user has to be assured that it signs the correct request, an additional mechanism is introduced: a challenge code. The challenge code is a kind of 'one-time-password': it is different for every session. The challenge code will be shown at the client screen as well as the screen of the mobile device (the DSS server sends the challenge code to the third-party). The end user compares the codes and if they are the same confirms the sign request.

Copyright © OASIS Open 2018. All Rights Reserved.

Note. The challenge code could even be used for a kind of challenge-response authentication by the Digital Signature Service. If the SignRequest also contains a response code, it is expected that the end user enters that code at the mobile device. The electronic signature is only created and processed (by the Digital Signature Service) if the response codes match. Note that the challenge-response does not protect the end user from misuse of the signature by the third-party. (But the signature is bound to the document by means of a unique hash value, making it useless for other documents.)

The DSS client is able to create the challenge (and optionally a response) code, unique for every session, and will display the code(s) onto the screen of the client.

The order of actions is as follows:

- The client sends a SignRequest, which also contains a challenge code, to the Digital Signature Service (1).
- The Digital Signature Service calculates the digest of the document (2) and delegates the computation of the digital signature value to a third-party (3), for instance a Mobile Signature Service.
- The third-party sends the request to the mobile device of the end user. The end user has to compare the challenge code with the code that is shown on the mobile device (as part of the information that is sent from the third-party to the mobile device). Once the end user has confirmed the code, the mobile device computes the digital signature value (4) and sends it back to the third-party. The third-party returns the digital signature value to the Digital Signature Service.
- The Digital Signature Service creates and processes the electronic signature (5) based on the digital signature value and the request received. The resulting electronic signature is returned to the client in the SignResponse (6).

The protocol between the Digital Signature Service and third-party is not specified by this profile, but the DSS protocol can be used for this as well, see for instance Figure 3, “Delegation of the signature creation to the LSCD or RSCD”. (It is assumed that the challenge code is included in the request to the third-party.)

Note that information is required about the identity of the end user to enable the third-party to communicate with the appropriate mobile device. It is assumed that the element `<dss:ClaimedIdentity>` is sufficient for this purpose.

For achieving this third-party approach some OptionalInputs elements need to be introduced for the SignRequest:

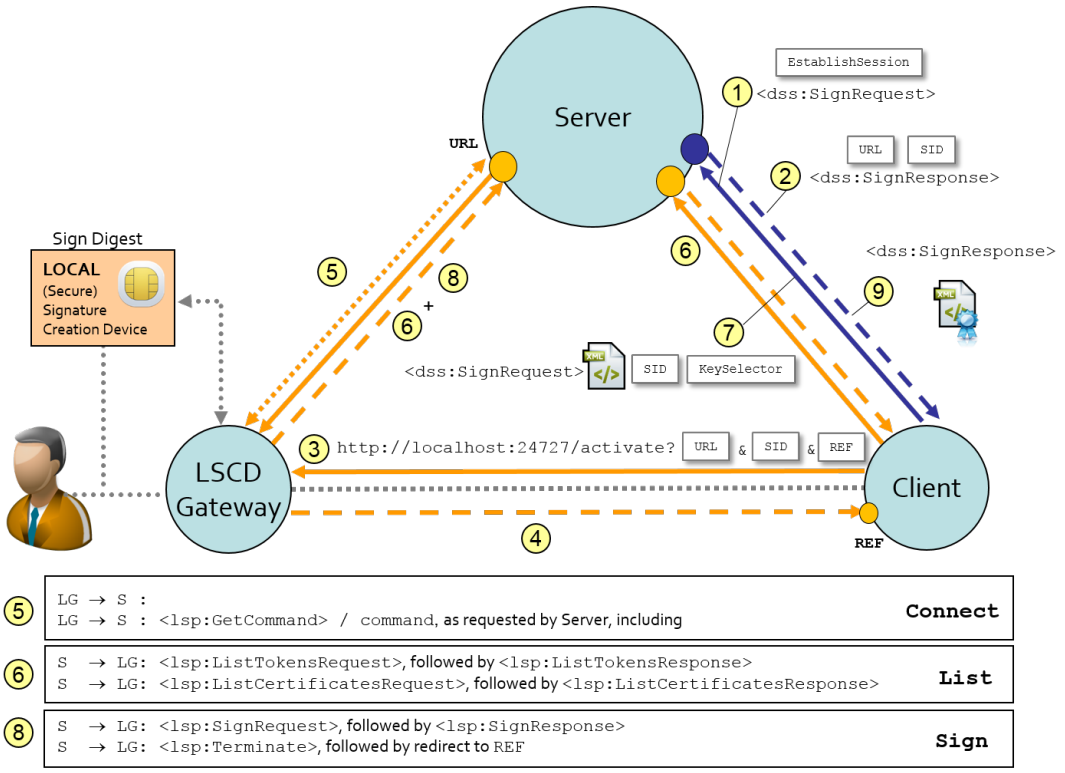
1. An optional input `<localsig:ChallengeCode>` element in the SignRequest. The challenge code is shown at the screen of the mobile device as well as the screen of the client.
2. An optional input `<localsig:ResponseCode>` element in the SignRequest. The response code is entered at the mobile device after which it is sent back to the Digital Signature Service by the third-party.

1.7.3.4 Introduction of a ChipGateway

The ChipGateway Protocol defined in the present document has been developed in a joint approach by LuxTrust S.A. (Luxembourg) and ecsec GmbH (Germany) based on the “eID-Client” specification [**BSI-TR-03124**] of the German Federal Office for Information Security and related standards. The protocol specification [**CGW-Contribution**] was contributed to OASIS Digital Signature Services eXtended (DSS-X) TC to foster the development of an open ecosystem for trust services for electronic transactions as envisioned by the eIDAS-Regulation (EU) Nr. 910/2014 [**eIDAS**].

Copyright © OASIS Open 2018. All Rights Reserved.

Figure 11. Signature creation via ChipGateway



The order of actions in Figure 11, “Signature creation via ChipGateway” is as follows:

1. In the first step, the Client initiates the protocol flow by sending a `SignRequest` to the Server, which contains the optional input `EstablishSession` as specified in Section 3.4, “ChipGateway”.

2. The Server returns a `SignResponse`, which contains the `ServerAddress` (depicted as `URL` in Figure 11, “Signature creation via ChipGateway”) and a `SessionIdentifier` (depicted as `SID` in Figure 11, “Signature creation via ChipGateway”).

3. In the this step the Client activates the ChipGateway and tells it to establish a connection to the Server at `URL` in order to perform the ChipGateway Protocol. This activation may be performed by sending an activation link via the localhost-interface of the ChipGateway via a `GET`-call with corresponding query-parameters. See Section 3.4.3.1, “ChipGateway Activation via localhost link” for details.

4. If the activation call was well formed, the ChipGateway immediately returns and performs a redirect to the refresh address (depicted as `REF` in Figure 11, “Signature creation via ChipGateway”).

5. Now the ChipGateway and the Server establish a connection using `HelloRequest`, `HelloResponse` and `GetCommand`, as specified in Section 3.4, “ChipGateway” .

6. Using the established connection, the Client may send `ListTokensRequest` and `ListCertificatesRequest` via the Server to the ChipGateway in order to determine the available signature tokens and certificates.

7. Next the Client sends another `SignRequest`, which contains the document to be signed, the `SessionIdentifier` (`SID`) obtained in step (2) and the `KeySelector`, which specifies the signing key.

8. The Server sends a `SignRequest` with the data to be signed to the ChipGateway and receives back the raw cryptographic signature, which may need to be inserted into the signed document.

Copyright © OASIS Open 2018. All Rights Reserved.

9. The Server finally answers with a `SignResponse`, which in case of success contains the signed document.

Copyright © OASIS Open 2018. All Rights Reserved.

2 Profile Features

2.1 Identifier

The attribute `Profile` MUST have the value:

```
http://docs.oasis-open.org/dss-x/ns/localsig
```

2.2 Scope

This profile extends the OASIS DSS signing functionality [**DSSCore**] such that end users can bring (use) their own (secure) signature-creation device. Such a device is referenced as *local (secure) signature-creation device* and is abbreviated to LSCD.

(This profile does not explore the use of an RSCD (see Figure 2, “Local and remote device for signature creation”).

2.3 Relationship to Other Profiles

The profile in this document is based on the [**DSSCore**].

This profile provides means for the explicit management of local signature computations and other existing profiles, and as such, it may be used in conjunction with these specifications.

Copyright © OASIS Open 2018. All Rights Reserved.

3 Profile of Signing Protocol

3.1 User Agent

This clause enables the DSS client to obtain an electronic signature or signed document using a (secure) signature-creation device from an end user in a single interaction with the Digital Signature Service by means of a HTTP User Agent. The HTTP User Agent is responsible for the creation of the digital signature value using a (secure) signature-creation device at the client.

The HTTP User Agent MUST be able to access the (secure) signature-creation device to create the digital signature value.

A transport binding is required that creates a session between the Digital Signature Service and the HTTP User Agent agent, see Section 4.1, “WEB FORM Transport Binding”.

The Digital Signature Service uses the session with the HTTP User Agent to obtain the digital signature value from the HTTP User Agent. How this functionality is implemented is not specified and is not part of this profile: it depends on the capabilities of the HTTP User Agent and the client platform (e.g. through the use of a Java Applet in the web browser and a PKCS#11 device driver at the client platform).

3.1.1 Element <dss:SignRequest>

This clause profiles the <dss:SignRequest> element.

The [**DSSCore**] attribute `Profile` (Section 3.1) MUST have the value:

```
http://docs.oasis-open.org/dss-x/ns/localsig
```

Whenever needed, the [**DSSCore**] element `<dss:AdditionalProfile>` MAY be used to specify additional profiles for the proper creation of the resulting document(s) by the Digital Signature Service. The interpretation of the additional profile(s) is determined by the corresponding specification(s).

3.1.1.1 Element <dss:OptionalInputs>

This clause profiles the use of Optional Input elements.

3.1.1.1.1 Element <dss:ServicePolicy>

The [**DSSCore**] element `<dss:ServicePolicy>` (Section 2.8.1) MUST be present and MUST have the value:

```
http://docs.oasis-open.org/dss-x/ns/localsig/user-agent
```

This policy instructs the Digital Signature Service to initiate the interaction with the User Agent to access the (secure) signature-creation device of the end user.

3.1.1.1.2 Element <ds:DigestMethod>

The [**XMLSig**] element `<ds:DigestMethod>` (Section 4.3.3.5) MAY be present to specify which digest method has to be used by the Digital Signature Service.

3.1.2 Element <dss:SignResponse>

This clause profiles the <dss:SignResponse> element.

The `<dss:SignResponse>` contains (in according to the [**DSSCore**] specification, Section 3.2) either the electronic signature or the signed document, or an error message.

Copyright © OASIS Open 2018. All Rights Reserved.

3.1.2.1 Element <dss:Result>

In case the end user cancelled the signing operation, the Digital Signature Service MUST return a `<dss:ResultMajor>` with the value `RequesterError` and a `<dss:ResultMinor>` with the value:

```
urn:oasis:names:tc:dss-x:profiles:localsig:user-cancelled
```

In case the Digital Signature Service detects a problem with the client runtime environment, the service returns a `<dss:ResultMajor>` with the value `RequesterError` and a `<dss:ResultMinor>` with the value:

```
urn:oasis:names:tc:dss-x:profiles:localsig:client-runtime-error
```

3.2 Two-Step Approach

This clause enables the client to obtain a signed document in two steps; the client MUST create of the digital signature value using a (secure) signature-creation device based on the document digest that is received from the Digital Signature Service. The interaction with the Digital Signature Service involves two `<dss:SignRequest>` requests:

- The FIRST `<dss:SignRequest>` MUST initiate the process by sending the document to the Digital Signature Service. (This is the *first step* in the two-step approach.) The Digital Signature Service MUST respond with the FIRST `<dss:SignResponse>`: it MUST contain the digest of the document. The Digital Signature Service MUST postpone processing of the document until the `<dss:SignatureObject>` is received in the SECOND `<dss:SignRequest>`.
- The SECOND `<dss:SignRequest>` MUST be sent to finalize the process by sending the `<dss:SignatureObject>` to the Digital Signature Service. (This is the *second step* in the two-step approach.) The Digital Signature Service MUST resume processing and MUST respond with the SECOND (and final) `<dss:SignResponse>` which contains the electronic signature or signed document, depending what has been requested.

3.2.1 Element <dss:SignRequest>

This clause profiles the `<dss:SignRequest>` element.

The [**DSSCore**] attribute `Profile` (Section 3.1) MUST have the value:

```
http://docs.oasis-open.org/dss-x/ns/localsig
```

Whenever needed, the [**DSSCore**] element `<dss:AdditionalProfile>` MAY be used to specify additional profiles for the proper creation of the resulting document(s) by the Digital Signature Service. The interpretation of the additional profile(s) is determined by the corresponding specification(s).

