Digital Signature Service Core Protocols, Elements, and Bindings Version 2.0

Committee Specification Draft 02 WD01

07 October 2018

Specification URIs

This version:

- http://docs.oasis-open.org/dss-x/dss-core/v2.0/csprd02/dss-core-v2.0-csprd02.md (Authoritative)
- http://docs.oasis-open.org/dss-x/dss-core/v2.0/csprd02/dss-core-v2.0-csprd02.html
- http://docs.oasis-open.org/dss-x/dss-core/v2.0/csprd02/dss-core-v2.0-csprd02.pdf

Previous version:

- http://docs.oasis-open.org/dss-x/dss-core/v2.0/csprd01/dss-core-v2.0-csprd01.docx (Authoritative)
- http://docs.oasis-open.org/dss-x/dss-core/v2.0/csprd01/dss-core-v2.0-csprd01.html
- http://docs.oasis-open.org/dss-x/dss-core/v2.0/csprd01/dss-core-v2.0-csprd01.pdf

Latest version:

- http://docs.oasis-open.org/dss-x/dss-core/v2.0/dss-core-v2.0.md (Authoritative)
- http://docs.oasis-open.org/dss-x/dss-core/v2.0/dss-core-v2.0.html
- http://docs.oasis-open.org/dss-x/dss-core/v2.0/dss-core-v2.0.pdf

Technical Committee:

- OASIS Digital Signature Services eXtended (DSS-X) TC

Chairs

- Andreas Kuehne (kuehne@trustable.de), Individual
- Stefan Hagen (stefan@hagen.link), Individual

Editors

- Andreas Kuehne (kuehne@trustable.de), Individual
- Stefan Hagen (stefan@hagen.link), Individual

Additional artifacts:

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

- JSON and XML schemas: http://docs.oasis-open.org/dss-x/dss-core/v2.0/csprd02/schema/

Related work:

This specification replaces or supersedes:

- *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.

Declared XML namespaces:

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

Abstract

This document defines JSON and XML based request/response protocols for signing and verifying documents and other data. It also defines a timestamp format, and a signature property for use with these protocols. Finally, it defines transport and security bindings for the protocols.

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.

TC members should send comments on this specification to the TC's email list. Others should send comments to the TC's public comment list, after subscribing to it by following the instructions at the "Send A Comment" button on the TC's web page at https://www.oasis-open.org/committees/dss-x/.

This specification is provided under the RF on Limited Terms Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. 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 TC’s web page (https://www.oasis-open.org/committees/dss-x/ipr.php).

Note that any machine-readable content (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:

[DSS-v2.0] *Digital Signature Service Core Protocols, Elements, and Bindings Version 2.0*. Edited by Andreas Kuehne and Stefan Hagen. 07 October 2018. OASIS Committee Specification Draft 02 / Public Review Draft 02. http://docs.oasis-open.org/dss-x/dss-core/v2.0/csprd02/dss-core-v2.0-csprd02.html. Latest version: http://docs.oasis-open.org/dss-x/dss-core/v2.0/dss-core-v2.0.html.

Notices

Table of Contents

---

1 Introduction

1.1 IPR Policy

This specification is provided under the RF on Limited Terms Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. 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 TC's web page (https://www.oasis-open.org/committees/dss-x/ipr.php).

1.2 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in IETF RFC 2119 [RFC2119] and RFC 8174 [RFC8174].

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119] and [RFC8174].

1.2.1 Terms and Definitions

For the purposes of this document no specific terms or definitions have been identified as deviating from the usual meaning in the context of XML / JSON schema, digital signatures or transport.

1.2.2 Abbreviated Terms

ASN.1 — Abstract Syntax Notation One

URI — (IETF) Uniform Resource Identifier

XML — (W3C) Extensible Markup Language

XSD — (W3C) XML Schema

1.3 Normative References

[DSBXSD] A. Kuehne, S. Hagen. *DSS 2.0 Base XML Schema*. OASIS.

[DSIGRWXSD] A. Kuehne, S. Hagen. *DSS 2.0 adapted XMLDSig XML Schema*. OASIS.

[DSS1Async] A. Kuehne. *Asynchronous Processing Abstract Profile*. OASIS, oasis-dss-profiles-asynchronous_processing-spec-v1.0-os.html

[DSS1Core] S. Hagen. *DSS 1.0 Core Protocols*. OASIS, oasis-dss-core-spec-v1.0-os.html.

[DSS2JSON] A. Kuehne, S. Hagen. *DSS 2.0 Core JSON Schema*. OASIS.

[DSS2XSD] A. Kuehne, S. Hagen. *DSS 2.0 Core XML Schema*. OASIS.

[ESIFrame]** ****TR 119 102 V1.2.1******Electronic Signatures and Infrastructures (ESI); The framework for standardization of signatures; Definitions and abbreviations

http://www.etsi.org/deliver/etsi_tr/119000_119099/119001/01.02.01_60/tr_119001v010201p.pdf

RFC2119

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, https://www.rfc-editor.org/info/rfc2119.

RFC2396

Berners-Lee, T., "Uniform Resource Identifiers (URI): Generic Syntax.", RFC 2396, DOI 10.17487/RFC2396, August 1998, http://www.rfc-editor.org/info/rfc2396. replace with: **[RFC3986]** Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, https://www.rfc-editor.org/info/rfc3986.

RFC2440

Callas J., Donnerhacke L., Finney H., Thayer R., "OpenPGP Message Format", RFC 2440, DOI 10.17487/RFC2440, November 1998, http://www.rfc-editor.org/info/rfc2440. replace with: **[RFC4880]** Callas, J., Donnerhacke, L., Finney, H., Shaw, D., and R. Thayer, "OpenPGP Message Format", RFC 4880, DOI 10.17487/RFC4880, November 2007, https://www.rfc-editor.org/info/rfc4880.

RFC2616

Fielding R., "Hypertext Transfer Protocol – HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, http://www.rfc-editor.org/info/rfc2616. replace with (maybe the right individual RFC): **[RFC7230]** Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, https://www.rfc-editor.org/info/rfc7230. **[RFC7231]** Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, https://www.rfc-editor.org/info/rfc7231. **[RFC7232]** Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, DOI 10.17487/RFC7232, June 2014, https://www.rfc-editor.org/info/rfc7232. **[RFC7233]** Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, DOI 10.17487/RFC7233, June 2014, https://www.rfc-editor.org/info/rfc7233. **[RFC7234]** Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, https://www.rfc-editor.org/info/rfc7234. **[RFC7235]** Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, https://www.rfc-editor.org/info/rfc7235.

RFC2648

Moats, R., "A URN Namespace for IETF Documents", RFC 2648, DOI 10.17487/RFC2648, August 1999, https://www.rfc-editor.org/info/rfc2648.

RFC2822

Resnick, P., "Internet Message Format", BCP_ETC, RFC 2822, DOI 10.17487/RFC2822, April 2001, http://www.rfc-editor.org/info/rfc2822. replace with: **[RFC5322]** Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, https://www.rfc-editor.org/info/rfc5322.

RFC3161

Adams, C., Cain, P., Pinkas, D., and R. Zuccherato, "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)", RFC 3161, DOI 10.17487/RFC3161, August 2001, https://www.rfc-editor.org/info/rfc3161.

RFC5652

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, https://www.rfc-editor.org/info/rfc5652. Remark: As used in DSS, all implementations based upon RFC 5652 and previous releases of CMS will suffice. For the sake of simplicity the "urn:ietf:rfc:3369" is used throughout the document to indicate a CMS message as specified in RFC 5652 or RFC 3369 or any version (including PKCS #7.

RFC8174

Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, http://www.rfc-editor.org/info/rfc8174.

RFC8259

Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 8259, DOI 10.17487/RFC8259, December 2017, http://www.rfc-editor.org/info/rfc8259.

[SAML2RWXSD] A. Kuehne, S. Hagen. *DSS 2.0 adapted SAML 2.0 XML Schema*. OASIS.

**[SOAP] **M. Gudgin et al. *SOAP Version 1.2 Part 1: Messaging Framework.* W3C Recommendation, June 2003. http://www.w3.org/TR/xmlschema-1/

[SOAPAtt] H. F. Nielsen, H. Ruellan *SOAP Message Transmission Optimization Mechanism,* W3C Working Group Note, 8 June 2004 http://www.w3.org/TR/soap12-af/

[SOAPMtom] Martin Gudgin, Noah Mendelsohn *SOAP 1.2 Attachment Feature*, W3C Recommendation 25 January 2005 http://www.w3.org/TR/soap12-mtom/

**[WS-I-Att] **Ch. Ferris, A. Karmarkar, C. K. Liu *Attachments Profile Version 1.0*, The Web Services-Interoperability Organization (WS-I)*,*20 April 2006 http://www.ws-i.org/Profiles/AttachmentsProfile-1.0.html

[XML] Extensible Markup Language (XML) 1.0 (Fifth Edition), T. Bray, J. Paoli, M. Sperberg-McQueen, E. Maler, F. Yergeau, Editors, W3C Recommendation, November 26, 2008, http://www.w3.org/TR/2008/REC-xml-20081126/. Latest version available at http://www.w3.org/TR/xml.

[XML-C14N]**** **J. Boyer. *Canonical XML Version 1.0*. W3C Recommendation, March 2001. http://www.w3.org/TR/xml-c14n

[XML-xcl-c14n] Exclusive XML Canonicalization Version 1.0. W3C Recommendation 18 July 2002 http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/

[**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-NT-Document] http://www.w3.org/TR/2004/REC-xml-20040204/#NT-document

[XML-PROLOG] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, et al. Prolog and Document Type Declaration in Extensible Markup Language (XML) 1.0 (Third Edition), W3C Recommendation, 04 February 2004, http://www.w3.org/TR/REC-xml/#sec-prolog-dtd

**[xml:id] **xml:id, Version 1.0, W3C Recommendation, 9 September 2005, http://www.w3.org/TR/xml-id/

[XMLDSIG] D. Eastlake et al. XML-Signature Syntax and Processing. W3C Recommendation, February 2002*. *http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/

[XML-Schema-1] W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures, S. Gao, M. Sperberg-McQueen, H. Thompson, N. Mendelsohn, D. Beech, M. Maloney, Editors, W3C Recommendation, April 5, 2012, http://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/. Latest version available at http://www.w3.org/TR/xmlschema11-1/.

[XML-Schema-2] W3C XML Schema Definition Language (XSD) 1.1 Part 2: DatatypesW3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes, D. Peterson, S. Gao, A. Malhotra, M. Sperberg-McQueen, H. Thompson, Paul V. Biron, Editors, W3C Recommendation, April 5, 2012, http://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/. Latest version available at http://www.w3.org/TR/xmlschema11-2/.

[**XPATH]** XML Path Language (XPath) Version 1.0. W3C Recommendation 16 November 1999 http://www.w3.org/TR/xpath

1.4 Non-Normative References

[ASN.1] Introduction to ASN.1. https://www.itu.int/en/ITU-T/asn1/Pages/introduction.aspx

[CHPGW] DSS Extension for Local Signature Computation Version 1.0, Working Draft for Committee Specification 04. https://www.oasis-open.org/committees/download.php/62576/localsig-v1.0-csprd04.pdf

[ISO8601] Data elements and interchange formats — Information interchange — Representation of dates and times, International Standard, ISO 8601:2004(E), December 1, 2004, https://www.iso.org/standard/40874.html.

[ISO639-1] Codes for the representation of names of languages — Part 1: Alpha-2 code, International Standard, ISO 639-1:2002 (en), https://www.iso.org/obp/ui#iso:std:iso:639:-1.

[JENSEN-2009] Meiko Jensen, Lijun Liao, and Jörg Schwenk. 2009. The curse of namespaces in the domain of XML signature. In Proceedings of the 2009 ACM workshop on Secure web services (SWS '09). ACM, New York, NY, USA, 29-36. DOI: https://doi.org/10.1145/1655121.1655129

[RFC7049] C. Bormann, University Bremen TZI, Concise Binary Object Representation (CBOR), ISSN: 2070-1721, October 2013. https://tools.ietf.org/html/rfc7049

[RFC7515]** **M. Jones, Microsoft, JSON Web Signature (JWS), ISSN: 2070-1721, May 2015. https://tools.ietf.org/html/rfc7515.

1.5 Typographical Conventions

Keywords defined by this specification use this monospaced font.

Normative source code uses this paragraph style.

Text following the special symbol («) – an opening Guillemet (or French quotation mark) – within this specification identifies automatically testable requirements to aid assertion tools. Every such statement is separated from the following text with the special end symbol (») – a closing Guillemet and has been assigned a reference that follows that end symbol in one of the three patterns:

1. [DSS-section#-local#] if it applies regardless of syntax

2. [JDSS-section#-local#] if it applies only to JSON syntax

3. [XDSS-section#-local#] if it applies only to XML syntax

Some sections of this specification are illustrated with non-normative examples.

Example 1: text describing an example uses this paragraph style

Non-normative examples use this paragraph style.

All examples in this document are non-normative and informative only.

Representation-specific text is indented and marked with vertical lines.

Representation-Specific Headline

Normative representation-specific text

All other text is normative unless otherwise labeled e.g. like:

Non-normative Comment:

This is a pure informative comment that may be present, because the information conveyed is deemed useful advice or common pitfalls learned from implementer or operator experience and often given including the rationale.

1.6 DSS Overview (Non-normative)

This specification describes two request/response protocols:

1. signing protocol

2. verifying protocol

Using the first protocol a client can send documents (or document hashes) to a server and receive back a signature on the documents. Using the second protocol a client can send documents (or document hashes) and a signature to a server and receive back an answer on whether the signature is valid or not.

The top-level components for the signing protocol are

· SignRequest (see section 4.2.6) as input and

· SignResponse (see section 4.2.7) as output.

For the verification protocol the top-level components are

· VerifyRequest (see section 4.2.10) as ínput and

· VerifyResponse (see section 4.2.11) as output.

Additionally, this version of the core includes asynchronous requests initially specified in the Asynchronous Processing Abstract Profile [DSSAsync].

The elements in which the protocols are formulated are provided in a sematic level and also in JSON and XML syntax. Provided are additional mappings from the generic to the specific entities.

These protocol operations could be useful in a variety of contexts – for example, they could allow clients to access a single corporate key for signing press releases, with centralized access control, auditing and archiving of signature requests. They could also allow clients to create and verify signatures without the need for complex client software and security-sensitive configuration.

The signing and verifying protocols are chiefly designed to support the creation and verification of XML signatures **[XMLDSIG]**, XML timestamps (see [DSS1Core], section 5.1), binary timestamps **[RFC 3161]** and CMS signatures **[RFC 5652]**. These protocols are intended be extensible to other types of signatures and timestamps, such as PGP signatures **[RFC 2440]**.

It is expected that the signing and verifying protocols will be *profiled* to meet many different application scenarios. In anticipation of this, these protocols have only a minimal set of required elements, which deal with transferring “input documents” and signatures back and forth between client and server. The input documents to be signed or verified can be transferred in their entirety or the client can hash the documents themselves and only send the hash values to save bandwidth and protect the confidentiality of the document content.

All functionality besides transferring input documents and signatures is relegated to a framework of “optional inputs” and “optional outputs”. This document defines a number of optional inputs and outputs. Profiles of these protocols can pick and choose which optional inputs and outputs to support and can introduce their own optional inputs and outputs when they need functionality not anticipated by this specification.

Examples of optional inputs to the signing protocol include: what type of signature to produce, which key to sign with, who the signature is intended for, and what signed and unsigned properties to place in the signature. Examples of optional inputs to the verifying protocol include: the time for which the client would like to know the signature’s validity status, additional validation data necessary to verify the signature (such as certificates and CRLs), and requests for the server to return information such as the signer’s name or the signing time.

The signing and verifying protocol messages must be transferred over some underlying protocol(s) which provide message transport and security. A *binding* specifies how to use the signing and verifying protocols with some underlying protocol such as HTTP POST or TLS. Section 7 Asynchronous Processing Model provides an initial set of bindings.

The previous version of specification ([DSS1Core]) defines two elements that are related to these protocols. First, an XML timestamp element is defined in [DSS1Core], section 5.1. The signing and verifying protocols can be used to create and verify both XML and binary timestamps; a profile for doing so is defined in **[XML-TSP]**. Second, a RequesterIdentity element is defined in (see [DSS1Core], section 5.2). This element can be used as a signature property in an XML signature, to give the name of the end-user who requested the signature. These elements remain unchanged and are not repeated in this specification.

2 Design Considerations

2.1 Version 2.0 goal [non-normative]

The main changes of this version of the DSS/X core document compared to version 1.0 are:

- Considering the set of comments and bug reports arrived since version DSS 1.0 became standard
- Inclusion of requirements that became known only after publication of version 1.0
- Simplification of the core schema, e.g. by dropping elements seldom used

· Support for syntaxes other than XML

· Support transport formats other than SOAP

· Integration of the ‘Asynchronous Processing Profile’ [DSSAsync] into the core

Define a sematic model that can be mapped to different syntaxes. In this document the focus is on XML and JSON, but support for other syntaxes should be possible. Therefore, only the common denominator of syntax features can be used:

· Focus on Base64 as the most versatile way to transport documents and signatures

· Avoid the use of XML specifics (like e.g. mixed content)

· Provide namespace / URI for XPath evaluation explicitly

· Avoid xs:any by replacing it with an enumeration of possible types, and if that is not feasible, use base64 blobs as a fallback

To support implementers and to ease the use of the protocol with common frameworks the following list of requirements was compiled:

· One unique object model for all transport syntaxes

· Define type and cardinality of OptionalInputs and OptionalOutputs child elements explicitly

· Rearrange sequences and choices to produce a strongly typed object model

Regardless of the use of JSON as a transport syntax the handling of JSON signatures will not be covered by this document. Specific profiles will address signatures e.g. conformant to [RFC7515].

The provided schemes of DSS-X version 2 reflect these requirements. The XML schemes of version 1 and 2 share many similarities but are not compatible.

2.2 Transforming DSS 1.0 into 2.0

This section describes the several actions taken to fulfil the goals listed in the previous section.

2.2.1 Circumventing xs:any

The XML schema type ‘any’ allows an object to contain arbitrary structures. This comes handy for writers of specifications as an extension point because the structures transported don’t need to be defined upfront. But this advantage at the specification stage comes with a price at the implementation stage. The structures intended to be supported by a client or a server system MUST be known to be implementable. But the usual tools for schema support leave the task of handling the content of an any type to the developer. Without extensive testing problems with unexpected content may occur at runtime, even while using typed languages.

As a successor of the OptionalInputs element (see section 2.7 of version 1.0 of this document) the component OptionalInputsVerify (see section 4.3.5) defines its child elements and their cardinality explicitly. When using additional profiles, the relevant components of the core schema can be redefined using the XML schema’s ‘redefine’ element or JSON schema’s ‘allOf’ as described in section 2.5.1 .

Another usage scenario for ‘xs:any’ is the transport of unknown data objects. As sample use case is the Property component (see section 4.3.17). This component is intended to contain signature attributes of unknown structure. In this version of the specification the ‘xs:any’ type is replaced by a structure containing base64-encoded data and meta data (component Any, see section 4.1.2). When using XML as the transport syntax this seems to be a disadvantage. But direct XML fragment copying may introduce namespace problems and security concerns. Most importantly the cherry-picking of transport syntax features would inhibit a transport independent object model, both on the client and the server side. More complex programming and testing would be inevitable.

2.2.2 Substituting the mixed Schema Attribute

Mixing sub-elements and text within a single element is a great advantage of XML. But when XML is applied for serializing an object model this ‘markup language’ feature is of little use. Other serialization syntaxes (like JSON) don’t support such a feature. There is the need to substitute the ‘mixed’ construct to become syntax independent. The substitution is done by removing the mixed attribute and introduce an additional ‘value’ element to contain the textual content.

2.2.3 Introducing the NsPrefixMappingType Component

Namespaces are an outstanding feature of the XML world. A replacement is required for all syntaxes that don’t such a feature. The use of naming conventions and prefixes are used to avoid naming collisions. A special challenge is the use of XPath-Expression as elements. The XPath expression itself is represented as a simple string. But the expression may depend on namespace/prefix mappings that are defined within the namespace context of the XML element. The NsPrefixMappingType component (see section 4.1.1) represents the required namespace/prefix mapping. It is recommended to use this element for XML syntax, too. This simplifies the handling on the consumer side and circumvents problems with namespace prefix assignments handled by web frameworks.

2.2.4 Imported XML schemes

A special challenge is imposed by the imported schemes, like the **[XMLDSIG]** scheme, that uses features not supportable by the mentioned ‘multi-syntax’ approach. For example, the **[XMLDSIG] **type ‘Transform’ is defined like this:

```
<xs:complexType name="TransformType" mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:any namespace="##other" processContents="lax"/>
        <!-- (1,1) elements from (0,unbounded) namespaces -->
        <xs:element name="XPath" type="string"/>
    </xs:choice>
    <xs:attribute name="Algorithm" type="xs:anyURI" use="required"/>
</xs:complexType>
```

Most of the restrictions listed above do apply here:

· The complexType may contain mixed content (child elements **and** text). This concept is not supported by JSON. The workaround for this limitation is to drop the ‘mixed’ attribute and to introduce a ‘value’ element.

· The choice construct is mapped in an untyped way by Java’s JAXB framework. Therefore, the choice element is changed to a sequence.

· The any type is replaced by a base64 encoded blob.

· The option to provide arbitrary namespace / prefix mappings to support the evaluation of XPath expression is not available in e.g. JSON syntax. Therefore an element mapping prefixes to namespaces (of type dsb:NsPrefixMappingType) is added.

```
<xs:complexType name="TransformType">
  <xs:sequence>
    <xs:element maxOccurs="1" minOccurs="0" name="value" type="string"/>
    <xs:element maxOccurs="1" minOccurs="0" name="Base64Content" type="xs:base64Binary"/>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="XPath"
      type="string"/>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="NsPrefixMapping"
      type="dsb:NsPrefixMappingType"/>
  </xs:sequence>
  <xs:attribute name="Algorithm" type="xs:string" use="required"/>
</xs:complexType>
```

To apply the necessary changes to the imported schemes the XML schema language provides the override functionality to change existing schemes. But Java's JAXB framework's schema compiler does not support override so the adapted schemes are provided alongside DSS-X core schemes.

2.2.5 Syntax variants

This version of the DSS/X core document handles the representation of requests and response elements according to the JSON and XML syntax. The general semantics of the elements is discussed in the element's main section. Details of the JSON or XML formats are discussed in specific subsections

*·   *Component – JSON Syntax*

*·   *Component – XML Syntax*

*2.2.6*JSON Syntax Extensions

JSON, as described in [RFC8259], defines a text format for serializing structured data. Objects are serialized as an unordered collection of name/value pairs.

JSON does not define any semantics around the name/value pairs that make up an object, nor does it define an extensibility mechanism for adding control information to a payload.

DSS's JSON format extends JSON by defining general conventions for name/value pairs that annotate a JSON object, property or array. DSS defines a set of canonical annotations for control information such as ids, types, and links, and custom annotations MAY be used to add domain-specific information to the payload.

Annotations are used in JSON to capture control information that cannot be predicted as well as a mechanism to provide values where a computed value would be wrong.

2.3 Construction Principles

2.3.1 Multi Syntax approach

In the years since DSS 1.0 became standard many other formats (like JSON) became popular for data interchange. Nevertheless, XML is still an important and commonly used format. To support these developments DSS 2.0 is taking a multi-syntax approach:

·   For each structural component there is semantic section describing the elements, restrictions and relations to other components in a syntax-neutral way.

·   Following the sematic definition there are syntax-specific sections describing the mapping of the given requirements to XML and JSON.

·   Schemes are provided for XML and JSON.

·   Element name mappings are given for JSON.

Subsequent versions of this protocol may define additional syntax mappings, e.g. for ASN.1 or CBOR.

The restriction of this approach is limitation to the common denominator of capabilities of the used transfer formats. The section ‘Transforming DSS 1.0 into 2.0’ targets these limitations. The imported schema files defined by other parties are also affected. An example is the ‘Component Transform’, that was originally defined in [XMLDSIG] and the aspects described in 2.2.1 Circumventing xs:any, 2.2.2 Substituting the mixed Schema Attribute and 2.2.3 Introducing the NsPrefixMappingType Component apply.

2.4 Schema Organization and Namespaces

The structures described in this specification are contained in the schema file **[Core2.0-XSD]**. All schema listings in the current document are excerpts from the schema file. In the case of a disagreement between the schema file and this document, the schema file shall take precedence.

This schema is associated with the following XML namespace

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

and

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

If a future version of this specification is needed, it will use a different namespace.

Conventional XML namespace prefixes are used in the schema:

- The prefix dss2: stands for the DSS core version 2.0 namespace**[DSS2XSD]**.refDSS2XSD
- The prefix dsb: stands for the DSS base namespace**[DSBXSD].**refDSS2XSD
- The prefix ds-rw: stands for a namespace of elements based on the W3C XML Signature **[XMLDSIG]**.
- The prefix xs: stands for the W3C XML Schema namespace **[Schema1]**.
- The prefix saml2-rw: stands for a namespace of elements based on the OASIS SAML 2 Schema namespace **[SAMLCore2.0]**.

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

The following schema fragment defines the XML namespaces and other header information for the DSS core schema:

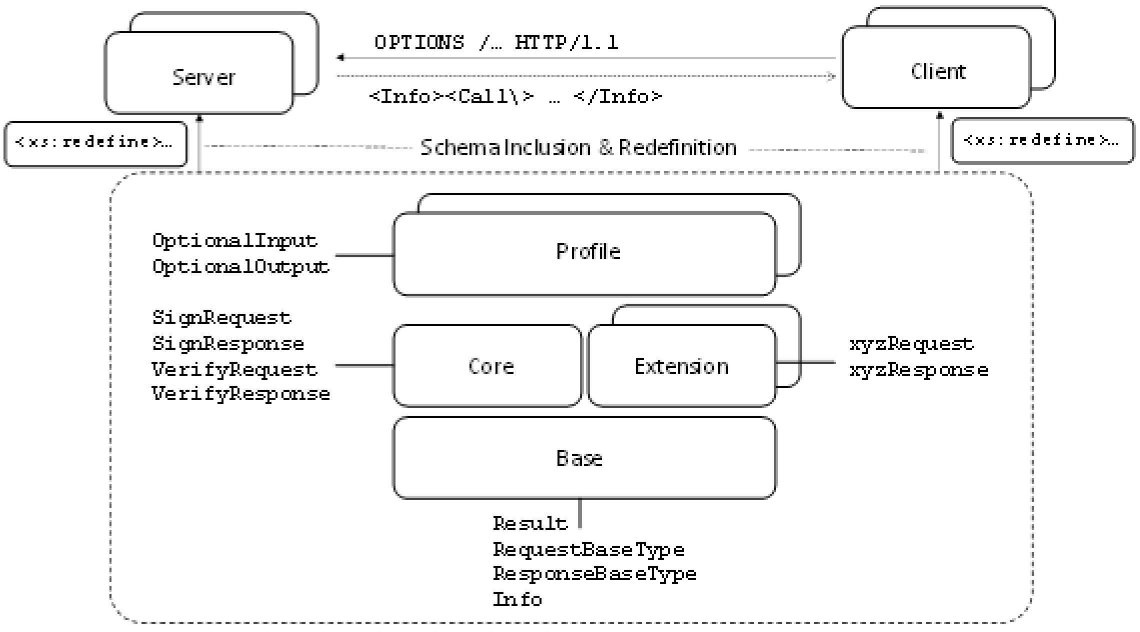
```
<xs:schema xmlns:dss2="http://docs.oasis-open.org/dss-x/ns/core"
  xmlns:dsb="http://docs.oasis-open.org/dss-x/ns/base"
  xmlns:ds-rw="http://docs.oasis-open.org/dss-x/ns/xmldsig/rewritten"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:saml-rw="http://docs.oasis-open.org/dss-x/ns/SAML\_1.0/assertion/rewritten"
  xmlns:saml2-rw="http://docs.oasis-open.org/dss-x/ns/saml2/rewritten"
  targetNamespace="http://docs.oasis-open.org/dss-x/ns/core"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
    <xs:annotation>
      <xs:documentation xml:lang="en">This Schema defines the Digital Signature Service (
    </xs:annotation>
    <xs:import namespace="http://docs.oasis-open.org/dss-x/ns/xmldsig/rewritten"
      schemaLocation=" xmldsig-core-schema-dss-rw.xsd"/>
    <xs:import namespace="http://docs.oasis-open.org/dss-x/ns/SAML\_1.0/assertion/rewrit
      schemaLocation="oasis-sstc-saml-schema-protocol-1.1-dss-rw.xsd"/>
    <xs:import namespace="http://docs.oasis-open.org/dss-x/ns/saml2/rewritten"
```

```
      schemaLocation="saml-schema-assertion-2.0-dss-rw.xsd"/>
    <xs:import namespace="http://www.w3.org/XML/1998/namespace"
      schemaLocation="http://www.w3.org/2001/xml.xsd"/>
```

2.5 DSS Component Overview

The DSS core is designed to be extended by profiles to support additional functionalities. The DSS specification comes with a set of profiles (see https://www.oasis-open.org/standards#dssv1.0). With version 2.0 there will be extensions to augment the use cases beyond the sign and verify scope of the previous version. The extensions will define other requests and responses while using e.g. the ResultType. A sample for an extension is the ChipGateway Protocol (c.f. clause 3.4 of [CHPGW]). To support this approach, the DSS 2.0 schema is split into a generic 'base' and the more specific 'core' schema.

Figure 1:Component overview



The diagram above shows the relationship between the different building blocks.

2.5.1 Schema Extensions

Most profiles define additional OptionalInputs or OptionalOutputs. To support a type-safe extension of the set of optional elements it is recommended to use the XML schema redefine mechanism to extend the core schema and derive the related JSON schema from it:

```
<xs:redefine schemaLocation="core-schema.xsd">
  <xs:complexType name="dss:OptionalOutputsVerifyType">
    <xs:complexContent>
      <xs:extension base="dss:OptionalOutputsVerifyType">
        <xs:group ref="prf:optionalOutputGroup"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:redefine>
```

The snippet above extends the set of sub-components of OptionalOutputsVerifyType with the group of elements of the profile.

In a similar way extension of the core's JSON scheme can be performed by using the 'allOf' keyword:

```
"dss2-OptionalOutputsVerifyType": {
  "allOf": [
    {"\$ref": "\#/definitions/prf-OptionialElement"},
    {
      "type": "object",
      "properties": {
        "policy": {
          "type": "array",
          "items": {
            "type": "string"
          }
        },
        // [...]
      }
    }
  ]
}
```

With this mechanism it is possible to extend the core schema to specific requirements while preserving the advantage of type safety and tool / IDE support. This sample illustrates the use of 'extension'. in the same way restriction can be applied. In more complex scenarios (e.g. multiple profiles apply, need for extending **and**restriction the core schema) the use of other techniques (e.g. XSLT) may be required.

It may be useful to process a profile (or a set of profiles) using a distinct endpoint. This enables the server instance to provide a specific WSDL including an appropriate schema with all profile-related elements.

3 Data Type Models

3.1 Boolean Model

The boolean data type is used to specify a true or false

3.2 Integer Model

The integer data type is used to specify a numeric value without a fractional component.

3.3 String Model

The string data type can represent characters, line feeds, carriage returns, and tab characters.

3.4 Binary Data Model

The base64Binary type holds Base64-encoded binary data

3.5 URI Model

Uniform Resource Identifier (URI) is a string of characters used to identify a resource

3.6 Unique Identifier Model

A unique identifier is a numeric or alphanumeric string that is associated with a single entity within a given system.

3.7 Date and Time Model

The specific concept of date and time used in this document is defined in this section and noted in subsequent usage as**:**

DateTime

« All date time values inside a DSS document MUST adhere to the ISO 8601 [ISO8601] basic or extended Format (as given there in section 4.3.2 “Complete representations” and with the addition of decimal fractions for seconds, similar to ibid. section 4.2.2.4 “Representations with decimal fraction” but with the full stop (.) being the preferred separator for DSS). » [DSS-3.7-1].

3.8 Lang Model

The specific concept of language used in this document is defined in this section and noted in subsequent usage as**:**

Language

« All language values inside a DSS document MUST adhere to the ISO 639-1 [ISO639-1] format (as given there in section 4 “Two-letter language code”. » [DSS-3.8-1].

4 Data Structure Models

Operation requests and responses

The XML elements of this section are defined in the XML namespace 'http://docs.oasis-open.org/dss/ns/core'.*[category operation in namespace http://docs.oasis-open.org/dss/ns/core explanation]*

Component SignRequest

The SignRequest component is sent by the client to request a signature or timestamp on some input documents.

Below follows a list of the sub-components that MAY be present within this component:

The optional InputDocuments element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section InputDocumentsType. *[sub component InputDocuments details]*

The optional OptionalInputs element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section OptionalInputsSignType. *It is intended to transport additional input elements of the signing request.*

Non-normative Comment:

[component SignRequest non normative details]

SignRequest – JSON Syntax

The SignRequestType JSON object SHALL implement in JSON syntax the requirements defined in the SignRequest component.

Properties of the JSON object SHALL implement the sub-components of SignRequestType using JSON-specific names mapped as shown in the table below.

Element	**Implementing JSON member name**	**Comments**
InputDocuments	inDocs	*[]*
OptionalInputs	optInp	*[]*

The SignRequestType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-SignRequestType" : {
  "type" : "object",
  "properties" : {
    "profile" : {
      "type" : "array",
      "items" : {
        "type" : "string"
      }
    },
    "reqID" : {
      "type" : "string"
    },
    "inDocs" : {
      "$ref" : "#/definitions/dss2-InputDocumentsType"
    },
    "optInp" : {
      "$ref" : "#/definitions/dss2-OptionalInputsSignType"
    }
  }
}
```

[component SignRequest JSON schema details]

SignRequest – XML Syntax

The XML type SignRequestType SHALL implement the requirements defined in the SignRequestcomponent.

The SignRequestType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="SignRequestType">
  <xs:complexContent>
    <xs:extension base="dsb:RequestBaseType">
      <xs:sequence>
        <xs:element minOccurs="0" name="InputDocuments" type="dss2:InputDocumentsType"/>
        <xs:element minOccurs="0" name="OptionalInputs" type="dss2:OptionalInputsSignType
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Each child element of SignRequestType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component SignRequest XML schema details]*

Component SignResponse

The SignResponse component returns the requested signature or timestamp to the requestor.

Below follows a list of the sub-components that MAY be present within this component:

The optional OptionalOutputs element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section OptionalOutputsSignType. *The OptionalOutputs element contains additional signing related outputs returned by the server.*

The optional SignatureObject element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section SignatureObjectType. *[sub component SignatureObject details]*

Non-normative Comment:

[component SignResponse non normative details]

SignResponse – JSON Syntax

The SignResponseType JSON object SHALL implement in JSON syntax the requirements defined in the SignResponse component.

Properties of the JSON object SHALL implement the sub-components of SignResponseType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
OptionalOutputs	optOutp	*[]*
SignatureObject	sigObj	*[]*

The SignResponseType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-SignResponseType" : {
  "type" : "object",
  "properties" : {
    "result" : {
      "$ref" : "#/definitions/dsb-ResultType"
    },
    "profile" : {
      "type" : "array",
      "items" : {
        "type" : "string"
      }
    },
    "reqID" : {
      "type" : "string"
    },
    "respID" : {
      "type" : "string"
    },
    "optOutp" : {
      "$ref" : "#/definitions/dss2-OptionalOutputsSignType"
    },
    "sigObj" : {
      "$ref" : "#/definitions/dss2-SignatureObjectType"
    }
  }
}
```

[component SignResponse JSON schema details]

SignResponse – XML Syntax

The XML type SignResponseType SHALL implement the requirements defined in the SignResponsecomponent.

The SignResponseType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="SignResponseType">
  <xs:complexContent>
    <xs:extension base="dsb:ResponseBaseType">
      <xs:sequence>
        <xs:element minOccurs="0" name="OptionalOutputs" type="dss2:OptionalOutputsSignTy
        <xs:element minOccurs="0" name="SignatureObject" type="dss2:SignatureObjectType"/
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Each child element of SignResponseType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component SignResponse XML schema details]*

Component VerifyRequest

The VerifyRequest component is sent by the client to verify a signature or timestamp on some input documents.

Below follows a list of the sub-components that MAY be present within this component:

The optional InputDocuments element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section InputDocumentsType. *[sub component InputDocuments details]*

The optional OptionalInputs element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section OptionalInputsVerifyType. *[sub component OptionalInputs details]*

The optional SignatureObject element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section SignatureObjectType. *The SignatureObject element contains a signature or timestamp, or else contains a <SignaturePtr> that points to an XML signature in one of the input documents.*

Non-normative Comment:

[component VerifyRequest non normative details]

VerifyRequest – JSON Syntax

The VerifyRequestType JSON object SHALL implement in JSON syntax the requirements defined in the VerifyRequest component.

Properties of the JSON object SHALL implement the sub-components of VerifyRequestType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
InputDocuments	inDocs	*[]*
OptionalInputs	optInp	*[]*
SignatureObject	sigObj	*[]*

The VerifyRequestType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-VerifyRequestType" : {
  "type" : "object",
  "properties" : {
    "profile" : {
      "type" : "array",
      "items" : {
        "type" : "string"
      }
    },
    "reqID" : {
      "type" : "string"
    },
    "inDocs" : {
      "$ref" : "#/definitions/dss2-InputDocumentsType"
    },
    "optInp" : {
      "$ref" : "#/definitions/dss2-OptionalInputsVerifyType"
    },
    "sigObj" : {
      "$ref" : "#/definitions/dss2-SignatureObjectType"
    }
  }
}
```

[component VerifyRequest JSON schema details]

VerifyRequest – XML Syntax

The XML type VerifyRequestType SHALL implement the requirements defined in the VerifyRequestcomponent.

The VerifyRequestType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="VerifyRequestType">
  <xs:complexContent>
    <xs:extension base="dsb:RequestBaseType">
      <xs:sequence>
        <xs:element minOccurs="0" name="InputDocuments" type="dss2:InputDocumentsType"/>
        <xs:element minOccurs="0" name="OptionalInputs" type="dss2:OptionalInputsVerifyTy
        <xs:element minOccurs="0" name="SignatureObject" type="dss2:SignatureObjectType"/
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Each child element of VerifyRequestType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component VerifyRequest XML schema details]*

Component VerifyResponse

The VerifyResponse component is returned by the server to provide the results of verification.

Below follows a list of the sub-components that MAY be present within this component:

The optional OptionalOutputs element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section OptionalOutputsVerifyType. *[sub component OptionalOutputs details]*

Non-normative Comment:

[component VerifyResponse non normative details]

VerifyResponse – JSON Syntax

The VerifyResponseType JSON object SHALL implement in JSON syntax the requirements defined in the VerifyResponse component.

Properties of the JSON object SHALL implement the sub-components of VerifyResponseType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
OptionalOutputs	optOutp	*[]*

The VerifyResponseType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-VerifyResponseType" : {
  "type" : "object",
  "properties" : {
    "result" : {
      "$ref" : "#/definitions/dsb-ResultType"
    },
    "profile" : {
      "type" : "array",
      "items" : {
        "type" : "string"
      }
    },
    "reqID" : {
      "type" : "string"
    },
    "respID" : {
      "type" : "string"
    },
    "optOutp" : {
      "$ref" : "#/definitions/dss2-OptionalOutputsVerifyType"
    }
  }
}
```

[component VerifyResponse JSON schema details]

VerifyResponse – XML Syntax

The XML type VerifyResponseType SHALL implement the requirements defined in the VerifyResponsecomponent.

The VerifyResponseType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="VerifyResponseType">
  <xs:complexContent>
    <xs:extension base="dsb:ResponseBaseType">
      <xs:sequence>
        <xs:element minOccurs="0" name="OptionalOutputs" type="dss2:OptionalOutputsVerify
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Each child element of VerifyResponseType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component VerifyResponse XML schema details]*

Component PendingRequest

The PendingRequest component is sent by the client to retrieve the result of a previous request. The client MUST provide the ResponseID received with the initial response. The Profile element MUST NOT be present as the profile selection was done with the initial request.

Below follows a list of the sub-components that MAY be present within this component:

The optional ClaimedIdentity element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section ClaimedIdentityType. *This element allows the authentication of the requestor.*

Non-normative Comment:

[component PendingRequest non normative details]

PendingRequest – JSON Syntax

The PendingRequestType JSON object SHALL implement in JSON syntax the requirements defined in the PendingRequest component.

Properties of the JSON object SHALL implement the sub-components of PendingRequestType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
ClaimedIdentity	claimedIdentity	*[]*

The PendingRequestType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-PendingRequestType" : {
  "type" : "object",
  "properties" : {
    "profile" : {
      "type" : "array",
      "items" : {
        "type" : "string"
      }
    },
    "reqID" : {
      "type" : "string"
    },
    "claimedIdentity" : {
      "$ref" : "#/definitions/dss2-ClaimedIdentityType"
    }
  }
}
```

[component PendingRequest JSON schema details]

PendingRequest – XML Syntax

The XML type PendingRequestType SHALL implement the requirements defined in the PendingRequestcomponent.

The PendingRequestType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="PendingRequestType">
  <xs:complexContent>
    <xs:extension base="dsb:RequestBaseType">
      <xs:sequence>
        <xs:element maxOccurs="1" minOccurs="0" name="ClaimedIdentity" type="dss2:Claimed
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Each child element of PendingRequestType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component PendingRequest XML schema details]*

Optional data structures defined in this document

The XML elements of this section are defined in the XML namespace 'http://docs.oasis-open.org/dss/ns/core'.*[category optionals in namespace http://docs.oasis-open.org/dss/ns/core explanation]*

Component RequestID

[component RequestID normative details]

Below follows a list of the sub-components that MAY be present within this component:

The value element MUST contain one instance of a string. *[sub component value details]*

Non-normative Comment:

[component RequestID non normative details]

RequestID – JSON Syntax

The component RequestIDis derived from the string type.

[component RequestID JSON schema details]

RequestID – XML Syntax

The XML type RequestID SHALL implement the requirements defined in the RequestIDcomponent.

The RequestID XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:simpleType name="RequestID">
  <xs:restriction base="xs:string"/>
</xs:simpleType>
```

Each child element of RequestID XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component RequestID XML schema details]*

Component ResponseID

[component ResponseID normative details]

Below follows a list of the sub-components that MAY be present within this component:

The value element MUST contain one instance of a string. *[sub component value details]*

Non-normative Comment:

[component ResponseID non normative details]

ResponseID – JSON Syntax

The component ResponseIDis derived from the string type.

[component ResponseID JSON schema details]

ResponseID – XML Syntax

The XML type ResponseID SHALL implement the requirements defined in the ResponseIDcomponent.