The [**DSSCore**] element `<dss:InputDocuments>` (Section 2.4) MUST be present in the FIRST request and MAY be present in the SECOND request. If present in the SECOND request, it MUST contain the document as used in the FIRST request.

3.2.1.1 Element <dss:OptionalInputs>

This clause profiles Optional Input elements.

3.2.1.1.1 Element <dss:ServicePolicy>

The [**DSSCore**] element `<dss:ServicePolicy>` (Section 2.8.1) MUST be present and MUST have the value:

Copyright © OASIS Open 2018. All Rights Reserved.

```
http://docs.oasis-open.org/dss-x/ns/localsig/two-step-approach
```

This policy instructs the Digital Signature Service that two request/response pairs are used to obtain the digital signature value from the (secure) signature-creation device of the end user.

3.2.1.1.2 Element <localsig:RequestDocumentHash>

The new element `<localsig:RequestDocumentHash>` MUST be present in the FIRST request. The type of the element is defined as follows (taken from [**LocalSigXSD**]):

```
<xs:element name="RequestDocumentHash">
   <xs:complexType>
      <xs:sequence>
         <xs:element
              minOccurs="0" maxOccurs="1"
              ref="ds:DigestMethod"/>
      </xs:sequence>
      <xs:attribute
              name="MaintainRequestState"
              use="optional"
              type="xs:boolean"/>
   </xs:complexType>
</xs:element>
```

The element `<localsig:RequestDocumentHash>` instructs the Digital Signature Service to return the digest of the document and to postpone further processing.

The attribute `MaintainRequestState` MAY be used to instruct the Digital Signature Service to maintain the state of the request, until the SECOND request is received or a certain predefined timeout has been reached (the timeout is determined by the Digital Signature Service). If the attribute is not specified, the assumed value is `false`. If the attribute `MaintainRequestState` is not used or `false`, then the SECOND request MUST contain the same document as specified in the FIRST request.

The [**XMLSig**] element `<ds:DigestMethod>` (Section 4.3.3.5) MAY be used to instruct the Digital Signature Service to use the specified type of digest method. If no method is specified, the Digital Signature Service will use a default digest method.

Note. The state is only useful (i) in case the document digest cannot be easily re-created, for instance if a signature has to be incorporated into a PDF document, or (ii) the document has to be transferred only once.

3.2.1.1.3 Element <dss:SignatureObject>

The [**DSSCore**] element `<dss:SignatureObject>` (Section 2.5) MUST be used in the SECOND request to provide the digital signature value, obtained from the (secure) signature-creation device at the client. The Digital Signature Service creates and processes the electronic signature based on the `<dss:SignatureObject>` and the request.

A `<dss:Base64Signature>` element MUST be used for the digital signature value together with a specified value for the `type` attribute. For OID's a urn according to [**RFC 3061**] and [**draft-larmouth-oid-iri-04**] MAY be used.

3.2.1.1.4 Element <localsig:CorrelationID>

The new element `<localsig:CorrelationID>` MUST only be used in the SECOND request if and only if the FIRST response contained the element `<localsig:CorrelationID>`. (Their value MUST be the same.)

Copyright © OASIS Open 2018. All Rights Reserved.

The type of the element is defined as follows (taken from [LocalSigXSD]):

```
<xs:element name="CorrelationID" type="xs:NCName"/>
```

3.2.2 Element <dss:SignResponse>

This clause profiles the <dss:SignResponse> element.

3.2.2.1 Element <dss:Result>

In response to a successful processing of a <dss:SignRequest> that specified the element <localsig:RequestDocumentHash>, the <dss:ResultMajor> contains the value Success and the <dss:ResultMinor> the value:

```
   urn:oasis:names:tc:dss-x:profiles:localsig:document-hash
```

In response to a successful processing of a <dss:SignRequest> that specified the element <localsig:SignatureObject>, the <dss:ResultMajor> and <dss:ResultMinor> follow the [DSSCore] specification, Section 2.6.

3.2.2.2 Element <dss:OptionalOutputs>

This profile defines the Optional Output elements.

3.2.2.2.1 Element <dss:DocumentHash>

The [DSSCore] element <dss:DocumentHash> (Section 2.4.4) MUST be returned in response to a <dss:SignRequest> that specified the element <localsig:RequestDocumentHash>. (Note that [DSSCore] specifies the use of this element as part of the OptionalInputs.)

The client uses the document digest for further processing by the (secure) signature-creation device.

3.2.2.2.2 Element <localsig:CorrelationID>

The new element <localsig:CorrelationID> MUST be returned in response to a <dss:SignRequest> if and only if the element <localsig:RequestDocumentHash> element was present within the SignRequest and its MaintainRequestState attribute was set to true. The type of the element is defined as follows (taken from [LocalSigXSD]):

```
<xs:element name="CorrelationID" type="xs:NCName"/>
```

The Digital Signature Service will generate a suitable value on its own behalf so that a client can refer to the state of its FIRST request.

The client MUST use this value in the SECOND request to refer to this state.

3.3 Third Party

This clause enables the client to obtain an electronic signature (or signed document whenever requested) from the Digital Signature Service.

3.3.1 Element <dss:SignRequest>

This clause profiles the <dss:SignRequest> element.

Copyright © OASIS Open 2018. All Rights Reserved.

The [DSSCore] attribute `Profile` (Section 3.1) MUST have the value:

```
http://docs.oasis-open.org/dss-x/ns/localsig
```

Whenever needed, the [DSSCore] element `<dss:AdditionalProfile>` MAY be used to specify additional profiles for the proper creation of the resulting document(s) by the Digital Signature Service. The interpretation of the additional profile(s) is determined by the corresponding specification(s).

3.3.1.1 Element <dss:OptionalInputs>

This clause profiles the Optional Inputs elements.

3.3.1.1.1 Element <dss:ServicePolicy>

The [DSSCore] element `<dss:ServicePolicy>` (Section 2.8.1) MUST be present and MUST have the value:

```
http://docs.oasis-open.org/dss-x/ns/localsig/delegation
```

This policy instructs the Digital Signature Service to delegate the creation of the signature to a third-party.

3.3.1.1.2 Element <ds:DigestMethod>

The [DSSCore] element `<ds:DigestMethod>` (Section 4.3.3.5) MAY be present to specify which digest method has to be used by the Digital Signature Service.

3.3.1.1.3 Element <localsig:ChallengeCode>

The new element `<localsig:ChallengeCode>` MUST be present in the request and MUST have a random value that can easily be read by a person. The type of the element is defined as follows (taken from [LocalSigXSD]):

```
<xs:element name="ChallengeCode" type="xs:NCName"/>
```

The client has to show the challenge code to the end user. The end user MUST be able to compare the code with the value that is shown on the mobile device before it confirms the computation of the digital signature value by the mobile device (the challenge code is sent to the third-party as well).

3.3.1.1.4 Element <localsig:ResponseCode>

The new element `<localsig:ResponseCode>` MAY be present in the request. If present it MUST have a random value that can easily be read and entered by a person. The type of the element is defined as follows (taken from [LocalSigXSD]):

```
<xs:element name="ResponseCode" type="xs:NCName"/>
```

The client has to show the response code to the end user. The end user MUST be able to enter the response code at the mobile device before it confirms the computation of the digital signature value by the mobile device; the third-party MUST return the response code to the Digital Signature Service. The Digital Signature Service only creates the requested electronic signature if the response code matches the value that was given by the client in the request.

Copyright © OASIS Open 2018. All Rights Reserved.

3.3.2 Element <dss:SignResponse>

This clause profiles the `<dss:SignResponse>` element.

The `<dss:SignResponse>` contains (in according to the [DSSCore] specification, Section 3.2) either the electronic signature or the signed document, or an error message.

3.3.2.1 Element <dss:Result>

If the signature creation is cancelled by the end user the `<dss:ResultMajor>` contains the value for ResponderError and the `<dss:ResultMinor>` the value:

```
urn:oasis:names:tc:dss-x:profiles:localsig:user-cancelled
```

If the third-party is not able to handle the signature creation the `<dss:ResultMajor>` contains the value for ResponderError and the `<dss:ResultMinor>` the value:

```
urn:oasis:names:tc:dss-x:profiles:localsig:delegation-failed
```

If the response code that is returned by the third-party does not correspond to the value that is provided in the request, the `<dss:ResultMajor>` contains the value for ResponderError and the `<dss:ResultMinor>` the value:

```
urn:oasis:names:tc:dss-x:profiles:localsig:incorrect-responsecode
```

3.4 ChipGateway

This clause enables the Client to obtain an electronic signature with the help of the Server and a local ChipGateway. To allow an efficient protocol flow the messages exchanged in the ChipGateway Protocol are JSON-based.

3.4.1 Element <dss:SignRequest>

This clause profiles the `<dss:SignRequest>` element.

The [DSSCore] attribute `Profile` (Section 3.1) MUST have the value:

```
http://docs.oasis-open.org/dss-x/ns/localsig
```

3.4.1.1 Element <dss:OptionalInputs>

This clause profiles the `<dss:OptionalInputs>` element.

3.4.1.1.1 Element <dss:ServicePolicy>

The [DSSCore] element `<dss:ServicePolicy>` (Section 2.8.1) MUST be present in any ChipGateway-specific call and MUST have the value:

```
http://docs.oasis-open.org/dss-x/ns/localsig/chipgateway
```

This policy instructs the Digital Signature Service that the ChipGateway protocol is used to obtain the digital signature value from the (qualified) signature-creation device of the end user.

3.4.1.1.2 Element <cg:EstablishSession>

The following XML schema snippet defines the `EstablishSession` element:

Copyright © OASIS Open 2018. All Rights Reserved.

An empty element can usually be omitted…

```
<element name="EstablishSession"/>
```

Including the EstablishSession element within OptionalInputs will yield the creation of a new session, for which the corresponding SessionIdentifier and ServerAddress will be returned within OptionalOutputs.

3.4.1.1.3 Element <cg:TerminateSession>

The following XML schema snippet defines the TerminateSession element:

```
<element name="TerminateSession" type="string"/>
```

Including the TerminateSession element within OptionalInputs will proactively terminate the corresponding session.

3.4.1.1.4 Element <cg:SessionIdentifier>

The following XML schema snippet defines the SessionIdentifier element:

```
<element name="SessionIdentifier" type="string"/>
```

In order to address a specific session, which has been established with EstablishSession, the SessionIdentifier element is included within OptionalInputs.

3.4.2 Element <dss:SignResponse>

This clause profiles the <dss:SignResponse> element.

The <dss:SignResponse> contains (in accordance with the [**DSSCore**] specification, Section 3.2) either the electronic signature, the signed document, or an error message.

3.4.2.1 Element <dss:Result>

If the signature creation is cancelled by the end user the <dss:ResultMajor> contains the value for ResponderError and the <dss:ResultMinor> contains the value:

```
urn:oasis:names:tc:dss-x:profiles:localsig:user-cancelled
```

If the provided session identifier within SessionIdentifier or TerminateSessionis unknown, the <dss:ResultMajor> contains the value for ResponderError and the <dss:ResultMinor> contains the value:

```
urn:oasis:names:tc:dss-x:profiles:localsig:unknown-sessionidentifier
```

3.4.2.2 Element <dss:OptionalOutputs>

This clause profiles the <dss:OptionalInputs> element.

3.4.2.2.1 Element <cg:SessionIdentifier>

The identifier of a session established with EstablishSession is returned in an SessionIdentifier element, which syntax is specified in Section 3.4.1.1.4, “Element <cg:SessionIdentifier>”, within OptionalInputs.