The ResponseID XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:simpleType name="ResponseID">
  <xs:restriction base="xs:string"/>
</xs:simpleType>
```

Each child element of ResponseID XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component ResponseID XML schema details]*

Component OptionalInputsBase

The OptionalInputsBase contains a common set of additional inputs associated with the processing of the request. Profiles will specify the allowed optional inputs and their default values. If a server doesn't recognize or can't handle any optional input, it MUST reject the request with a ResultMajor code of RequesterError and a ResultMinor code of NotSupported.All request messages can contain an OptionalInputSign or OptionalInputVerify element depending on the method called. The OptionalInputsBase component defines the elements that are common to all optional inputs defined in this document. Several optional inputs are defined in this document, and profiles can define additional ones.

Below follows a list of the sub-components that MAY be present within this component:

The optional ClaimedIdentity element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section ClaimedIdentityType. *This element indicates the identity of the client who is making a request. The server may use this to parameterize any aspect of its processing. Profiles that make use of this element MUST define its semantics.*

The optional Schemas element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section SchemasType. *The Schemas element provides a mechanism for transporting XML schemas required for validating an XML document along with the request message.*

The optional AddTimestamp element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section AugmentSignatureInstructionType. *The AddTimestamp element indicates that the client wishes the server to embed a timestamp token as a property or attribute of the resultant or the supplied signature. The timestamp token will be applied to the signature value in the case of CMS/PKCS7 signatures or the <ds:SignatureValue> element in the case of XML signatures. Note: Procedures for handling other forms of timestamp may be defined in profiles of the Core. In particular, the DSS AdES profile [DSS-AdES-P] defines procedures for generating timestamps over the content which is about to be signed (sometimes called content timestamps), and the DSS Timestamp profile [DSS-TS-P] defines procedures for handling standalone timestamps.*

Non-normative Comment:

[component OptionalInputsBase non normative details]

OptionalInputsBase – JSON Syntax

The component OptionalInputsBaseis abstract and therefore has no JSON definition.

[component OptionalInputsBase JSON schema details]

OptionalInputsBase – XML Syntax

The XML type OptionalInputsBaseType SHALL implement the requirements defined in the OptionalInputsBasecomponent.

The OptionalInputsBaseType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType abstract="true" name="OptionalInputsBaseType">
  <xs:complexContent>
    <xs:extension base="dsb:OptionalInputsType">
      <xs:sequence>
        <xs:element maxOccurs="1" minOccurs="0" name="ClaimedIdentity" type="dss2:Claimed
        <xs:element maxOccurs="1" minOccurs="0" name="Schemas" type="dss2:SchemasType"/>
        <xs:element maxOccurs="1" minOccurs="0" name="AddTimestamp" type="dss2:AugmentSig
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Each child element of OptionalInputsBaseType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component OptionalInputsBase XML schema details]*

Component OptionalInputsSign

The OptionalInputsSign component defines a set of additional inputs associated with the processing of a signing request.The OptionalInputsSign component contains additional inputs associated with the processing of a signing request. Profiles MAY specify the allowed optional inputs and their default values. The definition of an optional input MAY include a default value, so that a client may omit the OptionalInputsSign yet still get service from any profile-compliant DSS server. If a server doesn't recognize or can't handle any optional input, it MUST reject the request with a ResultMajor code of RequesterError and a ResultMinor code of NotSupported.

Below follows a list of the sub-components that MAY be present within this component:

The optional SignatureType element MUST contain a URI. *The SignatureType element indicates the type of signature or timestamp to produce (such as a XML signature, a XML timestamp, a RFC 3161 timestamp, a CMS signature, etc.). See section 7.1 for some URI references that MAY be used as the value of this element.*

The optional IntendedAudience element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section IntendedAudienceType. *This element gives a hint regarding the target audience of the requested signature.*

The optional KeySelector element MAY occur zero or more times containing a sub-component. If present each instance MUST satisfy the requirements specified in this document in section

KeySelectorType. *The KeySelector provides details which key or sets of keys the client is expecting to be used.*

The optional Properties element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section PropertiesHolderType. *The Properties element is used to instruct the server to add certain signed or unsigned properties (aka "signature attributes") into the signature. The client MAY send the server a particular value to use for each property, or leave the value up to the server to determine. The server MAY add additional properties, even if these aren't requested by the client.*

The optional IncludeObject element MAY occur zero or more times containing a sub-component. If present each instance MUST satisfy the requirements specified in this document in section IncludeObjectType. *The IncludeObject element is used to request the creation of an XMLSig enveloping signature.*

The optional IncludeEContent element MUST contain a boolean. Its default value is 'false'.
If the value of the IncludeEContent is 'true' a CMS signature includes enveloped (or 'encapsulated') content.

The optional SignaturePlacement element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section SignaturePlacementType. *The SignaturePlacement element is used to request the creation of an XMLSig enveloped signature placed within a document. The resulting document with the enveloped signature is placed in the optional output DocumentWithSignature.*

The optional SignedReferences element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section SignedReferencesType. *The SignedReferences element gives the client greater control over how the <ds:Reference> elements of a XMLSig signature are formed.*

The optional Nonce element MUST contain an integer. *The Nonce element MAY be used to provide a large random number to enable the client correlate a timestamp request with the response.*

The optional SignatureAlgorithm element MUST contain a string. *The SignatureAlgorithm element MAY be used to request a specific signing algorithm. This may be useful to narrow down the set of algorithms the server may apply. Support for specific signature algorithms may change over time and the use of other input elements, especially Profile and ServicePolicy. The use of the SignatureAlgorithm value is context specific, maybe different when requesting a CMS or XML signature.*

The optional SignatureQualityLevel element MUST contain a URI. *Legal and regulatory frameworks distinguish signatures by their level of quality, where a higher level of quality usually implies stronger restrictions on holder identification, protection of private key and certification of signature creation device and software. A server MAY be able to generate signatures of different quality levels. This element allows the requester to define a minimum signature quality level. Values for this URI may be specified by profiles.*

Non-normative Comment:

[component OptionalInputsSign non normative details]

OptionalInputsSign – JSON Syntax

The OptionalInputsSignType JSON object SHALL implement in JSON syntax the requirements defined in the OptionalInputsSign component.

Properties of the JSON object SHALL implement the sub-components of OptionalInputsSignType using JSON-specific names mapped as shown in the table below.

Element	**Implementing JSON member name**	**Comments**
SignatureType	sigType	*[]*
IntendedAudience	aud	*[]*
KeySelector	keySel	*[]*
Properties	props	*[]*
IncludeObject	incObj	*[]*
IncludeEContent	incContent	*[]*
SignaturePlacement	sigPlacement	*[]*
SignedReferences	signedRefs	*[]*
Nonce	nonce	*[]*
SignatureAlgorithm	sigAlgo	*[]*
SignatureQualityLevel	quality	*[]*

The OptionalInputsSignType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-OptionalInputsSignType" : {
  "type" : "object",
  "properties" : {
    "policy" : {
      "type" : "array",
      "items" : {
        "type" : "string"
      }
    },
    "lang" : {
      "type" : "string"
    },
    "other" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dsb-AnyType"
      }
    },
    "claimedIdentity" : {
      "$ref" : "#/definitions/dss2-ClaimedIdentityType"
    },
    "schemas" : {
      "$ref" : "#/definitions/dss2-SchemasType"
    },
    "addTimestamp" : {
      "$ref" : "#/definitions/dss2-AugmentSignatureInstructionType"
    },
    "sigType" : {
      "type" : "string"
    },
    "aud" : {
      "$ref" : "#/definitions/dss2-IntendedAudienceType"
    },
    "keySel" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dss2-KeySelectorType"
      }
    },
    "props" : {
      "$ref" : "#/definitions/dss2-PropertiesHolderType"
    },
    "incObj" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dss2-IncludeObjectType"
      }
    },
    "incContent" : {
      "type" : "boolean",
      "default" : "false"
    },
    "sigPlacement" : {
      "$ref" : "#/definitions/dss2-SignaturePlacementType"
    },
    "signedRefs" : {
      "$ref" : "#/definitions/dss2-SignedReferencesType"
    },
    "nonce" : {
      "type" : "integer"
    },
    "sigAlgo" : {
      "type" : "string"
    },
    "quality" : {
      "type" : "string"
    }
  }
}
```

[component OptionalInputsSign JSON schema details]

OptionalInputsSign – XML Syntax

The XML type OptionalInputsSignType SHALL implement the requirements defined in the OptionalInputsSigncomponent.

The OptionalInputsSignType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="OptionalInputsSignType">
  <xs:complexContent>
    <xs:extension base="dss2:OptionalInputsBaseType">
      <xs:sequence>
        <xs:element maxOccurs="1" minOccurs="0" name="SignatureType" type="xs:anyURI"/>
        <xs:element maxOccurs="1" minOccurs="0" name="IntendedAudience" type="dss2:Intend
        <xs:element maxOccurs="unbounded" minOccurs="0" name="KeySelector" type="dss2:Key
        <xs:element maxOccurs="1" minOccurs="0" name="Properties" type="dss2:PropertiesHo
        <xs:element maxOccurs="unbounded" minOccurs="0" name="IncludeObject" type="dss2:I
        <xs:element default="false" maxOccurs="1" minOccurs="0" name="IncludeEContent" ty
        <xs:element maxOccurs="1" minOccurs="0" name="SignaturePlacement" type="dss2:Sign
        <xs:element maxOccurs="1" minOccurs="0" name="SignedReferences" type="dss2:Signed
        <xs:element maxOccurs="1" minOccurs="0" name="Nonce" type="xs:integer"/>
        <xs:element maxOccurs="1" minOccurs="0" name="SignatureAlgorithm" type="xs:string
        <xs:element maxOccurs="1" minOccurs="0" name="SignatureQualityLevel" type="xs:any
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Each child element of OptionalInputsSignType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component OptionalInputsSign XML schema details]*

Component OptionalInputsVerify

The OptionalInputsVerify component defines a set of additional inputs associated with the processing of a verification request. Profiles MAY specify the allowed optional inputs and their default values. The definition of an optional input MAY include a default value, so that a client may omit the OptionalInputsVerify yet still get service from any profile-compliant DSS server.If a server doesn't recognize or can't handle any optional input, it MUST reject the request with a ResultMajor code of RequesterError and a ResultMinor code of NotSupported.

Below follows a list of the sub-components that MAY be present within this component:

The optional UseVerificationTime element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section UseVerificationTimeType. *The UseVerificationTime element instructs the server to attempt to determine the signature's validity at the specified time, instead of a time determined by the server policy.*

The optional ReturnVerificationTimeInfo element MUST contain a boolean. Its default value is 'false'. *This element cam be used by the client to obtain the time instant used by the server to validate the signature.*

The optional AdditionalKeyInfo element MAY occur zero or more times containing a sub-component. If present each instance MUST satisfy the requirements specified in this document in section AdditionalKeyInfoType. *This element provides the server with additional data (such as certificates and CRLs) which it can use to validate the signature. These options are not allowed in multi-signature verification.*

The optional ReturnProcessingDetails element MUST contain a boolean. Its default value is 'false'. *This element instructs the server to return a ProcessingDetails element. It is not allowed in multi-signature verification.*

The optional ReturnSigningTimeInfo element MUST contain a boolean. Its default value is 'false'. *This element allows the client to instruct the server to return the time instant associated to the signature creation as a SigningTimeInfo element.*

The optional ReturnSignerIdentity element MUST contain a boolean. Its default value is 'false'.

The optional ReturnAugmentedSignature element MAY occur zero or more times containing a sub-component. If present each instance MUST satisfy the requirements specified in this document in section AugmentSignatureInstructionType. *This element allows the client to instruct the server to return an AugmentedSignature output, containing an augmented signature. This document does not define values for this element, but profiles may provide a set of URIs.*

The optional ReturnTransformedDocument element MAY occur zero or more times containing a sub-component. If present each instance MUST satisfy the requirements specified in this document in section ReturnTransformedDocumentType. *The ReturnTransformedDocument element instructs the server to return an input document to which the XML signature transforms specified by a particular <ds:Reference> have been applied. The result of the transformations will be returned as a TransformedDocument element.*

The optional ReturnTimestampedSignature element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section AugmentSignatureInstructionType. *It indicates that the client wishes the server to update the signature after its verification by embedding a signature timestamp token as an unauthenticated attribute (see "unauthAttrs" in section 9.1 [RFC 3852]) or *unsigned* property (see section 6.2.5 "The UnsignedSignatureProperties element" and section 7.3 "The SignatureTimeStamp element" [XAdES]) of the supplied signature. The timestamp token will be on the signature value in the case of CMS/PKCS7signatures or the <ds:SignatureValue> element in the case of XML signatures.*

The optional VerifyManifests element MUST contain a boolean. Its default value is 'false'. *This element is allowed in multi-signature verification requests.*

Non-normative Comment:

[component OptionalInputsVerify non normative details]

OptionalInputsVerify – JSON Syntax

The OptionalInputsVerifyType JSON object SHALL implement in JSON syntax the requirements defined in the OptionalInputsVerify component.

Properties of the JSON object SHALL implement the sub-components of OptionalInputsVerifyType using JSON-specific names mapped as shown in the table below.

Element	**Implementing JSON member name**	**Comments**
UseVerificationTime	useVerificationTime	*[]*
ReturnVerificationTimeInfo	returnVerificationTime	*[]*
AdditionalKeyInfo	addKeyInfo	*[]*
ReturnProcessingDetails	returnProcDetails	*[]*
ReturnSigningTimeInfo	returnSigningTime	*[]*
ReturnSignerIdentity	returnSigner	*[]*
ReturnAugmentedSignature	returnAugmented	*[]*
ReturnTransformedDocument	returnTransformed	*[]*
ReturnTimestampedSignature	returnTimestamped	*[]*
VerifyManifests	verifyManifests	*[]*

The OptionalInputsVerifyType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-OptionalInputsVerifyType" : {
  "type" : "object",
  "properties" : {
    "policy" : {
      "type" : "array",
      "items" : {
        "type" : "string"
      }
    },
    "lang" : {
      "type" : "string"
    },
    "other" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dsb-AnyType"
      }
    },
    "claimedIdentity" : {
      "$ref" : "#/definitions/dss2-ClaimedIdentityType"
    },
    "schemas" : {
      "$ref" : "#/definitions/dss2-SchemasType"
    },
    "addTimestamp" : {
      "$ref" : "#/definitions/dss2-AugmentSignatureInstructionType"
    },
    "useVerificationTime" : {
      "$ref" : "#/definitions/dss2-UseVerificationTimeType"
    },
    "returnVerificationTime" : {
      "type" : "boolean",
      "default" : "false"
    },
    "addKeyInfo" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dss2-AdditionalKeyInfoType"
      }
    },
    "returnProcDetails" : {
      "type" : "boolean",
      "default" : "false"
    },
    "returnSigningTime" : {
      "type" : "boolean",
      "default" : "false"
    },
    "returnSigner" : {
      "type" : "boolean",
      "default" : "false"
    },
    "returnAugmented" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dss2-AugmentSignatureInstructionType"
      }
    },
    "returnTransformed" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dss2-ReturnTransformedDocumentType"
      }
    },
    "returnTimestamped" : {
      "$ref" : "#/definitions/dss2-AugmentSignatureInstructionType"
    },
    "verifyManifests" : {
      "type" : "boolean",
      "default" : "false"
    }
  }
}
```

[component OptionalInputsVerify JSON schema details]

OptionalInputsVerify – XML Syntax

The XML type OptionalInputsVerifyType SHALL implement the requirements defined in the OptionalInputsVerifycomponent.

The OptionalInputsVerifyType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="OptionalInputsVerifyType">
  <xs:complexContent>
    <xs:extension base="dss2:OptionalInputsBaseType">
      <xs:sequence>
        <xs:element maxOccurs="1" minOccurs="0" name="UseVerificationTime" type="dss2:Use
        <xs:element default="false" maxOccurs="1" minOccurs="0" name="ReturnVerification
        <xs:element maxOccurs="unbounded" minOccurs="0" name="AdditionalKeyInfo" type="ds
        <xs:element default="false" maxOccurs="1" minOccurs="0" name="ReturnProcessingDet
        <xs:element default="false" maxOccurs="1" minOccurs="0" name="ReturnSigningTimeIn
        <xs:element default="false" maxOccurs="1" minOccurs="0" name="ReturnSignerIdentit
        <xs:element maxOccurs="unbounded" minOccurs="0" name="ReturnAugmentedSignature" t
        <xs:element maxOccurs="unbounded" minOccurs="0" name="ReturnTransformedDocument"
        <xs:element maxOccurs="1" minOccurs="0" name="ReturnTimestampedSignature" type="d
        <xs:element default="false" maxOccurs="1" minOccurs="0" name="VerifyManifests" ty
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Each child element of OptionalInputsVerifyType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component OptionalInputsVerify XML schema details]*

Component OptionalOutputsBase

The OptionalOutputsBase contains a common set of additional outputs associated with the processing of the request. The client MAY request the server to respond with certain optional outputs by sending certain optional inputs. The server MAY also respond with outputs the client didn't request, depending on the server's profile and policy. If a server doesn't recognize or can't handle any optional input, it MUST reject the request with a ResultMajor code of RequesterError and a ResultMinor code of NotSupported.

Below follows a list of the sub-components that MAY be present within this component:

The optional TransformedDocument element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section TransformedDocumentType. *The TransformedDocument element contains a document corresponding to the specified <ds:Reference>, after all the transforms in the reference have been applied. In other words, the hash value of the returned document should equal the <ds:Reference> element's <ds:DigestValue>.*

The optional Schemas element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section SchemasType. *The Schemas element is typically used* *as an optional input in a VerifyRequest. However, there are situations where it may be used as an optional output. For example, a service that makes use of the ReturnUpdatedSignature mechanism may, after verifying a signature over an input document, generate a signature over a document of a different schema than the input document. In this case the Schemas element MAY be used to communicate the XML schemas required for validating a returned XML document.*

The optional DocumentWithSignature element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section [DocumentWithSignatureType](). *DocumentWithSignature element contains the input document with the signature inserted.*

Non-normative Comment:

[component OptionalOutputsBase non normative details]

OptionalOutputsBase – JSON Syntax

The component OptionalOutputsBaseis abstract and therefore has no JSON definition.

[component OptionalOutputsBase JSON schema details]

OptionalOutputsBase – XML Syntax

The XML type OptionalOutputsBaseType SHALL implement the requirements defined in the OptionalOutputsBasecomponent.

The OptionalOutputsBaseType XML element is defined in XML Schema [[DSS2XSD]()], and is copied below for information.

```
<xs:complexType abstract="true" name="OptionalOutputsBaseType">
  <xs:complexContent>
    <xs:extension base="dsb:OptionalOutputsType">
      <xs:sequence>
        <xs:element maxOccurs="1" minOccurs="0" name="TransformedDocument" type="dss2:Tra
        <xs:element maxOccurs="1" minOccurs="0" name="Schemas" type="dss2:SchemasType"/>
        <xs:element maxOccurs="1" minOccurs="0" name="DocumentWithSignature" type="dss2:I
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Each child element of OptionalOutputsBaseType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component OptionalOutputsBase XML schema details]*

Component OptionalOutputsSign

The OptionalOutputsSignType component defines a set of additional outputs associated with the processing of a signing request. This document does not define any additional outputs but profiles may extend the set of additional outputs.

Below follows a list of the sub-components that MAY be present within this component:

Non-normative Comment:

[component OptionalOutputsSign non normative details]

OptionalOutputsSign – JSON Syntax

The OptionalOutputsSignType JSON object SHALL implement in JSON syntax the requirements defined in the OptionalOutputsSign component.

The OptionalOutputsSignType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-OptionalOutputsSignType" : {
  "type" : "object",
  "properties" : {
    "policy" : {
      "type" : "array",
      "items" : {
        "type" : "string"
      }
    },
    "other" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dsb-AnyType"
      }
    },
    "transformed" : {
      "$ref" : "#/definitions/dss2-TransformedDocumentType"
    },
    "schemas" : {
      "$ref" : "#/definitions/dss2-SchemasType"
    },
    "docWithSignature" : {
      "$ref" : "#/definitions/dss2-DocumentWithSignatureType"
    }
  }
}
```

[component OptionalOutputsSign JSON schema details]

OptionalOutputsSign – XML Syntax

The XML type OptionalOutputsSignType SHALL implement the requirements defined in the OptionalOutputsSigncomponent.

The OptionalOutputsSignType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="OptionalOutputsSignType">
  <xs:complexContent>
    <xs:extension base="dss2:OptionalOutputsBaseType"/>
  </xs:complexContent>
</xs:complexType>
```

Each child element of OptionalOutputsSignType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component OptionalOutputsSign XML schema details]*

Component OptionalOutputsVerify

The OptionalOutputsVerify component defines a set of additional outputs associated with the processing of a verification request.

Below follows a list of the sub-components that MAY be present within this component:

The optional VerifyManifestResults element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section VerifyManifestResultsType. *[sub component VerifyManifestResults details]*

The optional SigningTimeInfo element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section SigningTimeInfoType. *The SigningTimeInfo element returns the signature's creation date and time. When there's no way for the server to determine the signing time, the server MUST omit this element.*

The optional VerificationTimeInfo element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section VerificationTimeInfoType. *In addition to the verification time, the server MAY include in the VerificationTimeInfo element any other relevant time instants that may have been used when determining the verification time or that may be useful for its qualification.*

The optional ProcessingDetails element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section ProcessingDetailsType. *The ProcessingDetails element elaborates on what signature verification steps succeeded or failed.*

The optional SignerIdentity element MUST contain a sub-component. A given element MUST satisfy the requirements specified in section NameIDType. *The SignerIdentity element contains an indication of who performed the signature.*

The optional AugmentedSignature element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section AugmentedSignatureType. *This element contains the processed signature.*

The optional TimestampedSignature element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section AugmentedSignatureType. *The TimestampedSignature element contains the returned timestamped signature.*

Non-normative Comment:

[component OptionalOutputsVerify non normative details]

OptionalOutputsVerify – JSON Syntax

The OptionalOutputsVerifyType JSON object SHALL implement in JSON syntax the requirements defined in the OptionalOutputsVerify component.

Properties of the JSON object SHALL implement the sub-components of OptionalOutputsVerifyType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
VerifyManifestResults	result	*[]*
SigningTimeInfo	signingTimeInfo	*[]*
VerificationTimeInfo	verificationTimeInfo	*[]*
ProcessingDetails	procDetails	*[]*
SignerIdentity	signerIdentity	*[]*
AugmentedSignature	augSig	*[]*
TimestampedSignature	timestampedSig	*[]*

The OptionalOutputsVerifyType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-OptionalOutputsVerifyType" : {
  "type" : "object",
  "properties" : {
    "policy" : {
      "type" : "array",
      "items" : {
        "type" : "string"
      }
    },
    "other" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dsb-AnyType"
      }
    },
    "transformed" : {
      "$ref" : "#/definitions/dss2-TransformedDocumentType"
    },
    "schemas" : {
      "$ref" : "#/definitions/dss2-SchemasType"
    },
    "docWithSignature" : {
      "$ref" : "#/definitions/dss2-DocumentWithSignatureType"
    },
    "result" : {
      "$ref" : "#/definitions/dss2-VerifyManifestResultsType"
    },
    "signingTimeInfo" : {
      "$ref" : "#/definitions/dss2-SigningTimeInfoType"
    },
    "verificationTimeInfo" : {
      "$ref" : "#/definitions/dss2-VerificationTimeInfoType"
    },
    "procDetails" : {
      "$ref" : "#/definitions/dss2-ProcessingDetailsType"
    },
    "signerIdentity" : {
      "$ref" : "#/definitions/saml2rw-NameIDType"
    },
    "augSig" : {
      "$ref" : "#/definitions/dss2-AugmentedSignatureType"
    },
    "timestampedSig" : {
      "$ref" : "#/definitions/dss2-AugmentedSignatureType"
    }
  }
}
```

[component OptionalOutputsVerify JSON schema details]

OptionalOutputsVerify – XML Syntax

The XML type OptionalOutputsVerifyType SHALL implement the requirements defined in the OptionalOutputsVerifycomponent.

The OptionalOutputsVerifyType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="OptionalOutputsVerifyType">
  <xs:complexContent>
    <xs:extension base="dss2:OptionalOutputsBaseType">
      <xs:sequence>
        <xs:element maxOccurs="1" minOccurs="0" name="VerifyManifestResults" type="dss2:V
        <xs:element maxOccurs="1" minOccurs="0" name="SigningTimeInfo" type="dss2:Signing
        <xs:element maxOccurs="1" minOccurs="0" name="VerificationTimeInfo" type="dss2:Ve
        <xs:element maxOccurs="1" minOccurs="0" name="ProcessingDetails" type="dss2:Proce
        <xs:element maxOccurs="1" minOccurs="0" name="SignerIdentity" type="saml2-rw:Name
        <xs:element maxOccurs="1" minOccurs="0" name="AugmentedSignature" type="dss2:Augm
        <xs:element maxOccurs="1" minOccurs="0" name="TimestampedSignature" type="dss2:Au
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Each child element of OptionalOutputsVerifyType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component OptionalOutputsVerify XML schema details]*

Component ClaimedIdentity

This element indicates the identity of the client who is making a request. The server may use this to parameterize any aspect of its processing. Profiles that make use of this element MUST define its semantics.

Below follows a list of the sub-components that MAY be present within this component:

The Name element MUST contain one instance of a sub-component. This element MUST satisfy the requirements specified in section NameIDType. *The claimed identity may be authenticated using the security binding, according to section 6, or using authentication data provided in the SupportingInfo element. The server MUST check that the asserted Name is authenticated before relying upon the Name.*

The optional SupportingInfo element MUST contain a sub-component. A given element MUST satisfy the requirements specified in section AnyType. *The SupportingInfo element can be used by profiles to carry information related to the claimed identity. One possible use of SupportingInfo is to carry authentication data that authenticates the request as originating from the claimed identity (examples of authentication data include a password or SAML Assertion, a signature or MAC calculated over the request using a client key).*

Non-normative Comment:

[component ClaimedIdentity non normative details]

ClaimedIdentity – JSON Syntax

The ClaimedIdentityType JSON object SHALL implement in JSON syntax the requirements defined in the ClaimedIdentity component.

Properties of the JSON object SHALL implement the sub-components of ClaimedIdentityType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
Name	name	*[]*
SupportingInfo	suppInfo	*[]*

The ClaimedIdentityType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-ClaimedIdentityType" : {
  "type" : "object",
  "properties" : {
    "name" : {
      "$ref" : "#/definitions/saml2rw-NameIDType"
    },
    "suppInfo" : {
      "$ref" : "#/definitions/dsb-AnyType"
    }
  },
  "required" : ["name"]
}
```

[component ClaimedIdentity JSON schema details]

ClaimedIdentity – XML Syntax

The XML type ClaimedIdentityType SHALL implement the requirements defined in the ClaimedIdentitycomponent.

The ClaimedIdentityType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="ClaimedIdentityType">
  <xs:sequence>
    <xs:element name="Name" type="saml2-rw:NameIDType"/>
    <xs:element minOccurs="0" name="SupportingInfo" type="dsb:AnyType"/>
  </xs:sequence>
</xs:complexType>
```

Each child element of ClaimedIdentityType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component ClaimedIdentity XML schema details]*

Component Schemas

The Schemas component provides an in band mechanism for communicating XML schemas required for validating an XML document.

Below follows a list of the sub-components that MAY be present within this component:

The Schema element MUST occur 1 or more times containing a sub-component. Each instance MUST satisfy the requirements specified in this document in section DocumentType. *[sub component Schema details]*

Non-normative Comment:

Note: It is recommended to use xml:id as defined in [xml:id] as id in the payload being referenced by a <ds:Reference>, because the schema then does not have to be supplied for identifying the Id elements.

Schemas – JSON Syntax

The SchemasType JSON object SHALL implement in JSON syntax the requirements defined in the Schemas component.

Properties of the JSON object SHALL implement the sub-components of SchemasType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
Schema	schema	*[]*

The SchemasType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-SchemasType" : {
  "type" : "object",
  "properties" : {
    "schema" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dss2-DocumentType"
      }
    }
  },
  "required" : ["schema"]
}
```

[component Schemas JSON schema details]

Schemas – XML Syntax

The XML type SchemasType SHALL implement the requirements defined in the Schemascomponent.

The SchemasType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="SchemasType">
  <xs:sequence>
    <xs:element maxOccurs="unbounded" name="Schema" type="dss2:DocumentType"/>
  </xs:sequence>
</xs:complexType>
```

Each child element of SchemasType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component Schemas XML schema details]*

Component AugmentSignatureInstruction

The AugmentSignatureInstruction component can be used as an optional input for both signing and verification requests and defines the type of augmentation that should to be applied. The augmented signature will be returned in an AugmentedSignature (see section 4.3.32) or a DocumentWithSignature (see section 4.3.20) component. ESI defines the term 'augmentation' as follows in ETSI TR 119 001 [ESIFrame]: "signature augmentation: process of incorporating to a digital signature information aiming to maintain the validity of that signature over the long term"

Below follows a list of the sub-components that MAY be present within this component:

The optional Type element MUST contain one instance of a URI. *[sub component Type details]*

Non-normative Comment:

[component AugmentSignatureInstruction non normative details]

AugmentSignatureInstruction – JSON Syntax

The AugmentSignatureInstructionType JSON object SHALL implement in JSON syntax the requirements defined in the AugmentSignatureInstruction component.

Properties of the JSON object SHALL implement the sub-components of AugmentSignatureInstructionType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
Type	type	*[]*

The AugmentSignatureInstructionType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-AugmentSignatureInstructionType" : {
  "type" : "object",
  "properties" : {
    "type" : {
      "type" : "string",
      "format" : "uri"
    }
  }
}
```

[component AugmentSignatureInstruction JSON schema details]

AugmentSignatureInstruction – XML Syntax

The XML type AugmentSignatureInstructionType SHALL implement the requirements defined in the AugmentSignatureInstructioncomponent.

The AugmentSignatureInstructionType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="AugmentSignatureInstructionType">
  <xs:attribute name="Type" type="xs:anyURI" use="optional"/>
</xs:complexType>
```

Each child element of AugmentSignatureInstructionType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component AugmentSignatureInstruction XML schema details]*

Component IntendedAudience

The IntendedAudience element tells the server who the target audience of this signature is. The server MAY use this to parameterize any aspect of its processing (for example, the server MAY choose to sign with a key that it knows a particular recipient trusts).

Below follows a list of the sub-components that MAY be present within this component:

The Recipient element MUST occur 1 or more times containing a sub-component. Each instance MUST satisfy the requirements specified in section NameIDType. *[sub component Recipient details]*

Non-normative Comment:

[component IntendedAudience non normative details]

IntendedAudience – JSON Syntax

The IntendedAudienceType JSON object SHALL implement in JSON syntax the requirements defined in the IntendedAudience component.

Properties of the JSON object SHALL implement the sub-components of IntendedAudienceType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
Recipient	recipient	*[]*

The IntendedAudienceType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-IntendedAudienceType" : {
  "type" : "object",
  "properties" : {
    "recipient" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/saml2rw-NameIDType"
      }
    }
  },
  "required" : ["recipient"]
}
```

[component IntendedAudience JSON schema details]

IntendedAudience – XML Syntax

The XML type IntendedAudienceType SHALL implement the requirements defined in the IntendedAudiencecomponent.

The IntendedAudienceType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="IntendedAudienceType">
  <xs:sequence>
    <xs:element maxOccurs="unbounded" name="Recipient" type="saml2-rw:NameIDType"/>
  </xs:sequence>
</xs:complexType>
```

Each child element of IntendedAudienceType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component IntendedAudience XML schema details]*

Component KeySelector

The KeySelector component holds data that selects a specific key or certificate or group of certificates. Only one of its sub-components MUST be present. But a KeySelector component can occur multiple times as a sub-component in the OptionalInputsSign component

Below follows a list of the sub-components that MAY be present within this component:

The optional X509Digest element MUST contain one instance of a sub-component. This element MUST satisfy the requirements specified in this document in section X509DigestType. *[sub component X509Digest details]*

The optional X509SubjectName element MUST contain one instance of a string. *The X509SubjectName element contains an X.509 subject distinguished name that SHOULD be represented as a string that complies with section 3 of RFC4514 [LDAP-DN].*

The optional X509SKI element MUST contain one instance of base64 encoded binary data. *The X509SKI element contains the base64 encoded plain (i.e. non-DER-encoded) value of a X509 V.3 SubjectKeyIdentifier extension.*

The optional X509Certificate element MUST contain one instance of base64 encoded binary data. *The X509Certificate element contains a base64-encoded [X509V3] certificate.*

The optional KeyName element MUST contain one instance of a string. *It selects a key to be used for signing in a generic way. Usually the client knows about the valid values for KeyName.*

Non-normative Comment:

[component KeySelector non normative details]

KeySelector – JSON Syntax

The KeySelectorType JSON object SHALL implement in JSON syntax the requirements defined in the KeySelector component.

Properties of the JSON object SHALL implement the sub-components of KeySelectorType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
X509Digest	x509Digest	*[]*
X509SubjectName	sub	*[]*
X509SKI	ski	*[]*
X509Certificate	cert	*[]*
KeyName	name	*[]*

The KeySelectorType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-KeySelectorType" : {
  "type" : "object",
  "properties" : {
    "x509Digest" : {
      "$ref" : "#/definitions/dss2-X509DigestType"
    },
    "sub" : {
      "type" : "string"
    },
    "ski" : {
      "type" : "string"
    },
    "cert" : {
      "type" : "string"
    },
    "name" : {
      "type" : "string"
    }
  },
  "minProperties" : 1,
  "maxProperties" : 1
}
```

[component KeySelector JSON schema details]

KeySelector – XML Syntax

The XML type KeySelectorType SHALL implement the requirements defined in the KeySelectorcomponent.

The KeySelectorType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="KeySelectorType">
  <xs:choice>
    <xs:element name="X509Digest" type="dss2:X509DigestType"/>
    <xs:element name="X509SubjectName" type="xs:string"/>
    <xs:element name="X509SKI" type="xs:base64Binary"/>
    <xs:element name="X509Certificate" type="xs:base64Binary"/>
    <xs:element name="KeyName" type="xs:string"/>
  </xs:choice>
</xs:complexType>
```

Each child element of KeySelectorType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component KeySelector XML schema details]*

Component X509Digest

The X509Digest component contains a base64-encoded digest of a certificate. The digest algorithm URI is identified with a required Algorithm element. The input to the digest MUST be the raw octets that would be base64-encoded of a X509Certificate.

Below follows a list of the sub-components that MAY be present within this component:

The value element MUST contain one instance of base64 encoded binary data. *[sub component value details]*

The Algorithm element MUST contain one instance of a string. *The string describes the digest algorithm in an appropriate way for the server side processing. Depending on the signature format this may be an OID (e.g. '2.16.840.1.101.3.4.2.1'), an URI (e.g. 'http://www.w3.org/2001/04/xmlenc#sha256') or a descriptive string ('SHA-256').*

Non-normative Comment:

[component X509Digest non normative details]

X509Digest – JSON Syntax

The X509DigestType JSON object SHALL implement in JSON syntax the requirements defined in the X509Digest component.

Properties of the JSON object SHALL implement the sub-components of X509DigestType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
value	value	*[]*
Algorithm	alg	*[]*

The X509DigestType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-X509DigestType" : {
  "type" : "object",
  "properties" : {
    "value" : {
      "type" : "string"
    },
    "alg" : {
      "type" : "string"
    }
  },
  "required" : ["alg"]
}
```

[component X509Digest JSON schema details]

X509Digest – XML Syntax

The XML type X509DigestType SHALL implement the requirements defined in the X509Digestcomponent.

The X509DigestType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="X509DigestType">
  <xs:simpleContent>
    <xs:extension base="xs:base64Binary">
      <xs:attribute name="Algorithm" type="xs:string" use="required"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

Each child element of X509DigestType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component X509Digest XML schema details]*

Component PropertiesHolder

The PropertiesHolder component is used to request that the server add certain signed or unsigned properties (aka “signature attributes”) into the signature. The client can send the server a particular value to use for each property, or leave the value up to the server to determine. The server can add additional properties, even if these aren’t requested by the client.

Below follows a list of the sub-components that MAY be present within this component:

The optional SignedProperties element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section PropertiesType. *These properties will be covered by the signature.*

The optional UnsignedProperties element MUST contain a sub-component. A given element MUST satisfy the requirements specified in this document in section PropertiesType. *These properties will not be covered by the signature.*

Non-normative Comment:

[component PropertiesHolder non normative details]

PropertiesHolder – JSON Syntax

The PropertiesHolderType JSON object SHALL implement in JSON syntax the requirements defined in the PropertiesHolder component.

Properties of the JSON object SHALL implement the sub-components of PropertiesHolderType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
SignedProperties	signedProps	*[]*
UnsignedProperties	unsignedProps	*[]*

The PropertiesHolderType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-PropertiesHolderType" : {
  "type" : "object",
  "properties" : {
    "signedProps" : {
      "$ref" : "#/definitions/dss2-PropertiesType"
    },
    "unsignedProps" : {
      "$ref" : "#/definitions/dss2-PropertiesType"
    }
  }
}
```

[component PropertiesHolder JSON schema details]

PropertiesHolder – XML Syntax

The XML type PropertiesHolderType SHALL implement the requirements defined in the PropertiesHoldercomponent.

The PropertiesHolderType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="PropertiesHolderType">
  <xs:sequence>
    <xs:element minOccurs="0" name="SignedProperties" type="dss2:PropertiesType"/>
    <xs:element minOccurs="0" name="UnsignedProperties" type="dss2:PropertiesType"/>
  </xs:sequence>
</xs:complexType>
```

Each child element of PropertiesHolderType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component PropertiesHolder XML schema details]*

Component Properties

[component Properties normative details]

Below follows a list of the sub-components that MAY be present within this component:

The Property element MUST occur 1 or more times containing a sub-component. Each instance MUST satisfy the requirements specified in this document in section PropertyType. *[sub component Property details]*

Non-normative Comment:

[component Properties non normative details]

Properties – JSON Syntax

The PropertiesType JSON object SHALL implement in JSON syntax the requirements defined in the Properties component.

Properties of the JSON object SHALL implement the sub-components of PropertiesType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
Property	prop	*[]*

The PropertiesType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-PropertiesType" : {
  "type" : "object",
  "properties" : {
    "prop" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dss2-PropertyType"
      }
    }
  },
  "required" : ["prop"]
}
```

[component Properties JSON schema details]

Properties – XML Syntax

The XML type PropertiesType SHALL implement the requirements defined in the Propertiescomponent.

The PropertiesType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="PropertiesType">
  <xs:sequence>
    <xs:element maxOccurs="unbounded" name="Property" type="dss2:PropertyType"/>
  </xs:sequence>
</xs:complexType>
```

Each child element of PropertiesType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component Properties XML schema details]*

Component Property

[component Property normative details]

Below follows a list of the sub-components that MAY be present within this component:

The Identifier element MUST contain one instance of a string.

The optional Value element MUST contain a sub-component. A given element MUST satisfy the requirements specified in section AnyType. *The Value element contains arbitrary content wrapped in an Fehler! Verweisquelle konnte nicht gefunden werden..*

Non-normative Comment:

[component Property non normative details]

Property – JSON Syntax

The PropertyType JSON object SHALL implement in JSON syntax the requirements defined in the Property component.

Properties of the JSON object SHALL implement the sub-components of PropertyType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
Identifier	id	*[]*
Value	val	*[]*

The PropertyType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-PropertyType" : {
  "type" : "object",
  "properties" : {
    "id" : {
      "type" : "string"
    },
    "val" : {
      "$ref" : "#/definitions/dsb-AnyType"
    }
  },
  "required" : ["id"]
}
```

[component Property JSON schema details]

Property – XML Syntax

The XML type PropertyType SHALL implement the requirements defined in the Propertycomponent.

The PropertyType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="PropertyType">
  <xs:sequence>
    <xs:element name="Identifier" type="xs:string"/>
    <xs:element minOccurs="0" name="Value" type="dsb:AnyType"/>
  </xs:sequence>
</xs:complexType>
```

Each child element of PropertyType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *. Therefore it occurs in the XML schema, too.*

Component IncludeObject

The IncludeObject component is used to request the creation of an XMLSig enveloping signature. Multiple occurrences of this optional input can be present in a single SignRequest message. Each occurrence will cause the inclusion of an object inside the signature being created.

Below follows a list of the sub-components that MAY be present within this component:

The optional WhichDocument element MUST contain one instance of a unique identifier reference. *This element identifies the input document which will be inserted into the returned signature.*

The optional HasObjectTagsAndAttributesSet element MUST contain one instance of a boolean. Its default value is 'false'.
[sub component HasObjectTagsAndAttributesSet details]

The optional ObjId element MUST contain one instance of a string. *It sets the Id attribute on the returned <ds:Object>.*

The optional createReference element MUST contain one instance of a boolean. Its default value is 'true'. *If the createReference element is set to false inhibits the creation of the <ds:Reference> associated to the RefURI element of the input document referred by the WhichDocument element, effectively allowing clients to include <ds:Object> elements not covered/protected by the signature being created.*

Non-normative Comment:

[component IncludeObject non normative details]

IncludeObject – JSON Syntax

The IncludeObjectType JSON object SHALL implement in JSON syntax the requirements defined in the IncludeObject component.

Properties of the JSON object SHALL implement the sub-components of IncludeObjectType using JSON-specific names mapped as shown in the table below.

Element	**Implementing JSON member name**	**Comments**
WhichDocument	whichDoc	*[]*
HasObjectTagsAndAttributesSet	hasObjectTagsAndAttributesSet	*[]*
ObjId	objId	*[]*
createReference	createRef	*[]*

The IncludeObjectType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-IncludeObjectType" : {
  "type" : "object",
  "properties" : {
    "whichDoc" : {
      "$ref" : "#/definitions/dss2-DocumentBaseType"
    },
    "hasObjectTagsAndAttributesSet" : {
      "type" : "boolean",
      "default" : "false"
    },
    "objId" : {
      "type" : "string"
    },
    "createRef" : {
      "type" : "boolean",
      "default" : "true"
    }
  }
}
```

[component IncludeObject JSON schema details]

IncludeObject – XML Syntax

The XML type IncludeObjectType SHALL implement the requirements defined in the IncludeObjectcomponent.

The IncludeObjectType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="IncludeObjectType">
  <xs:attribute name="WhichDocument" type="xs:IDREF"/>
  <xs:attribute default="false" name="HasObjectTagsAndAttributesSet" type="xs:boolean"/>
  <xs:attribute name="ObjId" type="xs:string" use="optional"/>
  <xs:attribute default="true" name="createReference" type="xs:boolean" use="optional"/>
</xs:complexType>
```

Each child element of IncludeObjectType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component IncludeObject XML schema details]*

Component SignaturePlacement

The SignaturePlacement component is used to request the creation of an XMLSig enveloped signature placed within an input document. The resulting document with the enveloped signature is placed in the optional output DocumentWithSignature element. The server places the signature in the document identified using the WhichDocument attribute. In the case of a non-XML input document then the server will return an error unless alternative procedures are defined by a profile or in the server policy for handling such a situation.

Below follows a list of the sub-components that MAY be present within this component:

The optional XPathAfter element MUST contain one instance of a string. *This elements holds an XPath expression which identifies an element, inside the XML input document, after which the signature will be inserted.*

The optional XPathFirstChildOf element MUST contain one instance of a string. *This elements holds an XPath expression which identifies an element, in the XML input document, which the signature will be inserted as the first child of.*

The optional NsPrefixMapping element MAY occur zero or more times containing a sub-component. If present each instance MUST satisfy the requirements specified in section NsPrefixMappingType. *[sub component NsPrefixMapping details]*

The optional WhichDocument element MUST contain one instance of a unique identifier reference. *The WhichDocument element identifies the input document which the signature will be inserted into.*

The optional CreateEnvelopedSignature element MUST contain one instance of a boolean. Its default value is 'true'.
If the CreateEnvelopedSignature element is set to true a reference having an enveloped signature transform is created.

Non-normative Comment:

[component SignaturePlacement non normative details]

SignaturePlacement – JSON Syntax

The SignaturePlacementType JSON object SHALL implement in JSON syntax the requirements defined in the SignaturePlacement component.

Properties of the JSON object SHALL implement the sub-components of SignaturePlacementType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
XPathAfter	xPathAfter	*[]*
XPathFirstChildOf	xPathFirstChildOf	*[]*
NsPrefixMapping	nsDecl	*[]*
WhichDocument	whichDoc	*[]*
CreateEnvelopedSignature	createEnvelopedSignature	*[]*

The SignaturePlacementType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-SignaturePlacementType" : {
  "type" : "object",
  "properties" : {
    "xpathAfter" : {
      "type" : "string"
    },
    "xpathFirstChildOf" : {
      "type" : "string"
    },
    "xPathAfter" : {
      "type" : "string"
    },
    "xPathFirstChildOf" : {
      "type" : "string"
    },
    "nsDecl" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dsb-NsPrefixMappingType"
      }
    },
    "whichDoc" : {
      "$ref" : "#/definitions/dss2-DocumentBaseType"
    },
    "createEnvelopedSignature" : {
      "type" : "boolean",
      "default" : "true"
    }
  }
}
```

[component SignaturePlacement JSON schema details]

SignaturePlacement – XML Syntax

The XML type SignaturePlacementType SHALL implement the requirements defined in the SignaturePlacementcomponent.

The SignaturePlacementType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="SignaturePlacementType">
  <xs:sequence>
    <xs:choice>
      <xs:element name="XPathAfter" type="xs:string"/>
      <xs:element name="XPathFirstChildOf" type="xs:string"/>
    </xs:choice>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="NsPrefixMapping" type="dsb:NsP
  </xs:sequence>
  <xs:attribute name="WhichDocument" type="xs:IDREF"/>
  <xs:attribute default="true" name="CreateEnvelopedSignature" type="xs:boolean"/>
</xs:complexType>
```

Each child element of SignaturePlacementType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component SignaturePlacement XML schema details]*

Component DocumentWithSignature

The DocumentWithSignature component contains a 3.1.14 with the signature inserted as requested with the SignaturePlacement component.

Below follows a list of the sub-components that MAY be present within this component:

The Document element MUST contain one instance of a sub-component. This element MUST satisfy the requirements specified in this document in section DocumentType. *This contains the input document with a signature inserted in some fashion.*

Non-normative Comment:

[component DocumentWithSignature non normative details]

DocumentWithSignature – JSON Syntax

The DocumentWithSignatureType JSON object SHALL implement in JSON syntax the requirements defined in the DocumentWithSignature component.