3.4.2.2.2 Element <cg:ServerAddress>

The following XML schema snippet defines the ServerAdddress element:

```
<element name="ServerAddress" type="anyURI"/>
```

The ServerAddress is the endpoint address of the ChipGateway Server, which is returned after providing the optional input EstablishSession and used for the establishment of the connection

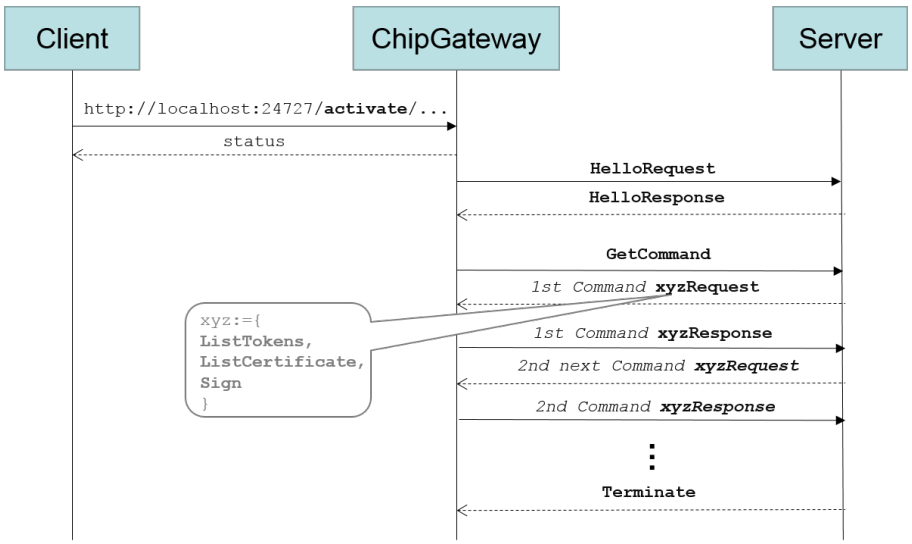
Copyright © OASIS Open 2018. All Rights Reserved.

between the ChipGateway and the ChipGateway-Server, as explained in Section 3.4.3, “ChipGateway Connection Establishment”.

3.4.3 ChipGateway Connection Establishment

The connection establishment between the Client, the ChipGateway and the Server is performed as depicted in Figure 12, “ChipGateway Connection Establishment”.

Figure 12. ChipGateway Connection Establishment



The Client initiates the connection establishment process by providing a web page with an embedded activation link (`http://localhost:24727/activate/...`) on which the user clicks to start the process. The ChipGateway immediately returns to the Client with a status indication (see Section 3.4.3.1, “ChipGateway Activation via localhost link” for details).

Now the ChipGateway establishes the connection to the Server by sending `HelloRequest` as specified in Section 3.4.3.2, “Element HelloRequest”, which is acknowledged by the Server with `HelloResponse` as specified in Section 3.4.3.3, “HelloResponse”.

To conclude the connection establishment process, the ChipGateway finally sends `GetCommand` as specified in Section 3.4.3.4, “GetCommand” to the Server, which is now able to send any of the operational ChipGateway commands (i.e. `ListTokensRequest`, `ListCertificatesRequest`, `SignRequest`) to the ChipGateway before it stops the process with `Terminate`. The ChipGateway responds to each operational request with the corresponding response (i.e. `ListTokensResponse`, `ListCertificatesResponse`, `SignResponse`).

3.4.3.1 ChipGateway Activation via localhost link

The Client sends an activation link to the ChipGateway via the specified localhost-interface. The ChipGateway in turn uses the refresh address provided within the activation link to return the connection back to the Client.

For the activation via localhost link, the ChipGateway MUST provide suitable localhost interface at the following address:

```
http://127.0.0.1:24727/activate/ChipGateway
```

The query-parameters for the localhost interface are specified as follows:

`SessionIdentifier` [Required]
: The identifier of the session between the ChipGateway and the Server. As [**BSI-TR-02102**] requires 100 bits of entropy, the length of the challenge and the session identifier is defined to be 16 bytes.

Copyright © OASIS Open 2018. All Rights Reserved.

TBD: Shift to specific security section?

`ServerAddress` [Required]
The address of the Server.

`RefreshAddress` [Required]
After performing the basic validation of the received request parameters and the check whether there is already an established session, the ChipGateway redirects the browser to this address and indicates the status of the request within an additional URL-parameter status, which is equal to:

`ok` - if the requested session can be established and processed as expected, or
`busy` - if there is already an established session.

`Binding` [Required]
Identifies the binding of the message to the underlying transport protocol. The JSON-based binding specified in this document is addressed by

```
urn:oasis:names:tc:dss-x:profiles:localsig:binding:chipgateway:JSON
```

`PathSecurity-Protocol` [Required]
This element specifies the security protocol, which is to be used for securing the connection between the ChipGateway and the ChipGateway-Server. The present document defines the following values:

`urn:ietf:rfc:5246`
TLSv1.2 according to [**RFC 5246**] with additional encryption of PINs transported from the Client to the Server via the ChipGateway.

`PathSecurity-Parameters` [Optional]
Depending on the `PathSecurity-Protocol`-Parameter there may be additional parameters here. In case the `PathSecurity-Protocol` above is equal to

```
urn:oasis:names:tc:dss-x:profiles:localsig:pathsecurity:tlsv12-with-pin-encryption
```

the AES key which is to be used for PIN encryption key is provided here in form of a JSON Web Key according to [**RFC 7517**].

`ForceProcessing` [Optional]
If there is already an existing session with the Server and the `ForceProcessing` parameter is present and equal to `false`, then the ChipGateway will immediately return with a redirect to the `RefreshAddress` specified above with `status=busy`. If the parameter `ForceProcessing` is missing or equal to `true`, the Chipgateway will proactively terminate a possibly existing session, establish the new one and return with `status=ok`.

Calling the localhost activation link yields one of the following HTTP status codes:

Table 1. ChipgGateway localhost interface return status codes

HTTP Status Code	Description
`303 See Other`	The basic validation of the provided request parameters was performed successfully and hence the ChipGateway performs a redirect to the `RefreshAddress`, which is contained in the “Location” header field of the response and which includes the result of the call within the URL-parameter status as specified above.
`400 Bad Request`	Malformed `GET` request, e.g. because required parameters are missing.
`404 Not Found`	The requested resource is not found within the ChipGateway.
`500 Internal Server Error`	Other errors.

Copyright © OASIS Open 2018. All Rights Reserved.

3.4.3.2 Element HelloRequest

The `HelloRequest` initiates the protocol flow between the ChipGateway and the Server.

`Challenge` [Required]
Is a randomly generated string with sufficient entropy.

`Version` [Required]
Is the version of the ChipGateway consisting of three parts separated by "." (major.minor.subminor).
TBD: Add UserAgent-element to make Version unique?

`SessionIdentifier` [Required]
Identifies the session between the ChipGateway, the Client and the Server.

XML schema snippet defining `HelloRequest`:

```
<element name="HelloRequest" type="cg:HelloRequestType"/>
<complexType name="HelloRequestType">
    <sequence>
        <element name="Challenge" type="hexBinary"/>
        <element name="Version" type="string"/>
        <element name="SessionIdentifier" type="string"/>
    </sequence>
</complexType>
```

JSON Schema snippet defining `HelloRequest`:

```
{
    localName: 'HelloRequestType',
    propertyInfos: [{
        name: 'challenge',
        required: true,
        elementName: 'Challenge',
        typeInfo: 'HexBinary'
    },
    {
        name: 'version',
        required: true,
        elementName: 'Version'
    },
    {
        name: 'sessionIdentifier',
        required: true,
        elementName: 'SessionIdentifier'
    }]
}
```

JSON format example message for a `HelloRequest`

```
{
    "Challenge" : "000102030405060708090a0b0c0c0e0f",
    "Version" : "1.0.0",
    "SessionIdentifier" : "000102030405060708090a0b0c0c0e0f"
}
```

3.4.3.3 HelloResponse

The `HelloResponse` is returned after a successful `HelloRequest` and contains the following attributes and elements inherited from the `ResponseBaseType`:

Copyright © OASIS Open 2018. All Rights Reserved.

`Result` [Required]
Contains the status information and the errors of an executed action. The following error codes are defined:

urn:oasis:names:tc:dss-x:profiles:localsig:result:ok
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:unknownSessionIdentifier
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:unsuitableSessionIdentifier
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:unsuitableChallenge
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:unknownVersion
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:incorrectParameter
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:other
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:updateRequired
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:updateRecommended

`Signature` [Required]
Is a CMS-based signature according to [**RFC 3852**], which has been generated by the the Server on the received `Challenge`. TBD: Should we specify the recommended or admissible algorithms?

`MinimumVersion` [Optional]
Is the minimum required version of the ChipGateway consisting of three parts separated by "." (major.minor.subminor). The parameter is only present, if the ChipGateway version is not up to date, which is indicated by the `Result` being one of

urn:oasis:names:tc:dss-x:profiles:localsig:updateRequired
urn:oasis:names:tc:dss-x:profiles:localsig:updateRecommended

`DownloadAddress` [Optional]
Is the URL at which an updated version of the ChipGateway will be available. This parameter is only present, if the ChipGateway version is not up to date.

The ChipGateway may check, whether the provided update domain is admissible and open a dialogue box with the corresponding link to the update-site immediately, in case an update is required, or at the end of the transaction, if an update is only recommended.

`WebOrigin` [Optional]
Is a sequence of parameters, which defines the set of admissible web origins for starting the ChipGateway protocol. If the ChipGateway was activated from a web origin, which is not part of the list of admissible web origins specified, the ChipGateway shall terminate the communication and return to the presumably hostile web application.

XML schema snippet defining `HelloResponse`:

```
<element name="HelloResponse" type="cg:HelloResponseType"/>
<complexType name="HelloResponseType">
    <complexContent>
        <extension base="cg:ResponseType">
            <sequence maxOccurs="1" minOccurs="0">
                <element name="Signature" type="base64Binary"/>
                <element name="MinimumVersion" type="string"
                    maxOccurs="1" minOccurs="0"/>
                <element name="DownloadAddress" type="anyURI" maxOccurs="1"
                    minOccurs="0"/>
                <element name="WebOrigin" type="string"
                    maxOccurs="unbounded" minOccurs="0"></element>
            </sequence>
        </extension>
    </complexContent>
</complexType>
```

JSON Schema snippet for `HelloReponse`:

Copyright © OASIS Open 2018. All Rights Reserved.

```
{
    localName: 'HelloResponseType',
    baseTypeInfo: '.ResponseType',
    propertyInfos: [{
        name: 'signature',
        required: true,
        elementName: 'Signature',
        typeInfo: 'Base64Binary'
    },
    {
        name: 'minimumVersion',
        elementName: 'MinimumVersion'
    },
    {
        name: 'downloadAddress',
        elementName: 'DownloadAddress'
    },
    {
        name: 'webOrigin',
        minOccurs: 0,
        collection: true,
        elementName: 'WebOrigin'
    }]
}
```

JSON format example messages for a `HelloResponse`:

The following example corresponds to a successful `HelloRequest`:

```
{
   "Result" : "urn:oasis:names:tc:dss-x:profiles:localsig:result:error:ok",
   "Signature" : "AAECAw==",
   "WebOrigin" : "www.example.org"
}
```

The following example signals that there is a newer ChipGateway version than the one in use, while the ChipGateway version is not yet lower than the given minimum version.

```
{
   "Result" : "urn:oasis:names:tc:dss-x:profiles:localsig:result:warning:
updateRecommended",
   "SessionIdentifier" : "000102030405060708090a0b0c0c0e0f",
   "Signature" : "AAECAw==",
   "MinimumVersion" : "1.1.0",
   "DownloadAddress" : "http://example.org/download",
   "WebOrigin" : "www.example.org"
}
```

The following example signals that there is newer ChipGateway version than the one in use, while the ChipGateway version is lower than the given minimum version.

```
{
   "Result" : "urn:oasis:names:tc:dss-x:profiles:localsig:result:error:
updateRequired",
   "SessionIdentifier" : "000102030405060708090a0b0c0c0e0f",
   "Signature" : "AAECAw==",
   "MinimumVersion" : "1.10.0",
   "DownloadAddress" : "http://example.org/download",
   "WebOrigin" : "www.example.org"
```


Copyright © OASIS Open 2018. All Rights Reserved.

```
}
```

3.4.3.4 GetCommand

Precondition: The Server has been authenticated by the ChipGateway by successfully verifying the `Signature` contained in `HelloResponse`.
Postcondition: none
Note: none

`SessionIdentifier` [Required]
Identifies the session between the ChipGateway and the Server.

`TokenInfo` [Optional]
This parameter is present for each Token, which is currently connected to the ChipGateway. See Section 3.4.4.2, “Element TokenInfo”.

XML Schema snippet for `GetCommand`:

```
<element name="GetCommand" type="cg:GetCommandType"/>
<complexType name="GetCommandType">
    <sequence>
        <element name="SessionIdentifier" type="string"/>
        <element ref="cg:TokenInfo" maxOccurs="unbounded" minOccurs="0">
        </element>
    </sequence>
</complexType>
```

JSON Schema snippet for `GetCommand`:

```
{
    localName: 'GetCommandType',
    propertyInfos: [{
        name: 'sessionIdentifier',
        required: true,
        elementName: 'SessionIdentifier'
    },
    {
        name: 'tokenInfo',
        minOccurs: 0,
        collection: true,
        elementName: 'TokenInfo',
        typeInfo: '.TokenInfoType'
    }]
}
```

JSON format example message for a `GetCommand`:

```
{
   "SessionIdentifier" : "000102030405060708090a0b0c0c0e0f",
   "TokenInfo" : [ {
      "ConnectionHandle" : {
         "CardType" : "http://example.org/cif/v3",
         "SlotHandle" : "7365637265742D73657373696F6E2D6964"
      },
      "HasProtectedAuthPath" : false,
      "NeedsPinForCertAccess" : false,
      "NeedsPinForPrivateKeyAccess" : true,
      "Algorithm" : "SHA256withRSA "
   } ]
}
```


Copyright © OASIS Open 2018. All Rights Reserved.