Properties of the JSON object SHALL implement the sub-components of DocumentWithSignatureType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
Document	doc	*[]*

The DocumentWithSignatureType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-DocumentWithSignatureType" : {
  "type" : "object",
  "properties" : {
    "doc" : {
      "$ref" : "#/definitions/dss2-DocumentType"
    }
  },
  "required" : ["doc"]
}
```

[component DocumentWithSignature JSON schema details]

DocumentWithSignature – XML Syntax

The XML type DocumentWithSignatureType SHALL implement the requirements defined in the DocumentWithSignaturecomponent.

The DocumentWithSignatureType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="DocumentWithSignatureType">
  <xs:sequence>
    <xs:element name="Document" type="dss2:DocumentType"/>
  </xs:sequence>
</xs:complexType>
```

Each child element of DocumentWithSignatureType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component DocumentWithSignature XML schema details]*

Component SignedReferences

The SignedReferences component gives the client greater control over how the <ds:Reference> elements are formed.

Below follows a list of the sub-components that MAY be present within this component:

The SignedReference element MUST occur 1 or more times containing a sub-component. Each instance MUST satisfy the requirements specified in this document in section SignedReferenceType. *[sub component SignedReference details]*

Non-normative Comment:

[component SignedReferences non normative details]

SignedReferences – JSON Syntax

The SignedReferencesType JSON object SHALL implement in JSON syntax the requirements defined in the SignedReferences component.

Properties of the JSON object SHALL implement the sub-components of SignedReferencesType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
SignedReference	signedRef	*[]*

The SignedReferencesType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-SignedReferencesType" : {
  "type" : "object",
  "properties" : {
    "signedRef" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dss2-SignedReferenceType"
      }
    }
  },
  "required" : ["signedRef"]
}
```

[component SignedReferences JSON schema details]

SignedReferences – XML Syntax

The XML type SignedReferencesType SHALL implement the requirements defined in the SignedReferencescomponent.

The SignedReferencesType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="SignedReferencesType">
  <xs:sequence>
    <xs:element maxOccurs="unbounded" name="SignedReference" type="dss2:SignedReferenceTy
  </xs:sequence>
</xs:complexType>
```

Each child element of SignedReferencesType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component SignedReferences XML schema details]*

Component SignedReference

Each SignedReference component refers to an input document and allows multiple <ds:Reference> elements to be based on a single input document. Furthermore, the client can request additional transforms to be applied to each <ds:Reference>, and can set each <ds:Reference> element's Id or URI attribute. These aspects of the <ds:Reference> can only be set through the SignedReference component; they cannot be set through the input documents, since they are aspects of the reference to the input document, not the input document itself.

Below follows a list of the sub-components that MAY be present within this component:

The optional Transforms element MUST contain a sub-component. A given element MUST satisfy the requirements specified in section TransformsType. *The Transforms element requests the server to perform additional transforms on this reference.*

The WhichDocument element MUST contain one instance of a unique identifier reference. *This defines w hich input document this reference refers to.*

The optional RefURI element MUST contain one instance of a URI. *If this element is present, the corresponding <ds:Reference> element's URI attribute is set to its value. If it is not present, the URI attribute is omitted in the corresponding <ds:Reference>.*

The optional RefId element MUST contain one instance of a string. *This element sets the Id attribute of the corresponding <ds:Reference>.*

Non-normative Comment:

[component SignedReference non normative details]

SignedReference – JSON Syntax

The SignedReferenceType JSON object SHALL implement in JSON syntax the requirements defined in the SignedReference component.

Properties of the JSON object SHALL implement the sub-components of SignedReferenceType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
Transforms	transforms	*[]*
WhichDocument	whichDoc	*[]*
RefURI	refURI	*[]*
RefId	refId	*[]*

The SignedReferenceType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-SignedReferenceType" : {
  "type" : "object",
  "properties" : {
    "transforms" : {
      "$ref" : "#/definitions/dsigrw-TransformsType"
    },
    "whichDoc" : {
      "$ref" : "#/definitions/dss2-DocumentBaseType"
    },
    "refURI" : {
      "type" : "string"
    },
    "refId" : {
      "type" : "string"
    }
  },
  "required" : ["whichDoc"]
}
```

[component SignedReference JSON schema details]

SignedReference – XML Syntax

The XML type SignedReferenceType SHALL implement the requirements defined in the SignedReferencecomponent.

The SignedReferenceType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="SignedReferenceType">
  <xs:sequence>
    <xs:element minOccurs="0" name="Transforms" type="ds-rw:TransformsType"/>
  </xs:sequence>
  <xs:attribute name="WhichDocument" type="xs:IDREF" use="required"/>
  <xs:attribute name="RefURI" type="xs:anyURI" use="optional"/>
  <xs:attribute name="RefId" type="xs:string" use="optional"/>
</xs:complexType>
```

Each child element of SignedReferenceType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component SignedReference XML schema details]*

Component VerifyManifestResults

The results of verifying individual <ds:Reference>'s within a <ds:Manifest> are returned in the VerifyManifestResults component.

Below follows a list of the sub-components that MAY be present within this component:

The ManifestResult element MUST occur 1 or more times containing a sub-component. Each instance MUST satisfy the requirements specified in this document in section ManifestResultType. *[sub component ManifestResult details]*

Non-normative Comment:

[component VerifyManifestResults non normative details]

VerifyManifestResults – JSON Syntax

The VerifyManifestResultsType JSON object SHALL implement in JSON syntax the requirements defined in the VerifyManifestResults component.

Properties of the JSON object SHALL implement the sub-components of VerifyManifestResultsType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
ManifestResult	result	*[]*

The VerifyManifestResultsType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-VerifyManifestResultsType" : {
  "type" : "object",
  "properties" : {
    "result" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dss2-ManifestResultType"
      }
    }
  },
  "required" : ["result"]
}
```

[component VerifyManifestResults JSON schema details]

VerifyManifestResults – XML Syntax

The XML type VerifyManifestResultsType SHALL implement the requirements defined in the VerifyManifestResultscomponent.

The VerifyManifestResultsType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="VerifyManifestResultsType">
  <xs:sequence>
    <xs:element maxOccurs="unbounded" name="ManifestResult" type="dss2:ManifestResultType
  </xs:sequence>
</xs:complexType>
```

Each child element of VerifyManifestResultsType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component VerifyManifestResults XML schema details]*

Component ManifestResult

The VerifyManifestResults component is comprised of one or more ManifestResult

Below follows a list of the sub-components that MAY be present within this component:

The ReferenceXpath element MUST contain one instance of a string. *This element identifies the manifest reference, in the XML signature, to which this result pertains.*

The Status element MUST contain one instance of a URI. Its value is limited to an item of the following set:

- urn:oasis:names:tc:dss:1.0:manifeststatus:Valid
- urn:oasis:names:tc:dss:1.0:manifeststatus:Invalid

This element indicates the manifest validation outcome.

The optional NsPrefixMapping element MAY occur zero or more times containing a sub-component. If present each instance MUST satisfy the requirements specified in section NsPrefixMappingType. *[sub component NsPrefixMapping details]*

Non-normative Comment:

[component ManifestResult non normative details]

ManifestResult – JSON Syntax

The ManifestResultType JSON object SHALL implement in JSON syntax the requirements defined in the ManifestResult component.

Properties of the JSON object SHALL implement the sub-components of ManifestResultType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
ReferenceXpath	xPath	*[]*
Status	status	*[]*
NsPrefixMapping	nsDecl	*[]*

The ManifestResultType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-ManifestResultType" : {
  "type" : "object",
  "properties" : {
    "xPath" : {
      "type" : "string"
    },
    "status" : {
      "type" : "string",
      "enum" : ["urn:oasis:names:tc:dss:1.0:manifeststatus:Valid",
        "urn:oasis:names:tc:dss:1.0:manifeststatus:Invalid"]
    },
    "nsDecl" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dsb-NsPrefixMappingType"
      }
    }
  },
  "required" : ["xPath",
    "status"]
}
```

[component ManifestResult JSON schema details]

ManifestResult – XML Syntax

The XML type ManifestResultType SHALL implement the requirements defined in the ManifestResultcomponent.

The ManifestResultType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="ManifestResultType">
  <xs:sequence>
    <xs:element name="ReferenceXpath" type="xs:string"/>
    <xs:element name="Status">
      <xs:simpleType>
        <xs:restriction base="xs:anyURI">
          <xs:enumeration value="urn:oasis:names:tc:dss:1.0:manifeststatus:Valid"/>
          <xs:enumeration value="urn:oasis:names:tc:dss:1.0:manifeststatus:Invalid"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="NsPrefixMapping" type="dsb:NsPr
  </xs:sequence>
</xs:complexType>
```

Each child element of ManifestResultType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component ManifestResult XML schema details]*

Component UseVerificationTime

This UseVerificationTime component instructs the server to attempt to determine the signature's validity at the specified time, instead of a time determined by the server policy.

Below follows a list of the sub-components that MAY be present within this component:

The optional CurrentTime element MUST contain one instance of a boolean. Its default value is 'false'. *This element instructs the server to use its current time (normally the time associated with the server-side request processing).*

The optional SpecificTime element MUST contain one instance of a date/time value. *The SpecificTime element allows the client to manage manually the time instant used in the verification process. It SHOULD be expressed as UTC time (Coordinated Universal Time) to reduce confusion with the local time zone use.*

The optional Base64Content element MUST contain base64 encoded binary data. *The Base64Content element allows the provision of additional date/time data.*

Non-normative Comment:

[component UseVerificationTime non normative details]

UseVerificationTime – JSON Syntax

The UseVerificationTimeType JSON object SHALL implement in JSON syntax the requirements defined in the UseVerificationTime component.

Properties of the JSON object SHALL implement the sub-components of UseVerificationTimeType using JSON-specific names mapped as shown in the table below.

Element	**Implementing JSON member name**	**Comments**
CurrentTime	currTime	*[]*
SpecificTime	specTime	*[]*
Base64Content	b64Content	*[]*

The UseVerificationTimeType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-UseVerificationTimeType" : {
  "type" : "object",
  "properties" : {
    "currTime" : {
      "type" : "boolean",
      "default" : "false"
    },
    "specTime" : {
      "type" : "integer",
      "format" : "utc-millisec"
    },
    "b64Content" : {
      "type" : "string"
    }
  },
  "minProperties" : 1,
  "maxProperties" : 1
}
```

[component UseVerificationTime JSON schema details]

UseVerificationTime – XML Syntax

The XML type UseVerificationTimeType SHALL implement the requirements defined in the UseVerificationTimecomponent.

The UseVerificationTimeType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="UseVerificationTimeType">
  <xs:choice>
    <xs:element default="false" name="CurrentTime" type="xs:boolean"/>
    <xs:element name="SpecificTime" type="xs:dateTime"/>
    <xs:element maxOccurs="1" minOccurs="0" name="Base64Content" type="xs:base64Binary"/
  </xs:choice>
</xs:complexType>
```

Each child element of UseVerificationTimeType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component UseVerificationTime XML schema details]*

Component AdditionalTimeInfo

The AdditionalTimeInfo component contains other time instant(s) relevant in the context of the verification time determination.

Below follows a list of the sub-components that MAY be present within this component:

The value element MUST contain one instance of a date/time value. *[sub component value details]*

The Type element MUST contain one instance of a URI. Its value is limited to an item of the following set:

- urn:oasis:names:tc:dss:1.0:additionaltimeinfo:signatureTimestamp
- urn:oasis:names:tc:dss:1.0:additionaltimeinfo:signatureTimemark
- urn:oasis:names:tc:dss:1.0:additionaltimeinfo:signedObjectTimestamp
- urn:oasis:names:tc:dss:1.0:additionaltimeinfo:claimedSigningTime

The Type attribute qualifies the kind of time information included in the response. This specification defines the listed types, whose values MUST satisfy the format defined as xs:dateTime and SHOULD be expressed as UTC time (Coordinated Universal Time). Profiles MAY include and define new values for the Type attribute.

The optional Ref element MUST contain one instance of a string. *It allows to establish references to the source of the time information, and SHOULD be used when there is a need to disambiguate several AdditionalTimeInfo components with the same Type attribute.*

Non-normative Comment:

[component AdditionalTimeInfo non normative details]

AdditionalTimeInfo – JSON Syntax

The AdditionalTimeInfoType JSON object SHALL implement in JSON syntax the requirements defined in the AdditionalTimeInfo component.

Properties of the JSON object SHALL implement the sub-components of AdditionalTimeInfoType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
value	value	*[]*
Type	type	*[]*
Ref	ref	*[]*

The AdditionalTimeInfoType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-AdditionalTimeInfoType" : {
  "type" : "object",
  "properties" : {
    "value" : {
      "type" : "integer",
      "format" : "utc-millisec"
    },
    "type" : {
      "type" : "string",
      "format" : "uri"
    },
    "ref" : {
      "type" : "string"
    }
  },
  "required" : ["type"]
}
```

[component AdditionalTimeInfo JSON schema details]

AdditionalTimeInfo – XML Syntax

The XML type AdditionalTimeInfoType SHALL implement the requirements defined in the AdditionalTimeInfocomponent.

The AdditionalTimeInfoType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="AdditionalTimeInfoType">
  <xs:simpleContent>
    <xs:extension base="xs:dateTime">
      <xs:attribute name="Type" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:anyURI">
            <xs:enumeration value="urn:oasis:names:tc:dss:1.0:additionaltimeinfo:signatu
            <xs:enumeration value="urn:oasis:names:tc:dss:1.0:additionaltimeinfo:signatu
            <xs:enumeration value="urn:oasis:names:tc:dss:1.0:additionaltimeinfo:signedOb
            <xs:enumeration value="urn:oasis:names:tc:dss:1.0:additionaltimeinfo:claimedS
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="Ref" type="xs:string" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

Each child element of AdditionalTimeInfoType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component AdditionalTimeInfo XML schema details]*

Component VerificationTimeInfo

The VerificationTimeInfo component allows the client to obtain the time instant used by the server to validate the signature.

Below follows a list of the sub-components that MAY be present within this component:

The VerificationTime element MUST contain one instance of a date/time value. *This time instant used by the server when verifying the signature. It SHOULD be expressed as UTC time (Coordinated Universal Time) to reduce confusion with the local time zone use.*

The optional AdditionalTimeInfo element MAY occur zero or more times containing a sub-component. If present each instance MUST satisfy the requirements specified in this document in section

AdditionalTimeInfoType. *The AdditionalTimeInfo element can contain any other time instant(s) relevant in the context of the verification time determination.*

Non-normative Comment:

[component VerificationTimeInfo non normative details]

VerificationTimeInfo – JSON Syntax

The VerificationTimeInfoType JSON object SHALL implement in JSON syntax the requirements defined in the VerificationTimeInfo component.

Properties of the JSON object SHALL implement the sub-components of VerificationTimeInfoType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
VerificationTime	verificationTime	*[]*
AdditionalTimeInfo	additionalTimeInfo	*[]*

The VerificationTimeInfoType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-VerificationTimeInfoType" : {
  "type" : "object",
  "properties" : {
    "verificationTime" : {
      "type" : "integer",
      "format" : "utc-millisec"
    },
    "additionalTimeInfo" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dss2-AdditionalTimeInfoType"
      }
    }
  },
  "required" : ["verificationTime"]
}
```

[component VerificationTimeInfo JSON schema details]

VerificationTimeInfo – XML Syntax

The XML type VerificationTimeInfoType SHALL implement the requirements defined in the VerificationTimeInfocomponent.

The VerificationTimeInfoType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="VerificationTimeInfoType">
  <xs:sequence>
    <xs:element name="VerificationTime" type="xs:dateTime"/>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="AdditionalTimeInfo" type="dss2
  </xs:sequence>
</xs:complexType>
```

Each child element of VerificationTimeInfoType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component VerificationTimeInfo XML schema details]*

Component AdditionalKeyInfo

The AdditionalKeyInfo component provides the server with additional data (such as certificates and CRLs) which it can use to validate the signature.

Below follows a list of the sub-components that MAY be present within this component:

The optional X509Digest element MUST contain one instance of a sub-component. This element MUST satisfy the requirements specified in this document in section X509DigestType. *This element contains a base64-encoded digest of a certificate.*

The optional X509SubjectName element MUST contain one instance of a string. *This element contains an X.509 subject distinguished name that is represented as a string.*

The optional X509SKI element MUST contain one instance of base64 encoded binary data. *This element contains the base64 encoded value of a X509 V.3 SubjectKeyIdentifier.*

The optional X509Certificate element MUST contain one instance of a sub-component. This element MUST satisfy the requirements specified in section Base64DataType. *This element MAY contain certificates useful to build a certificate chain.*

The optional KeyName element MUST contain one instance of a string. *It contains a string value to identify the key.*

The optional X509CRL element MUST contain one instance of a sub-component. This element MUST satisfy the requirements specified in section Base64DataType. *In addition to the elements included in component 3.1.33 the X509CRL element holds a CRL.*

The optional OCSPResponse element MUST contain one instance of a sub-component. This element MUST satisfy the requirements specified in section Base64DataType. *This element can be used by the client to provide available OCSP information. The server MAY consider this information.*

The optional PoE element MUST contain one instance of a sub-component. This element MUST satisfy the requirements specified in section Base64DataType. *This element can be used by the client to provide 'Proof of Existence' data. Valid information MAY be considered within the validation process.*

Non-normative Comment:

[component AdditionalKeyInfo non normative details]

AdditionalKeyInfo – JSON Syntax

The AdditionalKeyInfoType JSON object SHALL implement in JSON syntax the requirements defined in the AdditionalKeyInfo component.

Properties of the JSON object SHALL implement the sub-components of AdditionalKeyInfoType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
X509Digest	x509Digest	*[]*
X509SubjectName	sub	*[]*
X509SKI	ski	*[]*
X509Certificate	cert	*[]*
KeyName	name	*[]*
X509CRL	crl	*[]*
OCSPResponse	ocsp	*[]*
PoE	poe	*[]*

The AdditionalKeyInfoType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-AdditionalKeyInfoType" : {
  "type" : "object",
  "properties" : {
    "ocspresponse" : {
      "$ref" : "#/definitions/dsb-Base64DataType"
    },
    "x509Digest" : {
      "$ref" : "#/definitions/dss2-X509DigestType"
    },
    "sub" : {
      "type" : "string"
    },
    "ski" : {
      "type" : "string"
    },
    "cert" : {
      "$ref" : "#/definitions/dsb-Base64DataType"
    },
    "name" : {
      "type" : "string"
    },
    "crl" : {
      "$ref" : "#/definitions/dsb-Base64DataType"
    },
    "ocsp" : {
      "$ref" : "#/definitions/dsb-Base64DataType"
    },
    "poe" : {
      "$ref" : "#/definitions/dsb-Base64DataType"
    }
  },
  "minProperties" : 1,
  "maxProperties" : 1
}
```

[component AdditionalKeyInfo JSON schema details]

AdditionalKeyInfo – XML Syntax

The XML type AdditionalKeyInfoType SHALL implement the requirements defined in the AdditionalKeyInfocomponent.

The AdditionalKeyInfoType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="AdditionalKeyInfoType">
  <xs:choice>
    <xs:element name="X509Digest" type="dss2:X509DigestType"/>
    <xs:element name="X509SubjectName" type="xs:string"/>
    <xs:element name="X509SKI" type="xs:base64Binary"/>
    <xs:element name="X509Certificate" type="dsb:Base64DataType"/>
    <xs:element name="KeyName" type="xs:string"/>
    <xs:element name="X509CRL" type="dsb:Base64DataType"/>
    <xs:element name="OCSPResponse" type="dsb:Base64DataType"/>
    <xs:element name="PoE" type="dsb:Base64DataType"/>
  </xs:choice>
</xs:complexType>
```

Each child element of AdditionalKeyInfoType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component AdditionalKeyInfo XML schema details]*

Component ProcessingDetails

The ProcessingDetails component elaborates on what signature verification steps succeeded or failed.

Below follows a list of the sub-components that MAY be present within this component:

The optional ValidDetail element MAY occur zero or more times containing a sub-component. If present each instance MUST satisfy the requirements specified in this document in section DetailType. *The ValidDetail element holds verification details that were evaluated and found to be valid.*

The optional IndeterminateDetail element MAY occur zero or more times containing a sub-component. If present each instance MUST satisfy the requirements specified in this document in section DetailType. *The IndeterminateDetail element holds verification details that could not be evaluated or were evaluated and returned an indeterminate result.*

The optional InvalidDetail element MAY occur zero or more times containing a sub-component. If present each instance MUST satisfy the requirements specified in this document in section DetailType. *The optional InvalidDetail element holds verification details that were evaluated and found to be invalid.*

Non-normative Comment:

[component ProcessingDetails non normative details]

ProcessingDetails – JSON Syntax

The ProcessingDetailsType JSON object SHALL implement in JSON syntax the requirements defined in the ProcessingDetails component.

Properties of the JSON object SHALL implement the sub-components of ProcessingDetailsType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
ValidDetail	valid	*[]*
IndeterminateDetail	indeterminate	*[]*
InvalidDetail	invalid	*[]*

The ProcessingDetailsType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-ProcessingDetailsType" : {
  "type" : "object",
  "properties" : {
    "valid" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dss2-DetailType"
      }
    },
    "indeterminate" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dss2-DetailType"
      }
    },
    "invalid" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dss2-DetailType"
      }
    }
  }
}
```

[component ProcessingDetails JSON schema details]

ProcessingDetails – XML Syntax

The XML type ProcessingDetailsType SHALL implement the requirements defined in the ProcessingDetailscomponent.

The ProcessingDetailsType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="ProcessingDetailsType">
  <xs:sequence>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="ValidDetail" type="dss2:Detail
    <xs:element maxOccurs="unbounded" minOccurs="0" name="IndeterminateDetail" type="dss2
    <xs:element maxOccurs="unbounded" minOccurs="0" name="InvalidDetail" type="dss2:Deta
  </xs:sequence>
</xs:complexType>
```

Each child element of ProcessingDetailsType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component ProcessingDetails XML schema details]*

Component Detail

[component Detail normative details]

Below follows a list of the sub-components that MAY be present within this component:

The optional Code element MUST contain a URI. *This URI which more precisely specifies why this detail is valid, invalid, or indeterminate. It must be a value defined by some other specification, since this specification defines no values for this element.*

The optional Message element MUST contain a sub-component. A given element MUST satisfy the requirements specified in section InternationalStringType. *This is a human-readable message which MAY be logged, used for debugging, etc.*

The optional Base64Content element MUST contain base64 encoded binary data.

The Type element MUST contain one instance of a URI. *The Type URI identifies the detail. It may be a value defined by this specification, or a value defined by some other specification. Multiple detail elements of the same Type may appear in a single ProcessingDetails component. For example, when a signature contains a certificate chain that certifies the signing key, there may be details of the same Type present for each certificate in the chain, describing how each certificate was processed.*

Non-normative Comment:

Multiple detail elements of the same Type may appear in a single ProcessingDetails. For example, when a signature contains a certificate chain that certifies the signing key, there may be details of the same Type present for each certificate in the chain, describing how each certificate was processed.

Detail – JSON Syntax

The DetailType JSON object SHALL implement in JSON syntax the requirements defined in the Detail component.

Properties of the JSON object SHALL implement the sub-components of DetailType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
Code	code	*[]*
Message	msg	*[]*
Base64Content	b64Content	*[]*
Type	type	*[]*

The DetailType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-DetailType" : {
  "type" : "object",
  "properties" : {
    "code" : {
      "type" : "string"
    },
    "msg" : {
      "$ref" : "#/definitions/dsb-InternationalStringType"
    },
    "b64Content" : {
      "type" : "string"
    },
    "type" : {
      "type" : "string",
      "format" : "uri"
    }
  },
  "required" : ["type"]
}
```

[component Detail JSON schema details]

Detail – XML Syntax

The XML type DetailType SHALL implement the requirements defined in the Detailcomponent.

The DetailType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="DetailType">
  <xs:sequence>
    <xs:element minOccurs="0" name="Code" type="xs:anyURI"/>
    <xs:element minOccurs="0" name="Message" type="dsb:InternationalStringType"/>
    <xs:element maxOccurs="1" minOccurs="0" name="Base64Content" type="xs:base64Binary"/:
  </xs:sequence>
  <xs:attribute name="Type" type="xs:anyURI" use="required"/>
</xs:complexType>
```

Each child element of DetailType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component Detail XML schema details]*

Component SigningTimeInfo

This SigningTimeInfo component allows the client to obtain the time instant associated to the signature creation.

Below follows a list of the sub-components that MAY be present within this component:

The SigningTime element MUST contain one instance of a date/time value. *This element returns the time value considered by the server to be the signature creation time.*

The optional SigningTimeBoundaries element MUST contain sub-components. *This element returns the trusted time values considered as lower and upper limits for the signing time.*

The optional LowerBoundary element MUST contain a date/time value. *The SigningTimeBoundaries element MUST contain at least one of the LowerBoundary or UpperBoundary elements.*

The optional UpperBoundary element MUST contain a date/time value. *The SigningTimeBoundaries element MUST contain at least one of the LowerBoundary or UpperBoundary elements*

Non-normative Comment:

[component SigningTimeInfo non normative details]

SigningTimeInfo – JSON Syntax

The SigningTimeInfoType JSON object SHALL implement in JSON syntax the requirements defined in the SigningTimeInfo component.

```
"dss2-SigningTimeInfoType:SigningTimeBoundaries" : {
  "type" : "object",
  "properties" : {
    "lowerBound" : {
      "type" : "integer",
      "format" : "utc-millisec"
    },
    "upperBound" : {
      "type" : "integer",
      "format" : "utc-millisec"
    }
  }
}
```

Properties of the JSON object SHALL implement the sub-components of SigningTimeInfoType using JSON-specific names mapped as shown in the table below.

Element	**Implementing JSON member name**	**Comments**
SigningTime	signingTime	*[]*
SigningTimeBoundaries	signingTimeBounds	*[]*
LowerBoundary	lowerBound	*[]*
UpperBoundary	upperBound	*[]*

The SigningTimeInfoType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-SigningTimeInfoType" : {
  "type" : "object",
  "properties" : {
    "signingTime" : {
      "type" : "integer",
      "format" : "utc-millisec"
    },
    "signingTimeBounds" : {
      "$ref" : "#/definitions/dss2-SigningTimeInfoType%3ASigningTimeBoundaries"
    }
  },
  "required" : ["signingTime"]
}
```

[component SigningTimeInfo JSON schema details]

SigningTimeInfo – XML Syntax

The XML type SigningTimeInfoType SHALL implement the requirements defined in the SigningTimeInfocomponent.

The SigningTimeInfoType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="SigningTimeInfoType">
  <xs:sequence>
    <xs:element name="SigningTime" type="xs:dateTime"/>
    <xs:element minOccurs="0" name="SigningTimeBoundaries">
      <xs:complexType>
        <xs:sequence>
          <xs:element minOccurs="0" name="LowerBoundary" type="xs:dateTime"/>
          <xs:element minOccurs="0" name="UpperBoundary" type="xs:dateTime"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

Each child element of SigningTimeInfoType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component SigningTimeInfo XML schema details]*

Component AugmentedSignature

The AugmentedSignature component contains the resulting augmented signature or timestamp or, in the case of a signature being enveloped in an output document, a pointer to the signature.

Below follows a list of the sub-components that MAY be present within this component:

The SignatureObject element MUST contain one instance of a sub-component. This element MUST satisfy the requirements specified in this document in section SignatureObjectType. *This element contains an augmented signature or timestamp.*

The optional Type element MUST contain one instance of a URI. *The URI defines what 'augmentation' was applied to the signature.*

Non-normative Comment:

[component AugmentedSignature non normative details]

AugmentedSignature – JSON Syntax

The AugmentedSignatureType JSON object SHALL implement in JSON syntax the requirements defined in the AugmentedSignature component.

Properties of the JSON object SHALL implement the sub-components of AugmentedSignatureType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
SignatureObject	sigObj	*[]*
Type	type	*[]*

The AugmentedSignatureType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-AugmentedSignatureType" : {
  "type" : "object",
  "properties" : {
    "sigObj" : {
      "$ref" : "#/definitions/dss2-SignatureObjectType"
    },
    "type" : {
      "type" : "string",
      "format" : "uri"
    }
  },
  "required" : ["sigObj"]
}
```

[component AugmentedSignature JSON schema details]

AugmentedSignature – XML Syntax

The XML type AugmentedSignatureType SHALL implement the requirements defined in the AugmentedSignaturecomponent.

The AugmentedSignatureType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="AugmentedSignatureType">
  <xs:sequence>
    <xs:element name="SignatureObject" type="dss2:SignatureObjectType"/>
  </xs:sequence>
  <xs:attribute name="Type" type="xs:anyURI" use="optional"/>
</xs:complexType>
```

Each child element of AugmentedSignatureType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component AugmentedSignature XML schema details]*

Component ReturnTransformedDocument

The ReturnTransformedDocument component instructs the server to return an input document to which the XML signature transforms specified by a particular <ds:Reference> have been applied. The <ds: Reference> is indicated by the zero-based WhichReference attribute (0 means the first <ds:Reference> in the signature, 1 means the second, and so on). Multiple occurrences of this optional input can be present in a single verify request message. Each occurrence will generate a corresponding optional output.

Below follows a list of the sub-components that MAY be present within this component:

The WhichReference element MUST contain one instance of an integer. *To match outputs to inputs, each TransformedDocument will contain a WhichReference attribute which matches the corresponding optional input.*

Non-normative Comment:

[component ReturnTransformedDocument non normative details]

ReturnTransformedDocument – JSON Syntax

The ReturnTransformedDocumentType JSON object SHALL implement in JSON syntax the requirements defined in the ReturnTransformedDocument component.

Properties of the JSON object SHALL implement the sub-components of ReturnTransformedDocumentType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
WhichReference	whichRef	*[]*

The ReturnTransformedDocumentType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-ReturnTransformedDocumentType" : {
  "type" : "object",
  "properties" : {
    "whichRef" : {
      "type" : "integer"
    }
  },
  "required" : ["whichRef"]
}
```

[component ReturnTransformedDocument JSON schema details]

ReturnTransformedDocument – XML Syntax

The XML type ReturnTransformedDocumentType SHALL implement the requirements defined in the ReturnTransformedDocumentcomponent.

The ReturnTransformedDocumentType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="ReturnTransformedDocumentType">
  <xs:attribute name="WhichReference" type="xs:integer" use="required"/>
</xs:complexType>
```

Each child element of ReturnTransformedDocumentType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component ReturnTransformedDocument XML schema details]*

Component TransformedDocument

The TransformedDocument component contains a document corresponding to the specified <ds: Reference>, after all the transforms in the reference have been applied.

Below follows a list of the sub-components that MAY be present within this component:

The Document element MUST contain one instance of a sub-component. This element MUST satisfy the requirements specified in this document in section DocumentType. *This element contains the transformed document.*

The WhichReference element MUST contain one instance of an integer. *To match outputs to inputs, each TransformedDocument will contain a WhichReference element which matches the corresponding optional input.*

Non-normative Comment:

[component TransformedDocument non normative details]

TransformedDocument – JSON Syntax

The TransformedDocumentType JSON object SHALL implement in JSON syntax the requirements defined in the TransformedDocument component.

Properties of the JSON object SHALL implement the sub-components of TransformedDocumentType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
Document	doc	*[]*
WhichReference	whichRef	*[]*

The TransformedDocumentType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-TransformedDocumentType" : {
  "type" : "object",
  "properties" : {
    "doc" : {
      "$ref" : "#/definitions/dss2-DocumentType"
    },
    "whichRef" : {
      "type" : "integer"
    }
  },
  "required" : ["doc",
    "whichRef"]
}
```

[component TransformedDocument JSON schema details]

TransformedDocument – XML Syntax

The XML type TransformedDocumentType SHALL implement the requirements defined in the TransformedDocumentcomponent.

The TransformedDocumentType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="TransformedDocumentType">
  <xs:sequence>
    <xs:element name="Document" type="dss2:DocumentType"/>
  </xs:sequence>
  <xs:attribute name="WhichReference" type="xs:integer" use="required"/>
</xs:complexType>
```

Each child element of TransformedDocumentType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component TransformedDocument XML schema details]*

Request/Response related data structures defined in this document

The XML elements of this section are defined in the XML namespace 'http://docs.oasis-open.org/dss/ns/core'.*[category request in namespace http://docs.oasis-open.org/dss/ns/core explanation]*

Component InputDocuments

This element is used to send input documents to a DSS server, whether for signing or verifying. An input document can be any piece of data that can be used as input to a signature or timestamp calculation. An input document can even be a signature or timestamp (for example, a pre-existing signature can be counter-signed or timestamped). An input document could also be a <ds:Manifest>, allowing the client to handle manifest creation while using the server to create the rest of the signature. Manifest validation is supported by an optional input / output.

Below follows a list of the sub-components that MAY be present within this component:

The Document element MAY occur zero or more times containing a sub-component. If present each instance MUST satisfy the requirements specified in this document in section DocumentType. *[sub component Document details]*

The TransformedData element MAY occur zero or more times containing a sub-component. If present each instance MUST satisfy the requirements specified in this document in section TransformedDataType . *It contains the binary output of a chain of transforms applied by a client.*

The DocumentHash element MAY occur zero or more times containing a sub-component. If present each instance MUST satisfy the requirements specified in this document in section DocumentHashType. *It contains a set of digest algorithm and the corresponding hashes. Required transformation steps*

Non-normative Comment:

[component InputDocuments non normative details]

InputDocuments – JSON Syntax

The InputDocumentsType JSON object SHALL implement in JSON syntax the requirements defined in the InputDocuments component.

Properties of the JSON object SHALL implement the sub-components of InputDocumentsType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
Document	doc	*[]*
TransformedData	transformed	*[]*

DocumentHash	docHash	*[]*

The InputDocumentsType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-InputDocumentsType" : {
  "type" : "object",
  "properties" : {
    "doc" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dss2-DocumentType"
      }
    },
    "transformed" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dss2-TransformedDataType"
      }
    },
    "docHash" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dss2-DocumentHashType"
      }
    }
  }
}
```

[component InputDocuments JSON schema details]

InputDocuments – XML Syntax

The XML type InputDocumentsType SHALL implement the requirements defined in the InputDocumentscomponent.

The InputDocumentsType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="InputDocumentsType">
  <xs:sequence>
    <xs:sequence maxOccurs="unbounded" minOccurs="0">
      <xs:element name="Document" type="dss2:DocumentType"/>
    </xs:sequence>
    <xs:sequence maxOccurs="unbounded" minOccurs="0">
      <xs:element name="TransformedData" type="dss2:TransformedDataType"/>
    </xs:sequence>
    <xs:sequence maxOccurs="unbounded" minOccurs="0">
      <xs:element name="DocumentHash" type="dss2:DocumentHashType"/>
    </xs:sequence>
  </xs:sequence>
</xs:complexType>
```

Each child element of InputDocumentsType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component InputDocuments XML schema details]*

Component DocumentBase

The DocumentBaseType forwards its elements to the components DocumentType, TransformedDataType and DocumentHashType. The DocumentBaseType contains the basic information shared by the *inheriting components and remaining persistent during the process from input document retrieval until digest calculation for the relevant document.*

Below follows a list of the sub-components that MAY be present within this component:

The optional Id element MUST contain one instance of a unique identifier. *This identifier gives the input document a unique label within a particular request message. Through this identifier, an optional input can refer to a single input document. Using this identifier and the IdRef element it is possible to avoid redundant content.*

The optional RefURI element MUST contain one instance of a URI. *This specifies the value for a <ds:Reference> element's URI attribute when referring to this input document. The RefURI element SHOULD be specified. Not more than one RefURI element may be omitted in a single signing request.*

The optional RefType element MUST contain one instance of a URI. *This specifies the value for a <ds:Reference> element's Type attribute when referring to this input document.*

The optional SchemaRefs element MUST contain one instance of a unique identifier reference. *The identified schemas are to be used to process the Id element during parsing and for XPath evaluation. If anything else but Schema are referred to, the server MUST report an error. If a referred to Schema is not used by the XML document instance this MAY be ignored or reported to the client in the subcomponent ResultMessage. The Document is assumed to be valid against the first Schema referred to by SchemaRefs. If a Schemas element is referred to first by SchemaRefs the document is assumed to be valid against the first Schema inside SchemaRefs. In both cases, the remaining schemas may occur in any order and are used either directly or indirectly by the first schema. If present, the server MUST use the schemas to identify the Id element and MAY also perform complete validation against the schemas.*

Non-normative Comment:

It is recommended to use xml:id as defined in [xml:id] as id in the payload being referenced by a <ds:Reference>, because the schema then does not have to be supplied for identifying the ID attributes.

DocumentBase – JSON Syntax

The DocumentBaseType JSON object SHALL implement in JSON syntax the requirements defined in the DocumentBase component.

Properties of the JSON object SHALL implement the sub-components of DocumentBaseType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
Id	ID	*[]*
RefURI	refURI	*[]*
RefType	refType	*[]*
SchemaRefs	schemaRefs	*[]*

The DocumentBaseType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-DocumentBaseType" : {
  "type" : "object",
  "properties" : {
    "ID" : {
      "type" : "string"
    },
    "refURI" : {
      "type" : "string"
    },
    "refType" : {
      "type" : "string"
    },
    "schemaRefs" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dss2-DocumentType"
      }
    }
  }
}
```

[component DocumentBase JSON schema details]

DocumentBase – XML Syntax

The XML type DocumentBaseType SHALL implement the requirements defined in the DocumentBasecomponent.

The DocumentBaseType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType abstract="true" name="DocumentBaseType">
  <xs:attribute name="ID" type="xs:ID" use="optional"/>
  <xs:attribute name="RefURI" type="xs:anyURI" use="optional"/>
  <xs:attribute name="RefType" type="xs:anyURI" use="optional"/>
  <xs:attribute name="SchemaRefs" type="xs:IDREFS" use="optional"/>
</xs:complexType>
```

Each child element of DocumentBaseType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component DocumentBase XML schema details]*

Component Document

The Document component contains input data for DSS processing.

Below follows a list of the sub-components that MAY be present within this component:

The Base64Data element MUST contain one instance of a sub-component. This element MUST satisfy the requirements specified in section Base64DataType. *[sub component Base64Data details]*

Non-normative Comment:

[component Document non normative details]

Document – JSON Syntax

The DocumentType JSON object SHALL implement in JSON syntax the requirements defined in the Document component.

Properties of the JSON object SHALL implement the sub-components of DocumentType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
Base64Data	b64Data	*[]*

The DocumentType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-DocumentType" : {
  "type" : "object",
  "properties" : {
    "ID" : {
      "type" : "string"
    },
    "refURI" : {
      "type" : "string"
    },
    "refType" : {
      "type" : "string"
    },
    "schemaRefs" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dss2-DocumentType"
      }
    },
    "b64Data" : {
      "$ref" : "#/definitions/dsb-Base64DataType"
    }
  },
  "required" : ["b64Data"]
}
```

[component Document JSON schema details]

Document – XML Syntax

The XML type DocumentType SHALL implement the requirements defined in the Documentcomponent.

The DocumentType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="DocumentType">
  <xs:complexContent>
    <xs:extension base="dss2:DocumentBaseType">
      <xs:sequence>
        <xs:element maxOccurs="1" minOccurs="1" name="Base64Data" type="dsb:Base64DataTyp
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Each child element of DocumentType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component Document XML schema details]*

Component TransformedData

[component TransformedData normative details]

Below follows a list of the sub-components that MAY be present within this component:

The optional Transforms element MUST contain a sub-component. A given element MUST satisfy the requirements specified in section TransformsType. *This is the sequence of transforms applied by the client. It specifies the value for a <ds:Reference> element's <ds:Transforms> child element. In other words, this specifies transforms that the client has already applied to the input document before the server will hash it. This component is required on a SignRequest, optional on a VerifyRequest.*

The Base64Data element MUST contain one instance of a sub-component. This element MUST satisfy the requirements specified in section Base64DataType. *This element gives the binary output of a sequence of transforms to be hashed at the server side.*

The optional WhichReference element MUST contain one instance of an integer. *As there may be multiple TransformedDataType / DocumentHashType components of the same document having the same URI [RFC 2396] and RefType on a SignRequest or VerifyRequest - their correspondance to an already existing <ds:Reference> however needs to be established on a VerifyRequest only. There is a need to disambiguate such cases. This element hence offers a way to clearly identify the <ds:Reference> when URI and RefType match multiple components. The corresponding <ds:Reference> is indicated by this zero-based WhichReference element (0 means the first <ds:Reference> in the signature, 1 means the second, and so on). This component is ignored on a SignRequest, optional on a VerifyRequest.*

Non-normative Comment:

It may be possible to establish the <ds:References> / TransformedDataType / DocumentHashType correspondence by comparing the optionally supplied chain of transforms to those of the <ds: References> having the same URI and RefType in the supplied <ds:Signature> if this chain of transform has been supplied. This can be quite expensive and even outnumber the advantages of TransformedDataType / DocumentHashType.

TransformedData – JSON Syntax

The TransformedDataType JSON object SHALL implement in JSON syntax the requirements defined in the TransformedData component.

Properties of the JSON object SHALL implement the sub-components of TransformedDataType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
Transforms	transforms	*[]*
Base64Data	b64Data	*[]*
WhichReference	whichRef	*[]*

The TransformedDataType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-TransformedDataType" : {
  "type" : "object",
  "properties" : {
    "ID" : {
      "type" : "string"
    },
    "refURI" : {
      "type" : "string"
    },
    "refType" : {
      "type" : "string"
    },
    "schemaRefs" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dss2-DocumentType"
      }
    },
    "transforms" : {
      "$ref" : "#/definitions/dsigrw-TransformsType"
    },
    "b64Data" : {
      "$ref" : "#/definitions/dsb-Base64DataType"
    },
    "whichRef" : {
      "type" : "integer"
    }
  },
  "required" : ["b64Data"]
}
```

[component TransformedData JSON schema details]

TransformedData – XML Syntax

The XML type TransformedDataType SHALL implement the requirements defined in the TransformedDatacomponent.

The TransformedDataType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="TransformedDataType">
  <xs:complexContent>
    <xs:extension base="dss2:DocumentBaseType">
      <xs:sequence>
        <xs:element minOccurs="0" name="Transforms" type="ds-rw:TransformsType"/>
        <xs:element name="Base64Data" type="dsb:Base64DataType"/>
      </xs:sequence>
      <xs:attribute name="WhichReference" type="xs:integer" use="optional"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Each child element of TransformedDataType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component TransformedData XML schema details]*

Component DocumentHash

The DocumentHash component represents a document that will not be transported to the server but just the calculated digest of it. This may be useful to limit the amount of data transferred or to ensure privacy of the document.

Below follows a list of the sub-components that MAY be present within this component:

The optional Transforms element MUST contain a sub-component. A given element MUST satisfy the requirements specified in section TransformsType. *It specifies the value for a <ds:Reference> element's <ds:Transforms> child element when referring to this document hash. In other words, this specifies transforms that the client has already applied to the input document before hashing it. This component is required on a SignRequest, optional on a VerifyRequest. This component is required on a SignRequest, optional on a VerifyRequest.*

The DigestInfos element MUST occur 1 or more times containing a sub-component. Each instance MUST satisfy the requirements specified in section DigestInfoType. *This element MAY contain more than one DigestInfo sub-component to represent the digest values calculated with different digest algorithms. This may be useful when a requestor doesn't know upfront which digest algorithms are supported / accepted by the server for signing. In the case of a verification request the client may not be able to parse the signature and instead calculate the digest for a comprehensive set of digest algorithms.*

The optional WhichReference element MUST contain one instance of an integer. *As there may be multiple TransformedDataType / DocumentHashType components of the same document having the same URI [RFC 2396] and RefType on a SignRequest or VerifyRequest - their correspondence to an already existing <ds:Reference> however needs to be established on a VerifyRequest only. There is a need to disambiguate such cases. This element hence offers a way to clearly identify the <ds:Reference> when URI and RefType match multiple components. The corresponding <ds:Reference> is indicated by this zero-based WhichReference element (0 means the first <ds:Reference> in the signature, 1 means the second, and so on).*

Non-normative Comment:

[component DocumentHash non normative details]

DocumentHash – JSON Syntax

The DocumentHashType JSON object SHALL implement in JSON syntax the requirements defined in the DocumentHash component.

Properties of the JSON object SHALL implement the sub-components of DocumentHashType using JSON-specific names mapped as shown in the table below.