3.4.4 ChipGateway Commands

After a logical connection from the Server to the ChipGateway has been established, the Server is able to send the following commands to the ChipGateway:

- `ListTokensRequest`
- `ListCertificatesRequest`
- `SignRequest`
- `Terminate`

Furthermore the calls `ListTokensRequest` and `ListCertificatesRequest` may be send from the Client to the Server.

3.4.4.1 Type ResponseType

This clause specifies the basic `ResponseType` type, which is used in every ChipGateway-specific response.

`Result` [Required]
Indicates the result of a ChipGateway command in form of a specific URI.

XML Schema snippet for `ResponseType`:

```
<complexType name="ResponseType">
  <sequence>
   <element name="Result" type="anyURI"/>
  </sequence>
 </complexType>
```

3.4.4.2 Element TokenInfo

The `TokenInfo` element is used in the definition of `GetCommand`, `ListTokensRequest` and `ListTokensResponse` and is present for each Token currently connected to the ChipGateway. It contains the following elements:

`ConnectionHandle` [Optional]
Is a handle to a connected Token. Details of the parameter are specified in Section 3.4.4.3, "Element ConnectionHandle"

`HasProtectedAuthPath` [Optional]
Is `True`, if the card terminal which has captured the Token is equipped with a pin pad. If this parameter is `True`, the PIN is captured by the card reader.

`NeedsPinForCertAccess` [Optional]
Is `True`, if reading the certificates stored on the Token requires entry of a PIN.

`NeedsPinForPrivateKeyAccess` [Optional]
Is `True`, if accessing the private key requires entry of a PIN.

`Algorithm` [Optional]
Is present for each asymmetric cryptographic algorithm supported by the Token. This parameter is specified according to the Java Cryptographic Architecture (JCA) name of a supported algorithm (cf. Appendix D, *Chipgateway Algorithms*).

XML Schema snippet for `TokenInfo`:

```
<element name="TokenInfo" type="cg:TokenInfoType">
<complexType name="TokenInfoType">
<sequence>
```


Copyright © OASIS Open 2018. All Rights Reserved.

```
 <element ref="cg:ConnectionHandle" maxOccurs="1"; minOccurs="0"/>
 <element name="HasProtectedAuthPath" type="boolean" maxOccurs="1"
  minOccurs="0"/>
 <element name="NeedsPinForCertAccess" type="boolean" maxOccurs="1>
  minOccurs="0"/>
 <element name="NeedsPinForPrivateKeyAccess" type="boolean">
  maxOccurs="1" minOccurs="0"/>
 <element name="Algorithm" type="string" maxOccurs="unbounded">
  minOccurs="0"/>
 </element>
</sequence>
</complexType>
```

JSON Schema snippet for `TokenInfo`:

```
{
localName: 'TokenInfoType',
propertyInfos: [{
 name: 'connectionHandle',
 elementName: 'ConnectionHandle',
 typeInfo: '.ConnectionHandleType'
},
{
 name: 'hasProtectedAuthPath',
 elementName: 'HasProtectedAuthPath',
 typeInfo: 'Boolean'
},
{
 name: 'needsPinForCertAccess',
 elementName: 'NeedsPinForCertAccess',
 typeInfo: 'Boolean'
},
{
 name: 'needsPinForPrivateKeyAccess',
 elementName: 'NeedsPinForPrivateKeyAccess',
 typeInfo: 'Boolean'
},
{
 name: 'algorithm',
 minOccurs: 0,
 collection: true,
 elementName: 'Algorithm'
}]
}
```

3.4.4.3 Element ConnectionHandle

The `ConnectionHandle` element is a handle to a connected Token. It contains the following elements:

`CardType` [Required]
: The globally unique identifier for the type of the Token (cf. [**ISO 24727**], Part 3).

`SlotHandle` [Optional]
: The (with respect to the IFD-Layer of the ChipGateway) unique handle, which addresses a connected Token.

XML Schema snippet for `ConnectionHandle`:

```
element name="ConnectionHandle" type="cg:ConnectionHandleType"/>
```


Copyright © OASIS Open 2018. All Rights Reserved.

```
<complexType name="ConnectionHandleType">
<sequence>
 <element name="CardType" type="anyURI"/>
 <element name="SlotHandle" type="hexBinary" maxOccurs="1" minOccurs="0">
 </sequence>
</complexType>
```

JSON Schema snippet for `ConnectionHandle`:

```
{
localName: 'ConnectionHandleType',
propertyInfos: [{
 name: 'cardType',
 required: true,
 elementName: 'CardType'
}, {
 name: 'slotHandle',
 elementName: 'SlotHandle',
 typeInfo: 'HexBinary'
}]
}
```

3.4.4.4 Elements ListTokensRequest and ListTokensResponse

The `ListTokensRequest` command allows to wait for a specific Token identified by appropriate filter mechanisms.

This function requires that the Server has been authenticated by the ChipGateway by successfully verifying the `Signature` contained in `HelloResponse`.

3.4.4.4.1 Element ListTokensRequest

The `ListTokensRequest` element consists of the following elements:

`SessionIdentifier` [Optional]
: The `SessionIdentifier` parameter is present, if the `ListTokensRequest` is sent from the Client to the Server in order to identify the session. This parameter is omitted in `ListTokensRequest` messages transmitted between the Server and the ChipGateway.

`MaxWaitSeconds` [Required]
: Specifies the number of seconds until a timeout occurs.

`TokenInfo` [Required]
: May be present multiple times in order to specify the set of admissible Tokens.

  The sequence of `TokenInfo` elements is to be interpreted as combined with a logical OR and the set of features withing a given `TokenInfo` element (cf. Section 3.4.4.2, “Element TokenInfo”) is to interpreted as being combined with a logical AND.

XML Schema snippet for `ListTokensRequest`:

```
<element name="ListTokensRequest" type="cg:ListTokensRequestType"/>
    <complexType name="ListTokensRequestType">
        <sequence>
   <element name="SessionIdentifier" type="string" minOccurs="0"/>
   <element name="MaxWaitSeconds" type="positiveInteger"/>
            <element name="TokenInfo" type="cg:TokenInfoType"
                maxOccurs="unbounded" minOccurs="1"/>
        </sequence>
    </complexType>
```


Copyright © OASIS Open 2018. All Rights Reserved.

JSON Schema snippet for `ListTokensRequest`:

```
{
        localName: 'ListTokensRequestType',
        propertyInfos: [
  {
   name: 'sessionIdentifier',
            required: false,
            elementName: 'SessionIdentifier',
            typeInfo: 'string'
        },
  {
   name: 'maxWaitSeconds',
            required: true,
            elementName: 'MaxWaitSeconds',
            typeInfo: 'PositiveInteger'
        },
        {
            name: 'tokenInfo',
            required: true,
            collection: true,
            elementName: 'TokenInfo',
            typeInfo: '.TokenInfoType'
        }]
    }
```

JSON format example message for a `ListTokensRequest`:

```
"ListTokensRequest" : {
 "SessionIdentifier" : "000102030405060708090a0b0c0c0e0f",
 "MaxWaitSeconds" : 60,
    "TokenInfo" : [{
        "ConnectionHandle" : {
        "CardType" : "http://example.org/cif/v3"
    },
        "HasProtectedAuthPath" : false,
        "NeedsPinForCertAccess" : true,
        "NeedsPinForPrivateKeyAccess" : false,
        "Algorithm" : "SHA256withRSA"
    }]
}
```

3.4.4.4.2 Element ListTokensResponse

The element `ListTokensReponse` inherits `ResponseType` and contains the following attributes and elements:

`Result` [Required]
: Contains the status information for the executed action (i.e. `ListTokensRequest`, whereas there are the following error codes defined:

```
urn:oasis:names:tc:dss-x:profiles:localsig:result:ok
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:timeout
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:incorrectParameter
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:stopped
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:other
```

`SessionIdentifier` [Optional]
: Identifies the session between the ChipGateway and the Server. The `SessionIdentifier` parameter is present, if the `ListTokensResponse` is sent from the Server to the ChipGateway.

Copyright © OASIS Open 2018. All Rights Reserved.

This parameter is omitted in `ListTokensResponse` messages transmitted between the Server and the Client.

`TokenInfo`
May be present multiple times and contains the token-related information of the admissible tokens. The details of the `TokenInfo` structure are specified in Section 3.4.4.2, "Element TokenInfo".

XML Schema snippet for `ListTokensResponse`:

```
<element name="ListTokensResponse" type="cg:ListTokensResponseType"/>
<complexType name="ListTokensResponseType">
    <complexContent>
        <extension base="cg:ResponseType">
            <sequence>
                <element name="SessionIdentifier" type="string" minOccurs="0"/>
                <element name="TokenInfo" type="cg:TokenInfoType"
                    maxOccurs="unbounded" minOccurs="0"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
```

JSON Schema snippet for `ListTokensResponse`:

```
{
    localName: 'ListTokensResponseType',
    baseTypeInfo: '.ResponseType',
    propertyInfos: [{
        name: 'sessionIdentifier',
        required: false,
        elementName: 'SessionIdentifier'
    },
    {
        name: 'tokenInfo',
        minOccurs: 0,
        collection: true,
        elementName: 'TokenInfo',
        typeInfo: '.TokenInfoType'
    }]
}
```

JSON format example message for a `ListTokensResponse`:

```
"Result" : "urn:oasis:names:tc:dss-x:profiles:localsig:result:ok",
"SessionIdentifier" : "000102030405060708090a0b0c0c0e0f",
"TokenInfo" : [{
    "ConnectionHandle" : {
        "CardType" : "http://example.org/cif/v3",
        "SlotHandle" : "7365637265742D73657373696F6E2D6964"
    },
    "HasProtectedAuthPath" : false,
    "NeedsPinForCertAccess" : true,
    "NeedsPinForPrivateKeyAccess" : false,
    "Algorithm" : "SHA256withRSA"
}]
```

3.4.4.5 Elements ListCertificatesRequest and ListCertificatesResponse

The `ListCertificatesRequest` command allows to retrieve the available X.509 certificates from a connected local signature creation device.

Copyright © OASIS Open 2018. All Rights Reserved.

This function requires that the Server has been authenticated by the ChipGateway by successfully verifying the `Signature` contained in `HelloResponse`.

3.4.4.5.1 Element ListCertificatesRequest

The `ListCertificatesRequest` consists of the following elements:

`SessionIdentifier` [Optional]
: The `SessionIdentifier` parameter is present, if the `ListCertificatesRequest` is sent from the Client to the Server in order to identify the session. This parameter is omitted in `ListCertificateRequest` messages transmitted between the Server and the ChipGateway.

`MaxWaitSeconds` [Required]
: Specifies the number of seconds until a timeout occurs.

`SlotHandle` [Required]
: Addresses the specific token from which the certifcates are to be read.

`PIN` [Optional]
: Is an optional parameter, which allows to transport the `PIN`, if captured within the Client, which may be realised as web application, instead of the ChipGateway. This `PIN` is encrypted with the symmetric key which has been provided whithin the `PathSecurityParameter` of the activation call (see Section 3.4.3.1, "ChipGateway Activation via localhost link"). The format of the encrypted `PIN` is as defined in [**RFC 7516**].

  Whether entering the `PIN` in the Client and transportation via the Server is allowed or not is subject to the applicable policy. If a card reader is equipped with a PIN-pad, it is recommended to capture the `PIN` there.

`CertifcateFilter` [Optional]
: May be present multiple times in order to specificy the requested certificate.

  The sequence of `CertifcateFilter` elements is to be interpreted as combined with a logical `OR` and the set of features within a given `CertifcateFilter` element (see Section 3.4.4.5.1.1, "Element CertificateFilter") is to be interpreted as being combined with a logical `AND`.

XML Schema snippet for `ListCertificatesRequest`:

```
<element name="ListCertificatesRequest"
    type="cg:ListCertificatesRequestType"/>
<complexType name="ListCertificatesRequestType">
    <sequence>
  <element name="SessionIdentifier" type="string" minOccurs="0"/>
  <element name="MaxWaitSeconds" type="positiveInteger"/>
        <element name="SlotHandle" type="hexBinary"/>
        <element name="PIN" type="string" maxOccurs="1" minOccurs="0"/>
        <element name="CertificateFilter" type="cg:CertificateFilterType"
            maxOccurs="unbounded" minOccurs="0"/>
    </sequence>
</complexType>
```

JSON Schema snippet for `ListCertificatesRequest`:

```
{
    localName: 'ListCertificatesRequestType',
    propertyInfos: [
 {
        name: 'sessionIdentifier',
        required: false,
        elementName: 'SessionIdentifier',
```


Copyright © OASIS Open 2018. All Rights Reserved.

```
        typeInfo: 'string'
    },
 {
        name: 'maxWaitSeconds',
        required: true,
        elementName: 'MaxWaitSeconds',
        typeInfo: 'PositiveInteger'
    },
    {
        name: 'slotHandle',
        required: true,
        elementName: 'SlotHandle',
        typeInfo: 'HexBinary'
    },
    {
        name: 'pin',
        elementName: 'PIN'
    },
    {
        name: 'certificateFilter',
        minOccurs: 0,
        collection: true,
        elementName: 'CertificateFilter',
        typeInfo: '.CertificateFilterType'
    }]
    }
```

JSON format example message for a `ListCertificatesRequest`:

```
{
    "ListCertificatesRequest" : {
 "SessionIdentifier" : "000102030405060708090a0b0c0c0e0f",
    "MaxWaitSeconds" : 60,
    "SlotHandle" : "736C6F74312D68616E646C65",
    "PIN" : "eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkEyNTZHQ00ifQ…",
    "CertificateFilter" : [{
        "Policy" : "1.3.171.1.1.2.4",
        "Issuer" : "CN=Example",
        "KeyUsage" : "authentication"
    }]
    }
    }
```

where "`PIN`" equals:

```
BASE64URL(UTF8({"alg": "dir", "enc": "A256GCM"}))
BASE64URL(JWE Initialization Vector)
BASE64URL(JWE Ciphertext)
BASE64URL(JWE Authentication Tag)
```

3.4.4.5.1.1 Element CertificateFilter

The `Certificate` consists of the following elements:

`Policy` [Optional]
: Allows to filter the set of returned certificates according to the `Policy` of the certifcate.

`Issuer` [Optional]
: Is a regular expression, which allows to filter the set of returned certificates according to the issuer of the certificate.

Copyright © OASIS Open 2018. All Rights Reserved.

This string allows to search on the RDN-components

Common Name (CN) and
Given Name (GN)
of the Issuer with a prefix search ("PartialName*"), infix search ("*PartialName*") or exact match ("PartialName").

`KeyUsage` [Optional]
Allows to filter the set of returned certificates according to the key usage extension of the certificate.

The mapping from the `KeyUsage` element defined here to the `KeyUsage BIT STRING` according to [RFC 5280] is as follows:

Table 2. KeyUsage Mapping

KeyUsage	[RFC 5280]
AUTHENTICATION	digitalSignature (0)
SIGNATURE	nonRepudiation (1)
ENCRYPTION	keyEncipherment (2), dataEncipherment (3), keyAgreement (4)

3.4.4.5.2 Element ListCertificatesResponse

The element `ListCertificatesResponse` contains the following elements:

`Result` [Required]
Contains the status information and the erros of an executed action, whereas there are the following error codes defined:

urn:oasis:names:tc:dss-x:profiles:localsig:result:ok
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:timeout
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:unknownSlotHandle
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:incorrectParameter
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:securityConditionNotSatisfied
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:stopped
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:other

`SessionIdentifier` [Optional]
Identifies the session between the ChipGateway and the Server. The `SessionIdentifier` parameter is present, if the `ListCertificatesResponse` is sent from the Server to the ChipGateway. This parameter is omitted in `ListCertificateResponse` messages transmitted between the Server and the Client.

`RetryCounter` [Optional]
If the provided `PIN` was wrong, the number of remaining attempts is returned in this parameter.

`CertificateInfo` [Optional]
Contains information with respect to the available certifcates. The details of the `CertificateInfo` element are described in Section 3.4.4.5.2.1, “Element CertificateInfo”.

XML Schema snippet of `ListCertificatesResponse`:

```
<element name="ListCertificatesResponse"
    type="cg:ListCertificatesResponseType"/>
<complexType name="ListCertificatesResponseType">
    <complexContent>
        <extension base="cg:ResponseType">
            <sequence maxOccurs="1" minOccurs="1">
                <element name="SessionIdentifier" type="string" minOccurs="0"/>
```


Copyright © OASIS Open 2018. All Rights Reserved.

```
                <element name="RetryCounter" type="nonNegativeInteger"
                    maxOccurs="1" minOccurs="0"/>
                <element ref="cg:CertificateInfo" maxOccurs="unbounded"
                    minOccurs="0"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
```

JSON Schema snippet for `ListCertificatesResponse`:

```
{
    localName: 'ListCertificatesResponseType',
    baseTypeInfo: '.ResponseType',
    propertyInfos: [{
        name: 'sessionIdentifier',
        required: false,
        elementName: 'SessionIdentifier'
    },
    {
        name: 'retryCounter',
        elementName: 'RetryCounter',
        typeInfo: 'NonNegativeInteger'
    },
    {
        name: 'certificateInfo',
        minOccurs: 0,
        collection: true,
        elementName: 'CertificateInfo',
        typeInfo: '.CertificateInfoType'
    }]
}
```

JSON format example message for a `ListCertificatesResponse`:

```
{
   "Result" : "urn:oasis:names:tc:dss-x:profiles:localsig:result:ok",
   "SessionIdentifier" : "000102030405060708090a0b0c0c0e0f",
   "CertificateInfo" : [{
      "DIDName" : "SignKey",
      "Algorithm" : "SHA256withRSA",
      "Certificate" : [ "dGhpcyBzaG91bGQgYmUgYSBjZXJ0aWZpY2F0ZQ==" ],
      "UniqueSSN" : "12345678901234567890"
   }]
}
```

where "`PIN`" equals:

```
BASE64URL(UTF8({"alg": "dir", "enc": "A256GCM"}))
BASE64URL(JWE Initialization Vector)
BASE64URL(JWE Ciphertext)
BASE64URL(JWE Authentication Tag)
```

3.4.4.5.2.1 Element CertificateInfo

The Element `CertificateInfo`:

`DIDName` [Required]
: The name of the Differential Identity (cf. [ISO 24727], Part 3) corresponding to the private key of the certificate.

Copyright © OASIS Open 2018. All Rights Reserved.

`Algorithm` [Required]
Contains the name of the cryptographic algorithm related to certificate. Admissible values are specified in section Appendix D, *Chipgateway Algorithms*.

`Certificate` [Required]
Is the end user certificate under consideration and optionally further certificates in the certificate path which have been read from the Token.

`UniqueSSN` [Required]
Is the unique identifier of the certificate holder, which is contained in or derived from the certificate.

XML Schema snippet of `CertificateInfo`:

```
<element name="CertificateInfo" type="cg:CertificateInfoType"/>
<complexType name="CertificateInfoType">
    <sequence>
        <element name="DIDName" type="cg:NameType"/>
        <element name="Algorithm" type="string"/>
        <element name="Certificate" type="base64Binary"
            maxOccurs="unbounded" minOccurs="1"/>
        <element name="UniqueSSN" type="string"/>
    </sequence>
</complexType>
```

JSON Schema snippet of `CertificateInfo`:

```
{
localName: 'CertificateInfoType',
    propertyInfos: [{
        name: 'didName',
        required: true,
        elementName: 'DIDName',
        typeInfo: 'NormalizedString'
    },
    {
        name: 'algorithm',
        required: true,
        elementName: 'Algorithm'
    },
    {
        name: 'certificate',
        required: true,
        collection: true,
        elementName: 'Certificate',
        typeInfo: 'Base64Binary'
    },
    {
        name: 'uniqueSSN',
        required: true,
        elementName: 'UniqueSSN'
    }]
}
```

3.4.4.6 Elements `SignRequest` and `SignResponse`

The `SignRequest` function allows to sign a message with a specific Token connected to the ChipGateway.

This function requires that the Server has been authenticated by the ChipGateway by successfully verifying the `Signature` contained in `HelloResponse`.

Copyright © OASIS Open 2018. All Rights Reserved.

3.4.4.6.1 Element SignRequest

The Element `SignRequest` consists of the following elements:

`MaxWaitSeconds` [Required]
: Specifies the number of seconds until a timeout occurs.

`SlotHandle` [Required]
: Addresses the specific Token which is to be used for signing.

`DIDName` [Required]
: The name of the Differential Identity (cf. [**ISO 24727**], Part 3) corresponding to the private key of the certificate.

`PIN` [Optional]
: Is an optional parameter, which allows to transport the `PIN`, if captured within the Client, which may be realised as web application, instead of the ChipGateway. This `PIN` is encrypted with the symmetric key which has been provided whithin the `PathSecurityParameter` of the activation call (see Section 3.4.3.1, "ChipGateway Activation via localhost link"). The format of the encrypted `PIN` is as defined in [**RFC 7516**].

  Whether entering the `PIN` in the Client and transportation via the Server is allowed or not is subject to the applicable policy. If a card reader is equipped with a PIN-pad, it is recommended to capture the `PIN` there.

`Message` [Required]
: The message (or hash value), which is to be signed. 

XML Schema snippet of `SignRequest`:

```
<element name="SignRequest" type="cg:SignRequestType"/>
<complexType name="SignRequestType">
    <sequence>
        <element name="MaxWaitSeconds" type="positiveInteger"/>
        <element name="SlotHandle" type="hexBinary"/>
        <element name="DIDName" type="cg:NameType"/>
        <element name="PIN" type="string" maxOccurs="1" minOccurs="0"/>
        <element name="Message" type="hexBinary"/>
    </sequence>
</complexType>
```

JSON Schema snippet of `SignRequest`:

```
{
    localName: 'SignRequestType',
    propertyInfos: [{
        name: 'maxWaitSeconds',
        required: true,
        elementName: 'MaxWaitSeconds',
        typeInfo: 'PositiveInteger'
    },
    {
        name: 'slotHandle',
        required: true,
        elementName: 'SlotHandle',
        typeInfo: 'HexBinary'
    },
    {
        name: 'didName',
        required: true,
```


Copyright © OASIS Open 2018. All Rights Reserved.

```
        elementName: 'DIDName',
        typeInfo: 'NormalizedString'
    },
    {
        name: 'pin',
        elementName: 'PIN'
    },
    {
        name: 'message',
        required: true,
        elementName: 'Message',
        typeInfo: 'HexBinary'
    }]
}
```

JSON format example message for a `SignRequest`:

```
{
   "SignRequest" : {
      "MaxWaitSeconds" : 60,
      "SlotHandle" : "736C6F74312D68616E646C65",
      "DIDName" : "AuthKey",
      "PIN" : "eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkEyNTZHQ00ifQ…",
      "Message" :
    "61207772697474656E20636F6E7472616374206F76657220323030E282AC"
   }
}
```

where "`PIN`" equals:

```
BASE64URL(UTF8({"alg": "dir", "enc": "A256GCM"}))
BASE64URL(JWE Initialization Vector)
BASE64URL(JWE Ciphertext)
BASE64URL(JWE Authentication Tag)
```

3.4.4.6.2 Element `SignResponse`

The element `SignResponse` contains the following elements:

`Result` [Required]
: Contains the status information and the erros of an executed action, whereas there are the following error codes defined:

```
urn:oasis:names:tc:dss-x:profiles:localsig:result:ok
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:timeout
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:unknownSlotHandle
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:incorrectParameter
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:securityConditionNotSatisfied
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:stopped
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:other
```

`SessionIdentifier` [Required]
: Identifies the session between the ChipGateway and the Server.

`RetryCounter` [Optional]
: If the provided `PIN` was wrong, the number of remaining attempts is returned in this parameter.

`Signature` [Optional]
: Contains the generated signature.

XML Schema snippet for `SignResponse`:

Copyright © OASIS Open 2018. All Rights Reserved.

```
<element name="SignResponse" type="cg:SignResponseType"/>
<complexType name="SignResponseType">
    <complexContent>
        <extension base="cg:ResponseType">
            <sequence>
                <element name="SessionIdentifier" type="string"/>
                <element name="RetryCounter" type="nonNegativeInteger"
                    maxOccurs="1" minOccurs="0"/>
                <element name="Signature" type="base64Binary" maxOccurs="1"
                    minOccurs="0"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
```

JSON Schema snippet for `SignResponse`:

```
{
    localName: 'SignResponseType',
    baseTypeInfo: '.ResponseType',
    propertyInfos: [{
        name: 'sessionIdentifier',
        required: true,
        elementName: 'SessionIdentifier'
    },
    {
        name: 'retryCounter',
        elementName: 'RetryCounter',
        typeInfo: 'NonNegativeInteger'
    },
    {
        name: 'signature',
        elementName: 'Signature',
        typeInfo: 'Base64Binary'
    }]
}
```

JSON format example message for a `SignResponse`:

```
{
    "Result" : "urn:oasis:names:tc:dss-x:profiles:localsig:result:ok",
    "SessionIdentifier" : "000102030405060708090a0b0c0c0e0f ",
    "Signature" : "c2lnbmVkUmVzcG9uc2U="
}
```

3.4.4.7 Element `Terminate`

The `Terminate` element terminates the session.