Element	**Implementing JSON member name**	**Comments**
Transforms	transforms	*[]*
DigestInfos	dis	*[]*
WhichReference	whichRef	*[]*

The DocumentHashType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-DocumentHashType" : {
  "type" : "object",
  "properties" : {
    "ID" : {
      "type" : "string"
    },
    "refURI" : {
      "type" : "string"
    },
    "refType" : {
      "type" : "string"
    },
    "schemaRefs" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dss2-DocumentType"
      }
    },
    "transforms" : {
      "$ref" : "#/definitions/dsigrw-TransformsType"
    },
    "dis" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dsb-DigestInfoType"
      }
    },
    "whichRef" : {
      "type" : "integer"
    }
  },
  "required" : ["dis"]
}
```

[component DocumentHash JSON schema details]

DocumentHash – XML Syntax

The XML type DocumentHashType SHALL implement the requirements defined in the DocumentHashcomponent.

The DocumentHashType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="DocumentHashType">
  <xs:complexContent>
    <xs:extension base="dss2:DocumentBaseType">
      <xs:sequence>
        <xs:element minOccurs="0" name="Transforms" type="ds-rw:TransformsType"/>
        <xs:element maxOccurs="unbounded" minOccurs="1" name="DigestInfos" type="dsb:Dig
      </xs:sequence>
      <xs:attribute name="WhichReference" type="xs:integer" use="optional"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Each child element of DocumentHashType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component DocumentHash XML schema details]*

Component SignatureObject

The SignatureObject component contains a signature or timestamp of some sort. This element is returned in a sign response message, and sent in a verify request message.

Below follows a list of the sub-components that MAY be present within this component:

The optional Base64Signature element MUST contain one instance of a sub-component. This element MUST satisfy the requirements specified in section Base64DataType. *A base64 encoding of some arbitrary signature, such as a XML signature [XMLDSIG], PGP [RFC 2440] or CMS [RFC 5652] signature or a RFC 3161 timestamp [RFC 3161]. The type of signature is specified by the MimeType element of the Base64DataType component. Profiles MAY define the handling of additional types.*

The optional SignaturePtr element MUST contain one instance of a sub-component. This element MUST satisfy the requirements specified in this document in section SignaturePtrType. *This element is used to point to an XML signature in an input (for a verify request) or output (for a sign response) document in which a signature is enveloped.*

The optional SchemaRefs element MUST contain one instance of a unique identifier reference. *The identified schemas are to be used to process the Id elements during parsing and for XPath evaluation. If anything else but <Schema> are referred to, the server MUST report an error. If a referred to <Schema> is not used by the XML document instance this MAY be ignored or reported to the client in the Fehler! Verweisquelle konnte nicht gefunden werden. subcomponent ResultMessage (for the definition of Schema subcomponent see the specification of 3.1.30)*

Non-normative Comment:

[component SignatureObject non normative details]

SignatureObject – JSON Syntax

The SignatureObjectType JSON object SHALL implement in JSON syntax the requirements defined in the SignatureObject component.

Properties of the JSON object SHALL implement the sub-components of SignatureObjectType using JSON-specific names mapped as shown in the table below.

Element	**Implementing JSON member name**	**Comments**
Base64Signature	b64Sig	*[]*
SignaturePtr	sigPtr	*[]*
SchemaRefs	schemaRefs	*[]*

The SignatureObjectType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-SignatureObjectType" : {
  "type" : "object",
  "properties" : {
    "b64Sig" : {
      "$ref" : "#/definitions/dsb-Base64DataType"
    },
    "sigPtr" : {
      "$ref" : "#/definitions/dss2-SignaturePtrType"
    },
    "schemaRefs" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dss2-DocumentBaseType"
      }
    }
  },
  "minProperties" : 1
}
```

[component SignatureObject JSON schema details]

SignatureObject – XML Syntax

The XML type SignatureObjectType SHALL implement the requirements defined in the SignatureObjectcomponent.

The SignatureObjectType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="SignatureObjectType">
  <xs:choice>
    <xs:element name="Base64Signature" type="dsb:Base64DataType"/>
    <xs:element name="SignaturePtr" type="dss2:SignaturePtrType"/>
  </xs:choice>
  <xs:attribute name="SchemaRefs" type="xs:IDREFS" use="optional"/>
</xs:complexType>
```

Each child element of SignatureObjectType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component SignatureObject XML schema details]*

Component SignaturePtr

The SignaturePtr component is used to point to a signature in an input (for a verify request) or output (for a sign response) document in which a signature is enveloped.

Below follows a list of the sub-components that MAY be present within this component:

The optional NsPrefixMapping element MAY occur zero or more times containing a sub-component. If present each instance MUST satisfy the requirements specified in section NsPrefixMappingType. *[sub component NsPrefixMapping details]*

The WhichDocument element MUST contain one instance of a unique identifier reference. *This element identifies the input document being pointed at.*

The optional XPath element MUST contain one instance of a string. *This element identifies the signature element being pointed at within the selected document. The XPath expression is evaluated from the root node (see section 5.1 of [XPATH]) of the document identified by WhichDocument. The context node for the XPath evaluation is the document's DocumentElement (see section 2.1 Well-Formed XML Documents [XML]). Regarding namespace declarations for the expression necessary for evaluation see section 1 of [XPATH].*

Non-normative Comment:

[component SignaturePtr non normative details]

SignaturePtr – JSON Syntax

The SignaturePtrType JSON object SHALL implement in JSON syntax the requirements defined in the SignaturePtr component.

Properties of the JSON object SHALL implement the sub-components of SignaturePtrType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
NsPrefixMapping	nsDecl	*[]*
WhichDocument	whichDoc	*[]*
XPath	xPath	*[]*

The SignaturePtrType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dss2-SignaturePtrType" : {
  "type" : "object",
  "properties" : {
    "xpath" : {
      "type" : "string"
    },
    "nsDecl" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dsb-NsPrefixMappingType"
      }
    },
    "whichDoc" : {
      "$ref" : "#/definitions/dss2-DocumentBaseType"
    },
    "xPath" : {
      "type" : "string"
    }
  },
  "required" : ["whichDoc"]
}
```

[component SignaturePtr JSON schema details]

SignaturePtr – XML Syntax

The XML type SignaturePtrType SHALL implement the requirements defined in the SignaturePtrcomponent.

The SignaturePtrType XML element is defined in XML Schema [DSS2XSD], and is copied below for information.

```
<xs:complexType name="SignaturePtrType">
  <xs:sequence>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="NsPrefixMapping" type="dsb:NsPr
  </xs:sequence>
  <xs:attribute name="WhichDocument" type="xs:IDREF" use="required"/>
  <xs:attribute name="XPath" type="xs:string" use="optional"/>
</xs:complexType>
```

Each child element of SignaturePtrType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component SignaturePtr XML schema details]*

Referenced from DSS-X base

Component Base64Data

The Base64Data component is a generic holder for arbitrary data. In addition to the data itself it also contains additional elements to qualify the MimeType of the data. It also offers an Id / Reference pair to implement a deduplication strategy, useful especially for bigger data blobs. The content is contained inside the mutually exclusive elements Value or AttRefURI.

Below follows a list of the sub-components that MAY be present within this component:

The optional Value element MUST contain one instance of base64 encoded binary data. *This element holds an instance of generic content. This could be a document to be signed, a signature, a schema or other data.*

The optional AttRef element MUST contain one instance of sub-component. This element MUST satisfy the requirements specified in section AttachmentReferenceType. *This element allows to reference content that is transferred in a non-inlined way. These mechanisms may take advantage of optimizations (e.g. optimized transfer encodings). The content of MAY be integrity-protected by a message digest.*

The optional MimeType element MUST contain one instance of a string. *This element is denoting the type of the arbitrary data in the value element or the referenced attachment.*

The optional Id element MUST contain one instance of a unique identifier. *This identifier gives the binary data a unique label within a particular message. Using this identifier and the IdRef element it is possible to avoid redundant content.*

The optional IdRef element MUST contain one instance of a unique identifier reference. *This element identifies another binary data element within a particular message.*

Non-normative Comment:

There are different standards defined for handling and referencing an attachment. Maybe there will be more to come. Therefore the attachment reference mechanism is somehow generic here. Note: If MIME is used as encapsulation mechanism, the MIME content-type is available via a MIME header. However, the MIME headers may not be available to implementations and the SOAP 1.2 attachment feature is not restricted to MIME. Further the MIME header is not secured by the AttachmentReference's DigestInfo, which is calculated over the binary attachment data (not including the MIME headers).

Base64Data – JSON Syntax

The Base64DataType JSON object SHALL implement in JSON syntax the requirements defined in the Base64Data component.

Properties of the JSON object SHALL implement the sub-components of Base64DataType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
Value	val	*[]*
AttRef	attRef	*[]*
MimeType	mimeType	*[]*
Id	ID	*[]*
IdRef	idRef	*[]*

The Base64DataType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dsb-Base64DataType" : {
  "type" : "object",
  "properties" : {
    "ID" : {
      "type" : "string"
    },
    "val" : {
      "type" : "string"
    },
    "attRef" : {
      "$ref" : "#/definitions/dsb-AttachmentReferenceType"
    },
    "mimeType" : {
      "type" : "string"
    },
    "idRef" : {
      "type" : "string"
    }
  },
  "minProperties" : 0
}
```

[component Base64Data JSON schema details]

Base64Data – XML Syntax

The XML element is defined in the XML namespace 'http://docs.oasis-open.org/dss-x/ns/base' .The XML type Base64DataType SHALL implement the requirements defined in the Base64Datacomponent.

The Base64DataType XML element is defined in XML Schema [DSBXSD], and is copied below for information.

```
<xs:complexType name="Base64DataType">
  <xs:choice minOccurs="0">
    <xs:element name="Value" type="xs:base64Binary"/>
    <xs:element name="AttRef" type="dsb:AttachmentReferenceType"/>
  </xs:choice>
  <xs:attribute name="MimeType" type="xs:string" use="optional"/>
  <xs:attribute name="ID" type="xs:ID" use="optional"/>
  <xs:attribute name="IDREF" type="xs:IDREF" use="optional"/>
</xs:complexType>
```

Each child element of Base64DataType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *The elements 'Id' and 'IdRef' have slightly different names ('ID' and 'IDREF') within XML syntax to match the XML schema standards for unique identifiers and their reference.*

Component AttachmentReference

Applications MAY support SOAP 1.2 attachment feature [SOAPAtt] or other attachment specifications (e. g. [SOAPMtom]) to transmit documents.

Below follows a list of the sub-components that MAY be present within this component:

The optional DigestInfo element MAY occur zero or more times containing sub-component. If present each instance MUST satisfy the requirements specified in section DigestInfoType. *An element of this type can be used to ensure the integrity of the attachment data. If these elements are supplied the server SHOULD compute a message digest using the algorithm given in DigestMethod over the binary data in the octet stream and compare it against the supplied DigestValue. If the comparison fails then a RequesterError qualified by a GeneralError and an appropriate message containing the AttRefURI is returned.*

The AttRefURI element MUST contain one instance of a URI. *SOAP 1.2 attachment feature [SOAPAtt] states that any secondary part ("attachment") can be referenced by a URI of any URI scheme. AttRefURI refers to such a secondary part ("attachment") and MUST resolve within the compound SOAP message. The default encapsulation mechanism is MIME as specified in the WS-I Attachments Profile [WS-I-Att] (cf. swaRef, http://www.ws-i.org/Profiles/AttachmentsProfile-1.0. html#Referencing_Attachments_from_the_SOAP_Envelope).*

Non-normative Comment:

[component AttachmentReference non normative details]

AttachmentReference – JSON Syntax

The AttachmentReferenceType JSON object SHALL implement in JSON syntax the requirements defined in the AttachmentReference component.

Properties of the JSON object SHALL implement the sub-components of AttachmentReferenceType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
DigestInfo	di	*[]*
AttRefURI	attURI	*[]*

The AttachmentReferenceType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dsb-AttachmentReferenceType" : {
  "type" : "object",
  "properties" : {
    "di" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dsb-DigestInfoType"
      }
    },
    "attURI" : {
      "type" : "string"
    }
  },
  "required" : ["attURI"]
}
```

[component AttachmentReference JSON schema details]

AttachmentReference – XML Syntax

The XML element is defined in the XML namespace 'http://docs.oasis-open.org/dss-x/ns/base' .The XML type AttachmentReferenceType SHALL implement the requirements defined in the AttachmentReferencecomponent.

The AttachmentReferenceType XML element is defined in XML Schema [DSBXSD], and is copied below for information.

```
<xs:complexType name="AttachmentReferenceType">
  <xs:sequence>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="DigestInfo" type="dsb:DigestInf
  </xs:sequence>
  <xs:attribute name="AttRefURI" type="xs:anyURI" use="required"/>
</xs:complexType>
```

Each child element of AttachmentReferenceType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component AttachmentReference XML schema details]*

Component DigestInfo

The DigestInfo component holds a digest value and an identification of the used digest algorithm. The DigestMethod isn't strongly typed intentionally to support a broad variety of identifiers.

Below follows a list of the sub-components that MAY be present within this component:

The DigestMethod element MUST contain one instance of a string. *The string describes the digest algorithm in an appropriate way for the server side processing. Depending on the signature format this may be an OID (e.g. '2.16.840.1.101.3.4.2.1'), an URI (e.g. 'http://www.w3.org/2001/04/xmlenc#sha256 ') or a descriptive string ('SHA-256').*

The DigestValue element MUST contain one instance of base64 encoded binary data. *[sub component DigestValue details]*

Non-normative Comment:

[component DigestInfo non normative details]

DigestInfo – JSON Syntax

The DigestInfoType JSON object SHALL implement in JSON syntax the requirements defined in the DigestInfo component.

Properties of the JSON object SHALL implement the sub-components of DigestInfoType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
DigestMethod	alg	*[]*
DigestValue	val	*[]*

The DigestInfoType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dsb-DigestInfoType" : {
  "type" : "object",
  "properties" : {
    "alg" : {
      "type" : "string"
    },
    "val" : {
      "type" : "string"
    }
  },
  "required" : ["alg",
    "val"]
}
```

[component DigestInfo JSON schema details]

DigestInfo – XML Syntax

The XML element is defined in the XML namespace 'http://docs.oasis-open.org/dss-x/ns/base' .The XML type DigestInfoType SHALL implement the requirements defined in the DigestInfocomponent.

The DigestInfoType XML element is defined in XML Schema [DSBXSD], and is copied below for information.

```
<xs:complexType name="DigestInfoType">
  <xs:sequence>
    <xs:element name="DigestMethod" type="xs:string"/>
    <xs:element name="DigestValue" type="xs:base64Binary"/>
  </xs:sequence>
</xs:complexType>
```

Each child element of DigestInfoType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component DigestInfo XML schema details]*

Component NsPrefixMapping

The NsPrefixMapping component defines the mapping of namespace URIs to namespace prefixes. This is required to evaluate XPath expression when using transport syntaxes that don’t support namespace.

Below follows a list of the sub-components that MAY be present within this component:

The NamespaceURI element MUST contain one instance of a URI. *[sub component NamespaceURI details]*

The NamespacePrefix element MUST contain one instance of a string. *[sub component NamespacePrefix details]*

Non-normative Comment:

[component NsPrefixMapping non normative details]

NsPrefixMapping – JSON Syntax

The NsPrefixMappingType JSON object SHALL implement in JSON syntax the requirements defined in the NsPrefixMapping component.

Properties of the JSON object SHALL implement the sub-components of NsPrefixMappingType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
NamespaceURI	uri	*[]*
NamespacePrefix	pre	*[]*

The NsPrefixMappingType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dsb-NsPrefixMappingType" : {
  "type" : "object",
  "properties" : {
    "uri" : {
      "type" : "string"
    },
    "pre" : {
      "type" : "string"
    }
  },
  "required" : ["uri",
    "pre"]
}
```

[component NsPrefixMapping JSON schema details]

NsPrefixMapping – XML Syntax

The XML element is defined in the XML namespace 'http://docs.oasis-open.org/dss-x/ns/base' .The XML type NsPrefixMappingType SHALL implement the requirements defined in the NsPrefixMappingcomponent.

The NsPrefixMappingType XML element is defined in XML Schema [DSBXSD], and is copied below for information.

```
<xs:complexType name="NsPrefixMappingType">
  <xs:sequence>
    <xs:element name="NamespaceURI" type="xs:anyURI"/>
    <xs:element name="NamespacePrefix" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Each child element of NsPrefixMappingType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component NsPrefixMapping XML schema details]*

Component Any

This element MAY hold a set of base64 encoded arbitrary data. To help the processing of the data it may be qualified by the mime type element.

Below follows a list of the sub-components that MAY be present within this component:

Non-normative Comment:

This component was introduced in DSS core version 1.0 and is used as a placeholder for arbitrary data. In version 1.0 there were different ways defined to represent the data, e.g. as inline XML, encapsulated XML or base64 encoded. The expansion of the scope to different syntaxes limits the options to base64 encoded data or attachments as represented in Base64Data. In this version the component Any does not use additional subcomponents.

Any – JSON Syntax

The AnyType JSON object SHALL implement in JSON syntax the requirements defined in the Any component.

The AnyType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dsb-AnyType" : {
  "type" : "object",
  "properties" : {
    "ID" : {
      "type" : "string"
    },
    "val" : {
      "type" : "string"
    },
    "attRef" : {
      "$ref" : "#/definitions/dsb-AttachmentReferenceType"
    },
    "mimeType" : {
      "type" : "string"
    },
    "idRef" : {
      "type" : "string"
    }
  }
}
```

[component Any JSON schema details]

Any – XML Syntax

The XML element is defined in the XML namespace 'http://docs.oasis-open.org/dss-x/ns/base' .The XML type AnyType SHALL implement the requirements defined in the Anycomponent.

The AnyType XML element is defined in XML Schema [DSBXSD], and is copied below for information.

```
<xs:complexType name="AnyType">
  <xs:complexContent>
    <xs:extension base="dsb:Base64DataType"/>
  </xs:complexContent>
</xs:complexType>
```

Each child element of AnyType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

Component Result

The Result element is returned with every response message.

Below follows a list of the sub-components that MAY be present within this component:

The ResultMajor element MUST contain one instance of a URI. Its value is limited to an item of the following set:

- urn:oasis:names:tc:dss:1.0:resultmajor:Success
- urn:oasis:names:tc:dss:1.0:resultmajor:RequesterError
- urn:oasis:names:tc:dss:1.0:resultmajor:ResponderError
- urn:oasis:names:tc:dss:1.0:resultmajor:InsufficientInformation
- urn:oasis:names:tc:dss:1.0:profiles:asynchronousprocessing:resultmajor:Pending

The ResultMajor element describes the most significant component of the result code. The set values MAY be extended.

The optional ResultMinor element MUST contain a URI.

The optional ResultMessage element MUST contain sub-component. A given element MUST satisfy the requirements specified in section InternationalStringType. *It represents a message which MAY be returned to an operator, logged by the client, used for debugging, etc.*

The optional ProblemReference element MUST contain a string. *In the case of processing problems the server may want to give a reference to processing details (e.g. for debugging purposes) but doesn't want to disclose sensitive information this element can be used. It may contain a random string that links the current request to internal logs, processing protocols or crash dumps.*

Non-normative Comment:

[component Result non normative details]

Result – JSON Syntax

The ResultType JSON object SHALL implement in JSON syntax the requirements defined in the Result component.

Properties of the JSON object SHALL implement the sub-components of ResultType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
ResultMajor	maj	*[]*
ResultMinor	min	*[]*
ResultMessage	msg	*[]*
ProblemReference	pRef	*[]*

The ResultType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dsb-ResultType" : {
  "type" : "object",
  "properties" : {
    "maj" : {
      "type" : "string",
      "enum" : ["urn:oasis:names:tc:dss:1.0:resultmajor:Success",
        "urn:oasis:names:tc:dss:1.0:resultmajor:RequesterError",
        "urn:oasis:names:tc:dss:1.0:resultmajor:ResponderError",
        "urn:oasis:names:tc:dss:1.0:resultmajor:InsufficientInformation",
        "urn:oasis:names:tc:dss:1.0:profiles:asynchronousprocessing:resultmajor:Pending"
    },
    "min" : {
      "type" : "string"
    },
    "msg" : {
      "$ref" : "#/definitions/dsb-InternationalStringType"
    },
    "pRef" : {
      "type" : "string"
    }
  },
  "required" : ["maj"]
}
```

[component Result JSON schema details]

Result – XML Syntax

The XML element is defined in the XML namespace 'http://docs.oasis-open.org/dss-x/ns/base' .The XML type ResultType SHALL implement the requirements defined in the Resultcomponent.

The ResultType XML element is defined in XML Schema [DSBXSD], and is copied below for information.

```
<xs:complexType name="ResultType">
  <xs:sequence>
    <xs:element name="ResultMajor">
      <xs:simpleType>
        <xs:restriction base="xs:anyURI">
          <xs:enumeration value="urn:oasis:names:tc:dss:1.0:resultmajor:Success"/>
          <xs:enumeration value="urn:oasis:names:tc:dss:1.0:resultmajor:RequesterError"/
          <xs:enumeration value="urn:oasis:names:tc:dss:1.0:resultmajor:ResponderError"/
          <xs:enumeration value="urn:oasis:names:tc:dss:1.0:resultmajor:InsufficientInfo
          <xs:enumeration value="urn:oasis:names:tc:dss:1.0:profiles:asynchronousprocess
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element minOccurs="0" name="ResultMinor" type="xs:anyURI"/>
    <xs:element minOccurs="0" name="ResultMessage" type="dsb:InternationalStringType"/>
    <xs:element minOccurs="0" name="ProblemReference" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Each child element of ResultType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component Result XML schema details]*

Component InternationalString

This element attaches an element to a human-readable string to specify the string's language.

Below follows a list of the sub-components that MAY be present within this component:

The value element MUST contain one instance of a string. *The human readable string. In non-XML representations the value element contains the textual content.*

The lang element MUST contain one instance of a ISO language descriptor. *This element identifies the language of the value element.*

Non-normative Comment:

[component InternationalString non normative details]

InternationalString – JSON Syntax

The InternationalStringType JSON object SHALL implement in JSON syntax the requirements defined in the InternationalString component.

Properties of the JSON object SHALL implement the sub-components of InternationalStringType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
value	value	*[]*
lang	lang	*[]*

The InternationalStringType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dsb-InternationalStringType" : {
  "type" : "object",
  "properties" : {
    "value" : {
      "type" : "string"
    },
    "lang" : {
      "type" : "string"
    }
  },
  "required" : ["lang"]
}
```

[component InternationalString JSON schema details]

InternationalString – XML Syntax

The XML element is defined in the XML namespace 'http://docs.oasis-open.org/dss-x/ns/base' .The XML type InternationalStringType SHALL implement the requirements defined in the InternationalStringcomponent.

The InternationalStringType XML element is defined in XML Schema [DSBXSD], and is copied below for information.

```
<xs:complexType name="InternationalStringType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute ref="xml:lang" use="required"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

Each child element of InternationalStringType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component InternationalString XML schema details]*

Component ResponseBase

The ResponseBase component is the base structure for response elements defined by the core protocol or profiles.