3.4.4.7.1 Element Terminate

The element `Terminate` has the following child elements:

`Result` [Required]
Contains the status information and the errors of an executed action, whereas there are the following error codes defined:

```
urn:oasis:names:tc:dss-x:profiles:localsig:result:ok
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:timeout
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:unknownSessionIdentifier
```


Copyright © OASIS Open 2018. All Rights Reserved.

urn:oasis:names:tc:dss-x:profiles:localsig:result:error:unsuitableSessionIdentifier
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:incorrectParameter
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:cancellationByUser
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:updateRequired
urn:oasis:names:tc:dss-x:profiles:localsig:result:warning:updateRecommended
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:stopped
urn:oasis:names:tc:dss-x:profiles:localsig:result:error:other

`SessionIdentifier` [Optional]
Identifies the session between the ChipGateway and the Server.

This parameter MUST be present, if the ChipGateway proactively terminates an existing session and sends `Terminate` to the Server. In this case the `Result` MUST be equal to

urn:oasis:names:tc:dss-x:profiles:localsig:result:error:stopped

XML Schema snippet for `Terminate`:

```
<element name="Terminate" type="cg:TerminateType"/>
<complexType name="TerminateType">
    <complexContent>
        <extension base="cg:ResponseType">
            <sequence>
                <element name="SessionIdentifier" type="string"
                    maxOccurs="1" minOccurs="0"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
```

JSON Schema snippet for `Terminate`:

```
{
    localName: 'TerminateType',
    baseTypeInfo: '.ResponseType',
    propertyInfos: [{
        name: 'sessionIdentifier',
        elementName: 'SessionIdentifier'
    }]
}
```

JSON format example messages for `Terminate`:

The following example corresponds to the proactive termination of an existing session by the ChipGateway, where the `Terminate` message is sent from the ChipGateway to the Server:

```
{
    "Terminate" : {
    "Result" : "urn:oasis:names:tc:dss-x:profiles:localsig:result:error:stopped",
     "SessionIdentifier" : "000102030405060708090a0b0c0c0e0f"
    }
}
```

The following examples correspond to `Terminate` messages sent from the Server to the ChipGateway.

```
{
    "Terminate" : {
    "Result" : "urn:oasis:names:tc:dss-x:profiles:localsig:result:ok"
    }
}
```

 Copyright © OASIS Open 2018. All Rights Reserved.

```
{
    "Terminate" : {
    "Result" : "urn:oasis:names:tc:dss-x:profiles:localsig:result:error:other"
    }
}
```


Copyright © OASIS Open 2018. All Rights Reserved.

4 Protocol Bindings

OASIS DSS bindings are categorized under either transport bindings or security bindings as defined under Section 6 of [**DSSCore**]. The DSS signing protocol messages inherently assume that all security aspects are covered by the transport binding and appropriate security binding.

In Section Section 3.1, “User Agent” a transport binding is assumed by which a session is established between the Digital Signature Service and the HTTP User Agent. This document profiles a binding for a HTTP User Agent at the client using a web form.

In Section Section 3.2, “Two-Step Approach” it is assumed that a session (transaction) is created when a request contains the element `<localsig:RequestDocumentHash>` with the attribute `MaintainRequestState="true"`. The session is not profiled in this document.

4.1 WEB FORM Transport Binding

A WEB FORM binding is defined as a mechanism by which OASIS DSS protocol messages may be transmitted within the base64-encoded content of a HTML form control.

The reference URI for this binding is:

```
urn:oasis:names:tc:dss-x:profiles:localsig:bindings:web-form
```

4.1.1 Overview

The WEB FORM binding is intended for those cases where a requester and responder need to communicate using a HTTP User Agent (as defined in HTTP 1.1 [**RFC 2616**]) as an intermediary. This may be necessary, for example, if the communicating parties do not share a direct path of communication. It may also be needed if the responder requires an interaction with the User Agent in order to fulfill the request, such as when the User Agent must authenticate itself.

The OASIS DSS `<dss:SignRequest>` and `<dss:SignResponse>` XML messages are encoded into a HTML form, as described in the next Section.

4.1.2 Message Encoding using a HTML form

The HTML document MUST adhere to the XHTML 1.0 specification [**XHTML**] to ease parsing.

The `action` attribute of the HTML form MUST be the HTTP endpoint of the recipient: either the OASIS Digital Signature Service in case of the `<dss:SignRequest>` or the client in case of the `<dss:SignResponse>`.

The `method` attribute of the HTML form MUST be `POST`.

A `<dss:SignRequest>` or a `<dss:SignResponse>` message MUST be base64-encoded and placed in a hidden HTML form control within the HTML form (as defined by [**HTML401**] Section 17). The base64-encoded value MAY be line-wrapped at a reasonable length in accordance with common practice.

- If the message contains an OASIS DSS `<dss:SignRequest>` then the hidden HTML form control MUST be named `signrequest`.
- If the message contains an OASIS DSS `<dss:SignResponse>`, then the hidden HTML form control MUST be named `signresponse`.

The `clienturl` attribute MAY be used to provide the Digital Signature Service with a client URL. This URL will be used by the Digital Signature Service for the transmission of the response HTML form. If

Copyright © OASIS Open 2018. All Rights Reserved.

the `clienturl` is not specified, either an error is returned to the HTTP agent or a predefined URL is used by the Digital Signature Service.

Any additional HTML form controls or presentation MAY be included to allow the recipient to process the message.

Any technique supported by the User Agent MAY be used to cause the submission of the HTML form, and any HTML form content necessary to support this MAY be included, such as submit controls and client-side scripting commands. However, the Digital Signature Service MUST be able to process the message regardless for the mechanism by which the form submission is initiated.

Note that any HTML form control values included MUST be transformed so as to be safe to include in the XHTML document. This includes transforming characters such as quotes into HTML entities, etc.

4.1.3 HTTP and Caching Considerations

HTTP proxies and the User Agent intermediary should not cache OASIS DSS protocol messages. To ensure this, the following rules SHOULD be followed. When returning OASIS DSS protocol messages using HTTP 1.1, HTTP responders SHOULD:

- Include a `Cache-Control` header field set to " `no-cache, no-store` ".
- Include a `Pragma` header field set to " `no-cache` ".

There are no other restrictions on the use of HTTP headers.

4.2 Security Binding (Non-Normative)

4.2.1 Security Considerations

Before deployment, each combination of profile, transport binding, and security binding SHOULD be analyzed for vulnerability in the context of the specific protocol exchange and the deployment environment. Below we illustrate some of the security concerns that often come up with protocols of this type, but we stress that this is not an exhaustive list of concerns.

As the OASIS DSS protocol defined in this document is similar to the SAML protocol most of the security considerations defined in [**SAMLCore**] also apply to the OASIS DSS protocol.

The creation of document signatures using the OASIS DSS protocol yields additional attack vectors, due to possible manipulations of the document that is being transferred between DSS client and a DSS server. If the end user signs a different document as assumed by the DSS client, the impact could be huge. Therefore, it is of eminent importance to properly secure the OASIS DSS protocol request message that is transferred from DSS client to the DSS server via an intermediate User Agent.

4.2.2 TLS Security

The Section 4.1, “WEB FORM Transport Binding” SHOULD be used with the TLS Security Bindings as defined under Section 6.3 of [**DSSCore**].

4.2.3 Claimed Identity

The DSS Client can include the optional `<dss:ClaimedIdentity>` element as defined in [**DSSCore**] Section 2.8.2 to indicate the identity of the client who is making the request. The information provided by `<dss:ClaimedIdentity>` can be used to further personalize the interface presented to the end user by the DSS server.

In case the DSS server detects a problem with the claimed identity, the service returns in `<dss:SignResponse>` a `<dss:ResultMajor>` with the value `RequesterError` and a `<dss:ResultMinor>` with the value:

Copyright © OASIS Open 2018. All Rights Reserved.

```
urn:oasis:names:tc:dss-x:profiles:localsig:resultminor:claimed-
```

4.2.4 ChipGateway Transport Binding

[TODO] Introduction to the binding of the ChipGateway.

The reference URI for this binding is:

```
urn:oasis:names:tc:dss-x:profiles:localsig:bindings:chipgateway
```

To Do: Details of the binding of the ChipGateway.

Copyright © OASIS Open 2018. All Rights Reserved.

5 Conformance

There are three conformance profiles, one for each approach.

- Profile A: Stateless Two-Step Approach
- Profile B: Stateful Two-Step Approach
- Profile C: User Agent Approach
- Profile D: Third Party Approach
- Profile E: Chip Gateway Approach

5.1 Conformance Profile A - Stateless Two-Step Approach

This conformance profile only applies to a Two-Step approach; see Section 3.2, “Two-Step Approach”.

5.1.1 Conformance Target: Server

The subject of the conformance test is the server that implements the Digital Signature Service.

5.1.1.1 Level 1

The conformance level states that the Digital Signature Service cannot maintain state information between subsequent requests in the Two-Step approach.

The request MUST contain the element `<dss:ServicePolicy>` with the value

```
http://docs.oasis-open.org/dss-x/ns/localsig/two-step-appro
```

If a request contains the element `<localsig:RequestDocumentHash>` and the attribute value of `MaintainRequestState` is either absent or has a value of `"false"`, then compute the digest value of the given input document. In case of success, the response MUST contain the element `<dss:DocumentHash>` with the digest value.

If a request contains the element `<dss:SignatureObject>` (and optionally, the `<dss:DocumentHash>`), then the server MUST create the signed document by means of the given `<dss:SignatureObject>` and the given input document. In case of success, the response MUST contain the signed document.

The element `<localsig:CorrelationID>` is ignored in a request; the element `<localsig:CorrelationID>` MUST NOT be returned in a response.

5.1.2 Conformance Target: Client

The subject of the conformance test is the client of the Digital Signature Service.

5.1.2.1 Level 1

The conformance level states that the client of the Digital Signature Service is capable of computing a digital signature value based on a given digest value of a document

The requests MUST contain the element `<dss:ServicePolicy>` with the value

Copyright © OASIS Open 2018. All Rights Reserved.

```
http://docs.oasis-open.org/dss-x/ns/localsig/two-step-approa
```

The FIRST request MUST contain the document to be signed as well as the element `<localsig:RequestDocumentHash>` for which the attribute `MaintainRequestState` is either absent or has a value of `"false"`.

The `<dss:DocumentHash>` is obtained from the FIRST response and MUST be used to compute the digital signatue value.

The SECOND request MUST contain the document to be signed, as provided in the FIRST request, as well as the `<dss:SignatureObject>` that contains the digital signatue value.

If the FIRST request contained the `<dss:DocumentHash>` element, the SECOND request MUST contain that element too with the same value.

5.2 Conformance Profile B - Stateful Two-Step Approach

This conformance profile only applies to a Two-Step approach; see Section 3.2, “Two-Step Approach”.

5.2.1 Conformance Target: Server

The subject of the conformance test is the server that implements the Digital Signature Service.

5.2.1.1 Level 1

The conformance level 'Stateful' states that the Digital Signature Service is able to maintain state information between two subsequent requests (in the Two-Step approach).

The element `<dss:ServicePolicy>` MUST be present with the value

```
http://docs.oasis-open.org/dss-x/ns/localsig/two-step-approa
```

The FIRST request MUST contain the element `<localsig:RequestDocumentHash>` and the attribute `MaintainRequestState="true"` in the OptionalInputs. In case of success, the FIRST response MUST contain a reference to the state, by means of the element `<localsig:CorrelationID>` and MUST contain the digest value of the input document by means of the element `<dss:DocumentHash>`.

The SECOND request MUST contain the `<dss:SignatureObject>` and MUST contain the element `<localsig:CorrelationID>` that refers to the state of a corresponding (FIRST) request. The Digital Signature Service MUST use the referred state to create the signed document. In case of success, the SECOND response MUST contain the electronic signature or signed document, based on the input document found in the FIRST request and the referred state.

5.2.2 Conformance Target: Client

The subject of the conformance test is the client of the Digital Signature Service.

5.2.2.1 Level 1

The conformance level states that the client of the Digital Signature Service is capable of computing a digital signature value based on a given digest value of a document

The FIRST request MUST contain the element `<localsig:RequestDocumentHash>`; the attribute `MaintainRequestState` is either absent or has a value of `"false"`.

Copyright © OASIS Open 2018. All Rights Reserved.

The FIRST request MUST NOT contain the element <localsig:CorrelationID>

The <dss:DocumentHash> obtained from the FIRST response MUST be used to compute the digital signatue value.

The SECOND request MUST contain the <dss:SignatureObject> that contains the digital signatue value.

The SECOND request MUST contain the element <localsig:CorrelationID> with the value that is obtained from the FIRST response.

If the FIRST request contained the <dss:DocumentHash> element, the SECOND request MUST contain that element too with the same value.

5.3 Conformance Profile C - User Agent

This conformance level only applies to the User Agent approach; see Section 3.1, "User Agent".

5.3.1 Conformance Target: Server

The subject of the conformance test is the server that implements the Digital Signature Service.

5.3.1.1 Level 1

The conformance profile states that the Digital Signature Service is capable of using a HTTP User Agent.

The element <dss:ServicePolicy> MUST be present with the value

```
http://docs.oasis-open.org/dss-x/ns/localsig/user-agent
```

The request follows the [**DSSCore**] specification, Section 3.1. The response contains the electronic signature or signed document according to the request, as defined by the [**DSSCore**] specification, Section 3.2.

The server MUST implement the protocol binding according to Section 4.1, "WEB FORM Transport Binding".

The server MUST be able to receive the digital signature value that has been computed by the user agent. The server MUST use the received digital signature value to create the signed document.

5.4 Conformance Profile D - Third Party

This conformance level only applies to the Third Party approach; see Section 3.3, "Third Party".

5.4.1 Conformance Target: Server

The subject of the conformance test is the server that implements the Digital Signature Service.

5.4.1.1 Level 1

The conformance profile states that the Digital Signature Service is capable of delegating the signature creation to a third-party.

The element <dss:ServicePolicy> MUST be present in both requests with the value

```
http://docs.oasis-open.org/dss-x/ns/localsig/delegation
```


Copyright © OASIS Open 2018. All Rights Reserved.

The element <localsig:ChallengeCode>, MUST be present

The server MUST create a new SignRequest and send it to the Third Party. The SignRequest MUST contain the element <localsig:ChallengeCode>. and MUST NOT contain the element <localsig:ResponseCode>.

If the request contains the element <ds:DigestMethod> it must instruct the Third Party to use the specified digest method.

The server MUST wait for the SignResponse of the Third Party. In case of success, the SignResponse MUST contain the element <dss:SignatureObject>.

In case of success, the response MUST contain the electronic signature or signed document as requested, as defined by the [DSSCore] specification, Section 3.2, based on the input document and the <dss:SignatureObject> from the SignResponse of the Third Party.

5.4.1.2 Level 2

The conformance profile states the same capabilities a level 1 and the server is capable acting upon a response code.

The request MUST contain the element <localsig:ResponseCode>.

The SignResponse from the Third Party MUST contain an element <localsig:ResponseCode> with the value that has been entered by the user.

When the server receives the SignResponse from the Third Party, the SignResponse MUST contain the element <localsig:ResponseCode>. The electronic signature or signed document MUST ONLY be created if and only if the value of the element <localsig:ResponseCode>, obtained from the Third Party, is the same as the value of the element <localsig:ResponseCode>, obtained from the client.

5.4.2 Conformance Target: Client

The subject of the conformance test is the client of the Digital Signature Service.

5.4.2.1 Level 1

The conformance profile states that the client is capable of creating a challenge code and presenting the challenge code to the user.

The client MUST create a random value, the challenge code, that can easily be read and entered by a person.

The client MUST present the challenge code to the user. Information MUST be displayed to the user about the use of the code: the user has to compare the presented code with the code that is received from the Third Party. If the codes do not match, the user must cancel the operation.

The request MUST contain the element <localsig:ChallengeCode> with the challenge code.

The client MUST be able to create a request and process a response, as defined by the [DSSCore] specification.

5.4.2.2 Level 2

The conformance profile states the same capabilities a level 1 and the client is capable of creating a response code.

The client MUST create a random value, the response code, that can easily be read and entered by a person.

Copyright © OASIS Open 2018. All Rights Reserved.

The client MUST present the response code to the user. Information MUST be displayed to the user about the use of the code: the user has to enter the code into the application of the Third Party.

The request MUST contain the element `<localsig:ResponseCode>` with the response code.

5.5 Conformance Profile E - Chip Gateway

This conformance level only applies to the Chip Gateway approach; see Section 1.7.3.4, “Introduction of a ChipGateway”.

5.5.1 Conformance Target: Server

The subject of the conformance test is the server that implements the Digital Signature Service.

5.5.1.1 Level 1

The conformance profile states that ....

5.5.1.2 Level 2

The conformance profile states that ....

5.5.2 Conformance Target: Client

The subject of the conformance test is the client of the Digital Signature Service.

5.5.2.1 Level 1

The conformance profile states that ....

5.5.2.2 Level 2

The conformance profile states that ....

Copyright © OASIS Open 2018. All Rights Reserved.

Appendix A XML Schema Definition (Non-Normative)

A.1 Schema

The structures described in this specification are contained in the schema file which is part of [LocalSigXSD]. The xml schema definitions present within this document are copy of the XML schema file and must be considered as informative text, and that in case of discrepancy, definitions within the XML schema prevail.

```
                    <?xml version="1.0" encoding="UTF-8"?>
<!--
    XSD for "DSS Extension for Local Signature Computation Version 1.0"
    Committee Specification Draft 03 / Public Review Draft 03
    13 February 2017
    Copyright (c) OASIS Open 2017. All Rights Reserved.
-->
<xs:schema
    xmlns:localsig="http://docs.oasis-open.org/dss-x/ns/localsig"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
    targetNamespace="http://docs.oasis-open.org/dss-x/ns/localsig"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:annotation>
        <xs:documentation>
XSD for "DSS Extension for Local Signature Computation Version 1.0"
Committee Specification Draft 03 / Public Review Draft 03
13 February 2017
Copyright (c) OASIS Open 2017. All Rights Reserved.

Declared XML Namespace:
http://docs.oasis-open.org/dss-x/ns/localsig

XSD Schema Location:
http://docs.oasis-open.org/dss-x/localsig/v1.0/csprd03/schemas/localsig-v1.0.xsd
        </xs:documentation>
    </xs:annotation>
    <xs:import namespace="http://www.w3.org/2000/09/xmldsig#"
        schemaLocation=
        "http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd"/>
    <xs:import namespace="urn:oasis:names:tc:dss:1.0:core:schema"
        schemaLocation=
        "http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-schema-v1.0-os.xsd"/>
    <xs:element name="RequestDocumentHash">
        <xs:annotation>
            <xs:documentation>
                This element is part of the Two-Step approach
                in a SignRequest (as part of the OptionalInputs).
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element
```


Copyright © OASIS Open 2018. All Rights Reserved.

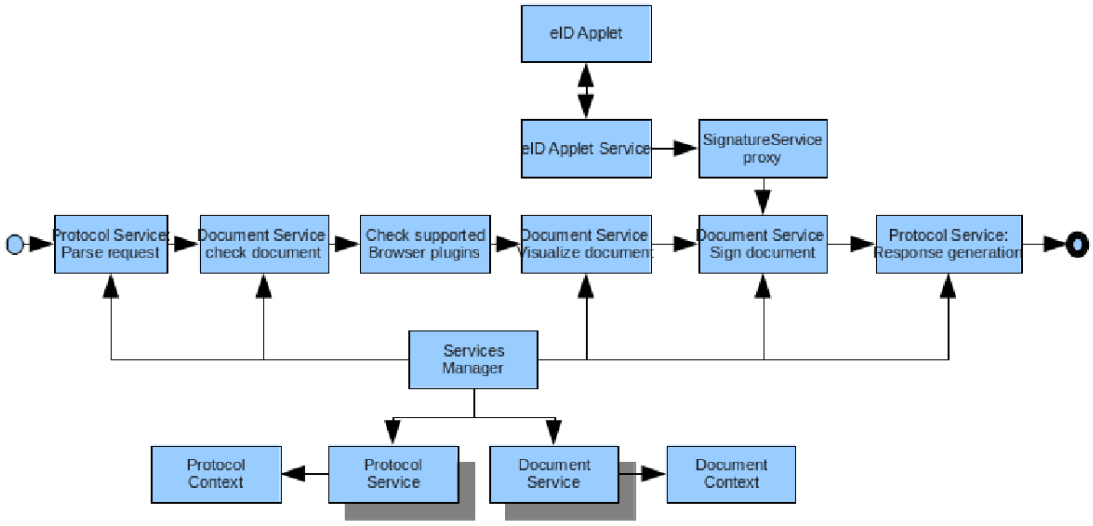
```
                    minOccurs="0"
                    maxOccurs="1"
                    ref="ds:DigestMethod"/>
            </xs:sequence>
            <xs:attribute
                name="MaintainRequestState"
                use="optional"
                type="xs:boolean"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="CorrelationID" type="xs:NCName">
        <xs:annotation>
            <xs:documentation>
                This element is part of the Two-Step approach.
                The CorrelationID is obtained in the first step, from
                a SignResponse (as part of the OptionalOutputs)
                and is provided in the second step, in a SignRequest
                (as part of the OptionalInputs).
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="ChallengeCode" type="xs:NCName">
        <xs:annotation>
            <xs:documentation>
                This element is part of the Third-Party approach
                in a SignRequest (as part of the OptionalInputs).
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="ResponseCode" type="xs:NCName">
        <xs:annotation>
            <xs:documentation>
                This element is part of the Third-Party approach
                in a SignRequest (as part of the OptionalInputs).
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <!--
        The element <dss:DocumentHash> as defined by the
        DSS-core xml schema definition (see corresponding import).
        The element is part of the Two-Step approach in the
        first step, in a SignResponse (as part of the OptionalOutputs).
    -->
    <!--
        The element <dss:SignatureObject> as defined by the
        DSS-core xml schema definition (see corresponding import).
        The element is part of the Two-Step approach in the
        second step, in a SignRequest (as part of the OptionalInputs).
    -->
</xs:schema>
```

 Copyright © OASIS Open 2018. All Rights Reserved.

Appendix B Sample Application (Non-Normative)

Figure B.1, “eID DSS Signature Pipeline” shows the design of a digital signature service that uses the Belgian eID card as client signing token.

Figure B.1. eID DSS Signature Pipeline



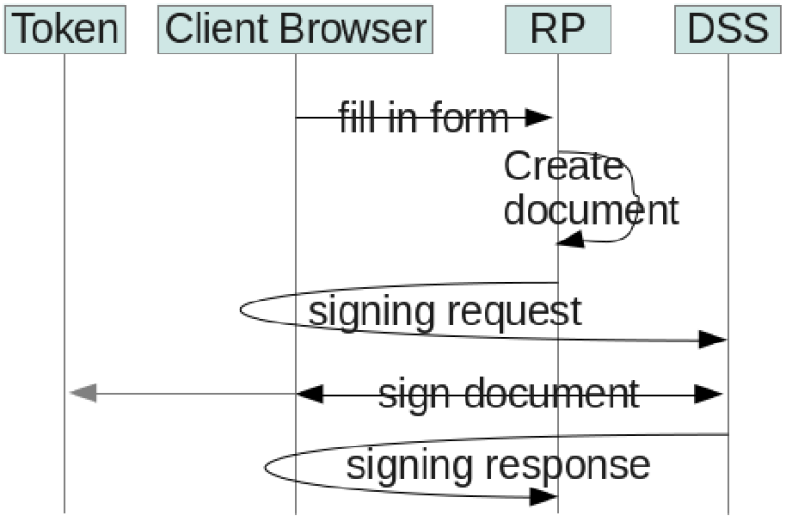
An end user enters the DSS signature pipeline via some protocol. First of all the appropriate protocol service parses the request. At this step the mime type of the incoming document is determined. Via the mime type the appropriate document service can be selected. The document service will first check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

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

A sample protocol flow is shown in Figure B.2, “Sequence diagram of a simple protocol flow”.

Copyright © OASIS Open 2018. All Rights Reserved.

Figure B.2. Sequence diagram of a simple protocol flow



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

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

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

Copyright © OASIS Open 2018. All Rights Reserved.

Appendix C Examples (Non-Normative)

C.1 User Agent

C.1.1 Web Form of the SignRequest

The client sends a request to the Digital Signature Service via the HTTP agent (a web browser) by means of a HTML form.

```
                        <html>
<head>SignRequest</head>
<body>
   <form
       id="dss-request-form"
       method="post"
       action="http://localsig.digitalsignatureservice.co.uk">
       <input type="hidden" name="signrequest"
            value=
            "JVBERi0xLjYNJeLjz9MNCjI4IDAgb2JqDTw8L0ZpbHRlcid
             ZW5ndGggMTY1L04gMi9UeXBlL09ialN0bT4+c3RyZWFt
             i4GBiYFBiYEZTDKBSUZGRrlJUHEgR72YAQz+/wcAlBAG
             eHJlZgo3NjA0MQolJUVPRgo="/>
        <input type="hidden" name="clienturl"
            value="http://localsig.digid.nl/654528644274424/"/>
   </form>
   <script type="text/javascript">
      document.getElementById('dss-request-form').submit();
   </script>
</body>
</html>
```

C.1.2 Web Form of the SignResponse

The Digital Signature Service sends the result back to the client, via the HTTP agent (a web browser, by using the URL of the client `http://localsig.digid.nl/654528644274424/` in the HTML form. The URL was provided in the request by the attribute `clienturl`.

```
                        <html>
<head>SignResponse</head>
<body>
   <form
       id="dss-response-form"
       method="post"
       action="http://localsig.digid.nl/654528644274424/">
       <input type="hidden" name="signresponse"
            value=
            "eHJlZgo3NjA0MQolJUVPRgoDAgb2JqDTw8L0ZpbHRlcid
             i4GBiYFBiYEZTDKBSUZGRrlJUHEgR72YAQz+/wcAlBAG
             ZW5ndGggMTY1L04gMi9UeXBlL09ialN0bT4+c3RyZWFt
             JVBERi0xLjYNJeLjz9MNCjI4I="/>
   </form>
   <script type="text/javascript">
      document.getElementById('dss-response-form').submit();
```


Copyright © OASIS Open 2018. All Rights Reserved.

```
   </script>
</body>
</html>
```

C.2 Two-Step Approach

C.2.1 FIRST Request/Response

The client application initiates a <dss:SignRequest> to request the digest of the document.

```
                        <?xml version="1.0" encoding="utf-8"?>
<dss:SignRequest
    xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:localsig="http://docs.oasis-open.org/dss-x/ns/localsig"
    RequestID="example1-initial-request"
    Profile="http://docs.oasis-open.org/dss-x/ns/localsig">
    <dss:OptionalInputs>
        <dss:ServicePolicy>
            http://docs.oasis-open.org/dss-x/ns/localsig/two-step-approach
        </dss:ServicePolicy>
        <localsig:RequestDocumentHash MaintainRequestState="true">
             <ds:DigestMethod
                    Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
        </localsig:RequestDocumentHash>
    </dss:OptionalInputs>
    <dss:InputDocuments>
        <dss:Document>
            <dss:Base64Data MimeType="application/pdf">
                JVBERi0xLjYNJeLjz9MNCjI4IDAgb2JqD
                [...]
                eHJlZgo3NjA0MQolJUVPRgo=
            </dss:Base64Data>
        </dss:Document>
    </dss:InputDocuments>
</dss:SignRequest>
```

The <dss:SignResponse> contains the <localsig:CorrelationID> to be used in the subsequent <dss:SignRequest>.

```
                        <?xml version="1.0" encoding="utf-8"?>
<dss:SignResponse
    xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:localsig="http://docs.oasis-open.org/dss-x/ns/localsig"
    RequestID="example1-initial-request"
    Profile="http://docs.oasis-open.org/dss-x/ns/localsig">
    <dss:Result>
        <dss:ResultMajor>
             urn:oasis:names:tc:dss:1.0:resultmajor:Success
        </dss:ResultMajor>
        <dss:ResultMinor>
            urn:oasis:names:tc:dss:1.0:resultminor:documentHash
        </dss:ResultMinor>
```


Copyright © OASIS Open 2018. All Rights Reserved.

```
    </dss:Result>
    <dss:OptionalOutputs>
        <localsig:CorrelationID>
            82962C67-F8D6-4480-A130-9280AB1F11A7
        </localsig:CorrelationID>
        <dss:DocumentHash>
            <dss:DocumentHash>
                <ds:DigestMethod
                    Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                <ds:DigestValue>
                    In6GUzH+gMFR5q4WpUTyPa+1b4s=
                </ds:DigestValue>
            </dss:DocumentHash>
        </dss:DocumentHash>
    </dss:OptionalOutputs>
</dss:SignResponse>
```

The client application uses the digest for the (secure) signature-creation device to obtain the signature.

C.2.2 SECOND Request/Response

The client application initiates a <dss:SignRequest> to provide the Digital Signature Service with the signed digest (a <dss:SignatureObject> element) and request for the final document.

```
                        <?xml version="1.0" encoding="utf-8"?>
<dss:SignRequest
    xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
    xmlns:localsig="http://docs.oasis-open.org/dss-x/ns/localsig"
    RequestID="example1-final-request"
    Profile="http://docs.oasis-open.org/dss-x/ns/localsig">
    <dss:OptionalInputs>
        <dss:ServicePolicy>
            http://docs.oasis-open.org/dss-x/ns/localsig/two-step-approach
        </dss:ServicePolicy>
        <localsig:CorrelationID>
            82962C67-F8D6-4480-A130-9280AB1F11A7
        </localsig:CorrelationID>
        <dss:SignatureObject>
            <dss:Base64Signature>
                MIAGCSqGSIb3DQEHAqCAMIIRdQIBATE
                DQEHAaCCD74wggWAMIIEaKADAgECAg
                [...]
                DQEBAQUABEA3YkuiPSDVaAhaAza49UT
                DZWtCGVc0LCc5QRlBOc54ZrVGp6AA==
            </dss:Base64Signature>
        </dss:SignatureObject>
    </dss:OptionalInputs>
</dss:SignRequest>
```

The final <dss:SignResponse> contains the signed document.

```
                        <?xml version="1.0" encoding="UTF-8"?>
<dss:SignResponse
    xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
    RequestID="example1-final-request"
```


Copyright © OASIS Open 2018. All Rights Reserved.

```
    Profile="http://docs.oasis-open.org/dss-x/ns/localsig" >
    <dss:Result>
        <dss:ResultMajor>
        urn:oasis:names:tc:dss:1.0:resultmajor:Success
        </dss:ResultMajor>
        <dss:ResultMinor>
     urn:oasis:names:tc:dss:1.0:resultminor:valid:signature:OnAllDocuments
        </dss:ResultMinor>
        <dss:ResultMessage lang="en-us"/>
    </dss:Result>
    <dss:OptionalOutputs>
        <dss:DocumentWithSignature>
            <dss:Document>
                <dss:Base64Data MimeType="application/pdf">
                  JVBERi0xLjYNJeLjz9MNCjI4IDAgb2JqDTw
                  [...]
                  qmMAg///AYYzCwcKZW5kc3RyZWFtCmVCg==
                </dss:Base64Data>
            </dss:Document>
        </dss:DocumentWithSignature>
    </dss:OptionalOutputs>
</dss:SignResponse>
```


Copyright © OASIS Open 2018. All Rights Reserved.

Appendix D Chipgateway Algorithms

The following table specifies the list of possible values for the Algorithm parameter according to JCA (cf. [**JCA-Names**]) together with its mapping to PKCS#11 (cf. [**PKCS#11-CM**]) and corresponding URIs (cf. [**XML-Algs**] and [**RFC 6931**]).

D.1 Chipgateway Signature Algorithms

Table D.1.

Algorithm	PKCS#11-Mechanism	URI for CardInfo
NONEwithRSA	CKM_RSA_PKCS	http://ws.openecard.org/alg/rsa
SHA1withRSA	CKM_SHA1_RSA_PKCS	http://www.w3.org/2000/09/xmldsig#rsa-sha1
SHA256withRSA	CKM_SHA256_RSA_PKCS	http://www.w3.org/2001/04/xmldsig-more#rsa-sha256
SHA384withRSA	CKM_SHA384_RSA_PKCS	http://www.w3.org/2001/04/xmldsig-more#rsa-sha384
NONEwithRSAandMGF1	CKM_xxx_RSA_PKCS_PSS with CK_RSA_PKCS_PSS_PARAMS indicating the hash algorithm xxx and MGF1, whereas the hash algorithm xxx (e.g. SHA1, SHA256 or SHA384) is implied by the length of the transmitted message.	http://ws.openecard.org/alg/rsa-MGF1
SHA1withRSAandMGF1	CKM_SHA1_RSA_PKCS_PSS with CK_RSA_PKCS_PSS_PARAMS indicating SHA1 and MGF1.	http://www.w3.org/2007/05/xmldsig-more#sha1-rsa-MGF1
SHA256withRSAandMGF1	CKM_SHA256_RSA_PKCS_PSS with CK_RSA_PKCS_PSS_PARAMS indicating SHA256 and MGF1.	http://www.w3.org/2007/05/xmldsig-more#sha256-rsa-MGF1
SHA384withRSAandMGF1	CKM_SHA384_RSA_PKCS_PSS with CK_RSA_PKCS_PSS_PARAMS indicating SHA384 and MGF1.	http://www.w3.org/2007/05/xmldsig-more#sha384-rsa-MGF1
NONEwithECDSA	CKM_ECDSA	http://ws.openecard.org/alg/ecdsa
SHA1withECDSA	CKM_SHA1_ECDSA	http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha1
SHA256withECDSA	CKM_SHA256_ECDSA	http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256
SHA384withECDSA	CKM_SHA384_ECDSA	http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha384

Copyright © OASIS Open 2018. All Rights Reserved.

D.2 ChipGateway Cipher Algorithms

Table D.2.

Algorithm	PKCS#11-Mechanism	URI for CardInfo
RSA/NONE/PKCS1Padding	CKM_RSA_PKCS	http://www.w3.org/2001/04/xmlenc#rsa-1_5
RSA/NONE/OAEPWithSHA1AndMGF1Padding	CKM_RSA_PKCS_OAEP with CK_RSA_PKCS_OAEP_PARAMS indicating SHA1 and MGF1	http://www.w3.org/2009/xmlenc11#mgf1sha1
RSA/NONE/OAEPWithSHA256AndMGF1Padding	CKM_RSA_PKCS_OAEP with CK_RSA_PKCS_OAEP_PARAMS indicating SHA256 and MGF1	http://www.w3.org/2009/xmlenc11#mgf1sha256
RSA/NONE/OAEPWithSHA384AndMGF1Padding	CKM_RSA_PKCS_OAEP with CK_RSA_PKCS_OAEP_PARAMS indicating SHA384 and MGF1	http://www.w3.org/2009/xmlenc11#mgf1sha384

Copyright © OASIS Open 2018. All Rights Reserved.

Appendix E Acknowledgements (Non-Normative)

The following persons have participated in the creation of this specification and are gratefully acknowledged (in alphabetical order):

- Andreas Kuehne, Individual.
- Daniel Nemmert, ecsec GmbH.
- Detlef Hühnlein, ecsec GmbH.
- Ernst Jan van Nigtevecht, Sonnenglanz Consulting BV.
- Ezer Farhi, ARX
- Frank Cornelis, Fedict
- Juan Carlos Cruellas, Departamento de Arquitectura de Computadores, Univ Politecnica de Cataluna.
- Oscar Burgos, CATCert-Agencia Catalana de Certificacio.
- Pim van der Eijk, Sonnenglanz Consulting BV.
- Stefan Hagen, Individual.

Copyright © OASIS Open 2018. All Rights Reserved.

Appendix F Revision History (Non-Normative)

Revision 0.1	13 Januari 2014	E.J. Van Nigtevecht
Creation of the CSPRD 01 based on the CSD 01.		
Revision 0.2	30 June 2014	E.J. Van Nigtevecht
Processed the comments on CSD 01; see "https://www.oasis-open.org/committees/document.php?document_id=53473&wg_abbrev=dss-x". Renamed localsig:ReturnDocumentHash into localsig:RequestDocumentHash (in Section 3.2.1.1.2 in the code).		
Revision 0.3	13 October 2016	E.J. Van Nigtevecht
Corrected url in Section 1.3 under [LocalSigXSD].		
Revision 0.4	13 October 2016	E.J. Van Nigtevecht
Corrected the XSD that was published under CS01: changed element ReturnDocumentHash into element RequestDocumentHash. Updated comments in the XSD and the import statement for http://www.w3.org/2000/09/xmldsig#.		
Revision 0.5	13 October 2016	E.J. Van Nigtevecht
Added a description/indication for the use of the element AdditionalProfile in Section 3.1.1, Section 3.2.1 and Section 3.3.1.		
Revision 0.6	13 February 2017	E.J. Van Nigtevecht
Editorial updates for CSPRD03.		
Revision 0.7	20 February 2017	E.J. Van Nigtevecht
Editorial updates for CSPRD03 into CS02.		
Revision 0.8	22 February 2018	D. Hühnlein
Added draft of ChipGateway mechanism.		

Copyright © OASIS Open 2018. All Rights Reserved.