Below follows a list of the sub-components that MAY be present within this component:

The Result element MUST contain one instance of sub-component. This element MUST satisfy the requirements specified in section ResultType. *The Fehler! Verweisquelle konnte nicht gefunden werden. element represents the status of the request..*

The optional AppliedProfile element MAY occur zero or more times containing a URI. *This element lists the set of DSS profile applied by the server. This set MAY include the set of profiles requested by the client. But the server MAY use more comprehensive set of profiles and add additional profiles not requested by the client.*

The optional RequestID element MUST contain one instance of a string. *The RequestID element is used to correlate this response with its request.*

The optional ResponseID element MUST contain one instance of a string. *The ResponseID element*

Non-normative Comment:

[component ResponseBase non normative details]

ResponseBase – JSON Syntax

The ResponseBaseType JSON object SHALL implement in JSON syntax the requirements defined in the ResponseBase component.

Properties of the JSON object SHALL implement the sub-components of ResponseBaseType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
Result	result	*[]*
AppliedProfile	profile	*[]*
RequestID	reqID	*[]*
ResponseID	respID	*[]*

The ResponseBaseType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dsb-ResponseBaseType" : {
  "type" : "object",
  "properties" : {
    "result" : {
      "$ref" : "#/definitions/dsb-ResultType"
    },
    "profile" : {
      "type" : "array",
      "items" : {
        "type" : "string"
      }
    },
    "reqID" : {
      "type" : "string"
    },
    "respID" : {
      "type" : "string"
    }
  },
  "required" : ["result"]
}
```

[component ResponseBase JSON schema details]

ResponseBase – XML Syntax

The XML element is defined in the XML namespace 'http://docs.oasis-open.org/dss-x/ns/base' .The XML type ResponseBaseType SHALL implement the requirements defined in the ResponseBasecomponent.

The ResponseBaseType XML element is defined in XML Schema [DSBXSD], and is copied below for information.

```
<xs:complexType abstract="true" name="ResponseBaseType">
  <xs:sequence>
    <xs:element name="Result" type="dsb:ResultType"/>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="AppliedProfile" type="xs:anyUR
  </xs:sequence>
  <xs:attribute name="RequestID" type="xs:string" use="optional"/>
  <xs:attribute name="ResponseID" type="xs:string" use="optional"/>
</xs:complexType>
```

Each child element of ResponseBaseType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component ResponseBase XML schema details]*

Referenced from other documents

Component Transforms

This component reflects the structure 'Transforms' defined in the XMLDSig specification [CLAUSE FOR LINK TO THE XMLDSig SPEC]. This section provides the definition required to support the DSS-X 2.0 multi-syntax approach.

Below follows a list of the sub-components that MAY be present within this component:

The Transform element MUST occur 1 or more times containing sub-component. Each instance MUST satisfy the requirements specified in section TransformType. *[sub component Transform details]*

Non-normative Comment:

[component Transforms non normative details]

Transforms – JSON Syntax

The TransformsType JSON object SHALL implement in JSON syntax the requirements defined in the Transforms component.

Properties of the JSON object SHALL implement the sub-components of TransformsType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
Transform	transform	*[]*

The TransformsType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dsigrw-TransformsType" : {
  "type" : "object",
  "properties" : {
    "transform" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dsigrw-TransformType"
      }
    }
  },
  "required" : ["transform"]
}
```

[component Transforms JSON schema details]

Transforms – XML Syntax

The XML element is defined in the XML namespace 'http://docs.oasis-open.org/dss-x/ns/xmldsig/rewritten' .The XML type TransformsType SHALL implement the requirements defined in the Transformscomponent.

The TransformsType XML element is defined in XML Schema [DSIGRWXSD], and is copied below for information.

```
<complexType name="TransformsType">
  <sequence>
    <xs:element maxOccurs="unbounded" name="Transform" type="ds-rw:TransformType"/>
  </sequence>
</complexType>
```

Each child element of TransformsType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component Transforms XML schema details]*

Component Transform

This component reflects the structure 'Transforms' defined in the XMLDSig specification [CLAUSE FOR LINK TO THE XMLDSig SPEC]. This section provides the definition required to support the DSS-X 2.0 multi-syntax approach. See section 'Transforming DSS 1.0 into 2.0' for a detailed discussion of the applied changes.

Below follows a list of the sub-components that MAY be present within this component:

The optional value element MUST contain a string. *This string holds the text content part of a 'mixed' XML element.*

The optional Base64Content element MUST contain base64 encoded binary data.

The optional XPath element MAY occur zero or more times containing a string. *[sub component XPath details]*

The optional NsPrefixMapping element MAY occur zero or more times containing sub-component. If present each instance MUST satisfy the requirements specified in section NsPrefixMappingType. *This list has no direct correspondence in the XMLDSig schema definition. It is used to represent the XML namespace prefix definitions in other syntaxes than XML.*

The Algorithm element MUST contain one instance of a URI. *[sub component Algorithm details]*

Non-normative Comment:

[component Transform non normative details]

Transform – JSON Syntax

The TransformType JSON object SHALL implement in JSON syntax the requirements defined in the Transform component.

Properties of the JSON object SHALL implement the sub-components of TransformType using JSON-specific names mapped as shown in the table below.

Element	**Implementing JSON member name**	**Comments**
value	val	*[]*
Base64Content	b64Content	*[]*
XPath	xPath	*[]*
NsPrefixMapping	nsDecl	*[]*
Algorithm	alg	*[]*

The TransformType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"dsigrw-TransformType" : {
  "type" : "object",
  "properties" : {
    "xpath" : {
      "type" : "array",
      "items" : {
        "type" : "string"
      }
    },
    "val" : {
      "type" : "string"
    },
    "b64Content" : {
      "type" : "string"
    },
    "xPath" : {
      "type" : "array",
      "items" : {
        "type" : "string"
      }
    },
    "nsDecl" : {
      "type" : "array",
      "items" : {
        "$ref" : "#/definitions/dsb-NsPrefixMappingType"
      }
    },
    "alg" : {
      "type" : "string"
    }
  },
  "required" : ["alg"]
}
```

[component Transform JSON schema details]

Transform – XML Syntax

The XML element is defined in the XML namespace 'http://docs.oasis-open.org/dss-x/ns/xmldsig/rewritten' . The original definition of this element uses the 'mixed' content attribute. To support non-XML syntax using a common object model the attribute is dropped and a 'value' component is introduced. The XML type TransformType SHALL implement the requirements defined in the Transformcomponent.

The TransformType XML element is defined in XML Schema [DSIGRWXSD], and is copied below for information.

```
<xs:complexType name="TransformType">
  <xs:sequence>
    <xs:element maxOccurs="1" minOccurs="0" name="value" type="string"/>
    <xs:element maxOccurs="1" minOccurs="0" name="Base64Content" type="xs:base64Binary"/
    <xs:element maxOccurs="unbounded" minOccurs="0" name="XPath" type="string"/>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="NsPrefixMapping" type="dsb:NsP
  </xs:sequence>
  <xs:attribute name="Algorithm" type="xs:anyURI" use="required"/>
</xs:complexType>
```

Each child element of TransformType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. *[component Transform XML schema details]*

Component NameID

The NameID component is used when an element serves to represent an entity by a string-valued name. This component reflects the structure 'NameID' defined in the SAML2 specification [CLAUSE FOR LINK TO THE SAML2 SPEC]. This section provides the definition required to support the DSS-X 2.0 multi-syntax approach.

Below follows a list of the sub-components that MAY be present within this component:

The value element MUST contain one instance of a string. *In non-XML representations the value element contains the actual identifier*

The optional Format element MUST contain one instance of a URI. *The Format element represents the classification of string-based identifier information.*

The optional SPProvidedID element MUST contain one instance of a string. *The SPProvidedID element defines the alternative identifier of the principal most recently set by the service provider or affiliation, if any*

The optional NameQualifier element MUST contain one instance of a string. *The NameQualifier element contains the security or administrative domain that qualifies the name. This attribute provides a means to federate names from disparate user stores without collision.*

The optional SPNameQualifier element MUST contain one instance of a string. *The SPNameQualifier element further qualifies a name with the name of a service provider or affiliation of providers. This attribute provides an additional means to federate names on the basis of the relying party or parties.*

Non-normative Comment:

[component NameID non normative details]

NameID – JSON Syntax

The NameIDType JSON object SHALL implement in JSON syntax the requirements defined in the NameID component.

Properties of the JSON object SHALL implement the sub-components of NameIDType using JSON-specific names mapped as shown in the table below.

Element	Implementing JSON member name	Comments
value	value	*[]*
Format	format	*[]*
SPProvidedID	provId	*[]*
NameQualifier	nameQual	*[]*
SPNameQualifier	spNameQual	*[]*

The NameIDType JSON object is defined in the JSON schema [DSS2JSON] and is provided below as a service to the reader.

```
"saml2rw-NameIDType" : {
  "type" : "object",
  "properties" : {
    "spprovidedID" : {
      "type" : "string"
    },
    "spnameQualifier" : {
      "type" : "string"
    },
    "value" : {
      "type" : "string"
    },
    "format" : {
      "type" : "string"
    },
    "provId" : {
      "type" : "string"
    },
    "nameQual" : {
      "type" : "string"
    },
    "spNameQual" : {
      "type" : "string"
    }
  }
}
```

[component NameID JSON schema details]

NameID – XML Syntax

The XML element is defined in the XML namespace 'http://docs.oasis-open.org/dss/ns/saml2/rewritten' . The XML type NameIDType SHALL implement the requirements defined in the NameIDcomponent.

The NameIDType XML element is defined in XML Schema [SAML2RWXSD], and is copied below for information.

```
<complexType name="NameIDType">
  <simpleContent>
    <extension base="string">
      <attributeGroup ref="saml2-rw:IDNameQualifiers"/>
      <xs:attribute name="Format" type="anyURI" use="optional"/>
      <xs:attribute name="SPProvidedID" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>
```

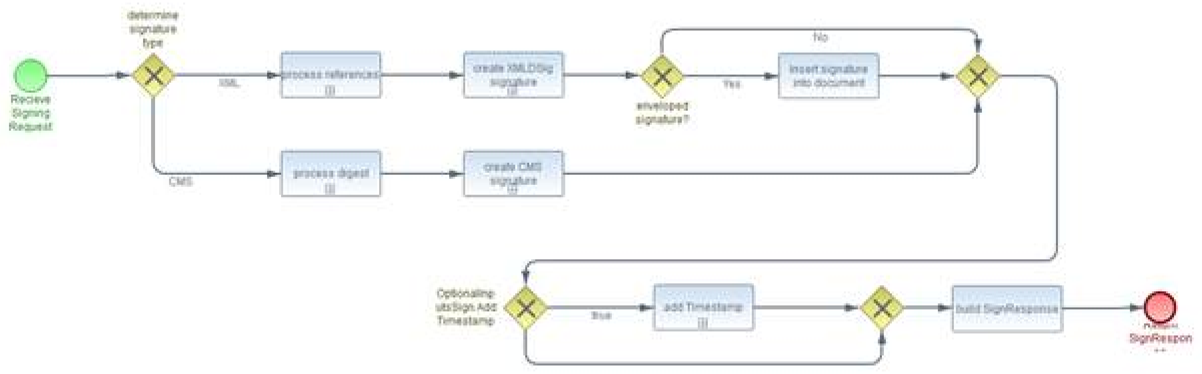
Each child element of NameIDType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name. The element 'value' is represented by the component's XML tag text content.*[component NameID XML schema details]*

Element / JSON name lookup tables The subsequent table allows to find the names of a component's element for a given JSON member name. JSON member name mapped from element name additionalTimeInfo AdditionalTimeInfo addKeyInfo AdditionalKeyInfo alg DigestMethod Algorithm attRef AttRef attURI AttRefURI aud IntendedAudience augSig AugmentedSignature b64Content Base64Content b64Data Base64Data b64Sig Base64Signature cert X509Certificate claimedIdentity ClaimedIdentity code Code createEnvelopedSignature CreateEnvelopedSignature createRef createReference crl X509CRL currTime CurrentTime di DigestInfo dis DigestInfos doc Document format Format hasObjectTagsAndAttributesSet HasObjectTagsAndAttributesSet ID Id id Identifier idRef IdRef incContent IncludeEContent incObj IncludeObject indeterminate IndeterminateDetail inDocs InputDocuments invalid InvalidDetail keySel KeySelector lowerBound LowerBoundary maj ResultMajor mimeType MimeType min ResultMinor msg ResultMessage Message name Name KeyName nonce Nonce nsDecl NsPrefixMapping objId ObjId ocsp OCSPResponse optInp OptionalInputs optOutp OptionalOutputs poe PoE pre NamespacePrefix pRef ProblemReference procDetails ProcessingDetails profile AppliedProfile prop Property props Properties provId SPProvidedID quality SignatureQualityLevel recipient Recipient ref Ref refId RefId refType RefType refURI RefURI reqID RequestID respID ResponseID result Result VerifyManifestResults ManifestResult returnAugmented ReturnAugmentedSignature returnProcDetails ReturnProcessingDetails returnSigner ReturnSignerIdentity returnSigningTime ReturnSigningTimeInfo returnTimestamped ReturnTimestampedSignature returnTransformed ReturnTransformedDocument returnVerificationTime ReturnVerificationTimeInfo schema Schema schemaRefs SchemaRefs sigAlgo SignatureAlgorithm signedProps SignedProperties signedRef SignedReference signedRefs SignedReferences signerIdentity SignerIdentity signingTime SigningTime signingTimeBounds SigningTimeBoundaries signingTimeInfo SigningTimeInfo sigObj SignatureObject sigPlacement SignaturePlacement sigPtr SignaturePtr sigType SignatureType ski X509SKI specTime SpecificTime status Status sub X509SubjectName suppInfo SupportingInfo timestampedSig TimestampedSignature transform Transform transforms Transforms type Type unsignedProps UnsignedProperties upperBound UpperBoundary uri NamespaceURI useVerificationTime UseVerificationTime val DigestValue Value value valid ValidDetail verificationTime VerificationTime verificationTimeInfo VerificationTimeInfo verifyManifests VerifyManifests whichDoc WhichDocument whichRef WhichReference x509Digest X509Digest xPath XPath ReferenceXpath xPathAfter XPathAfter xPathFirstChildOf XPathFirstChildOf The subsequent table allows to find the abbreviated JSON member names for a given element name. Element Implementing JSON member name AdditionalKeyInfo addKeyInfo AdditionalTimeInfo additionalTimeInfo Algorithm alg AppliedProfile profile AttRef attRef AttRefURI attURI AugmentedSignature augSig Base64Content b64Content Base64Data b64Data Base64Signature b64Sig ClaimedIdentity claimedIdentity Code code CreateEnvelopedSignature createEnvelopedSignature createReference createRef CurrentTime currTime DigestInfo di DigestInfos dis DigestMethod alg DigestValue val Document doc Format format HasObjectTagsAndAttributesSet hasObjectTagsAndAttributesSet Id ID Identifier id IdRef idRef IncludeEContent incContent IncludeObject incObj IndeterminateDetail indeterminate InputDocuments inDocs IntendedAudience aud InvalidDetail invalid KeyName name KeySelector keySel LowerBoundary lowerBound ManifestResult result Message msg MimeType mimeType Name name NamespacePrefix pre NamespaceURI uri Nonce nonce NsPrefixMapping nsDecl ObjId objId OCSPResponse ocsp OptionalInputs optInp OptionalOutputs optOutp PoE poe ProblemReference pRef ProcessingDetails procDetails Properties props Property prop Recipient recipient Ref ref ReferenceXpath xPath RefId refId RefType refType RefURI refURI RequestID reqID ResponseID respID Result result ResultMajor maj ResultMessage msg ResultMinor min ReturnAugmentedSignature returnAugmented ReturnProcessingDetails returnProcDetails ReturnSignerIdentity returnSigner ReturnSigningTimeInfo returnSigningTime ReturnTimestampedSignature returnTimestamped ReturnTransformedDocument returnTransformed ReturnVerificationTimeInfo returnVerificationTime Schema schema SchemaRefs schemaRefs

SignatureAlgorithm sigAlgo SignatureObject sigObj SignaturePlacement sigPlacement SignaturePtr sigPtr SignatureQualityLevel quality SignatureType sigType SignedProperties signedProps SignedReference signedRef SignedReferences signedRefs SignerIdentity signerIdentity SigningTime signingTime SigningTimeBoundaries signingTimeBounds SigningTimeInfo signingTimeInfo SpecificTime specTime SPProvidedID provId Status status SupportingInfo suppInfo TimestampedSignature timestampedSig Transform transform Transforms transforms Type type UnsignedProperties unsignedProps UpperBoundary upperBound UseVerificationTime useVerificationTime ValidDetail valid Value val value val VerificationTime verificationTime VerificationTimeInfo verificationTimeInfo VerifyManifestResults result VerifyManifests verifyManifests WhichDocument whichDoc WhichReference whichRef X509Certificate cert X509CRL crl X509Digest x509Digest X509SKI ski X509SubjectName sub XPath xPath XPathAfter xPathAfter XPathFirstChildOf xPathFirstChildOf 5   [Data Processing Model for Signing](#sec_DataProcessingModelForSigning)
==

The following process diagram illustrates the major buildings blocks of the processing of a signing request. The sub processes are described in the next chapters.

Figure 2: Signing Overview



The workflow splits into the sections for XMLDSig and CMS signature processing. The input component for a signing request is SignRequest (see section 4.2.6). The signature will be selected by the server considering a given SignatureType element of OptionalInputsSign and its configuration and policies. « Profiles MAY introduce additional signature types and thus MUST define the adequate processing steps. » [DSS-5-1]

If the element AddTimestamp of OptionalInputsSign is set to 'true' the sub-process 'add Timestamp' adds a timestamp to the signature.

The task of building the SignResponse component is shared between all signature formats.

5.1 Processing for XML Signatures

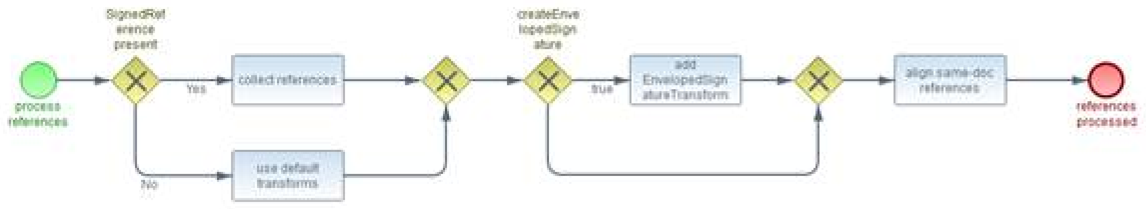
The first sub-process 'process references' of the XML signature creation is the processing of the references. The second sub-process handles the creation of the XML signature. These two sub-processes are described in detail below.

If the element CreateEnvelopedSignature of SignaturePlacement is set to true the signature will be inserted into the document and location selected by SignaturePlacement.

5.1.1 Sub process 'process references'

The following process diagram illustrates the processing steps for the assembly of references.

Figure 3: Process References



The input documents are read from the Base64Data element of the referred Document component into an octet stream. « This data MUST be a well-formed XML Document as defined in [XML] section 2.1. » [DSS-5.1.1-1]

If the optional input SignedReferences is present each SignedReference element controls the creation of a corresponding <ds:Reference>. The task ‘collect references’ handles the SignedReferences.

Otherwise there will be a <ds:Reference> element for each given input document. The set of transforms and their parameter will be selected by the server. The task ‘use default transforms’ select this set of <ds: Reference>.

Note: Transforms can be applied as a server implementation MAY choose to increase robustness of the Signatures created. These Transforms may reflect idiosyncrasies of different parsers or solve encoding issues or the like. Servers MAY choose not to apply transforms in basic processing and extract the binary data for direct hashing or canonicalize the data directly if certain optional inputs are not to be implemented.

If the element CreateEnvelopedSignature of SignaturePlacement is set to true the list of transforms will be prepended with an EnvelopedSignatureTransform entry. The task ‘add EnvelopedSignatureTransform’ processes the corresponding <ds:Reference>.

« The RefURI attribute of <ds:Reference> element MUST be set to include a “same-document” URI which references either:

· The whole Document containing the signature (by using a RefURI=””)

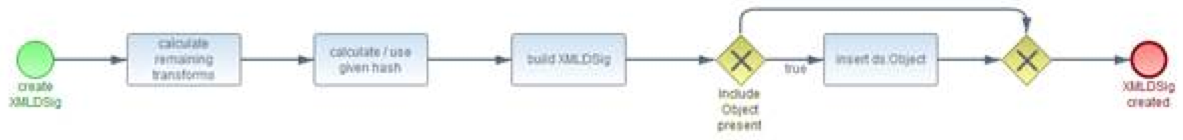
· The relevant parts of the Document to be covered/protected by the signature (by using a “same-document” RefURI attribute having a value starting with “#”, like RefURI=”#some-id”, RefURI=” #xpointer(/)”, RefURI=”#xpointer(/DocumentElement/ToBeSignedElement)” or the like). If the result of evaluating the expression included in the RefURI attribute doesn’t fit in any of the options described above, the server MUST reject the request using a ResultMajor RequesterError which MAY be qualified by a ResultMinor urn:oasis:names:tc:dss:1.0:resultminor:InvalidRefURI.

» [DSS-5.1.1-2]

This alignment will be performed by the task ‘align same-doc references’.

5.1.2 Sub process ‘create XML signature’

Figure 4: Create XML Signature



The first task ('calculate remaining transforms') of this section applies the given set of transforms. If a TransformedData element is provided by the client these calculations MUST be respected and just the remaining set of transforms must be processed by the server. « The case of a Document as base for a reference processing all transform steps MUST be applied. » [DSS-5.1.2-1]

Note: « As required in **[XMLDSIG]** if the end result is an XML node set, the server MUST attempt to convert the node set back into an octet stream using Canonical XML **[XML-C14N]**. » [DSS-5.1.2-2]

The 'calculate / use given hash' task computes the digest upon the transformation output. If a DocumentHash element is provided by the client the hash values are used as input for the following steps. The DocumentHash MAY contain digests of different algorithms. The server selects the appropriate hash algorithm.

Performing the task 'build XMLDSig' the server forms a set of <ds:Reference> with the elements and attributes set as follows:

· If the Document has a RefURI attribute, the <ds:Reference> element's URI attribute is set to the value of the RefURI attribute, else this attribute is omitted. « A signature MUST NOT be created if more than one RefURI is omitted in the set of input documents and the server MUST report a RequesterError by setting ResultMajor RequesterError qualified by a ResultMinor. » [DSS-5.1.2-3]

· If the Document has a RefType attribute, the <ds:Reference> element's Type attribute is set to the value of the RefType attribute, else this attribute is omitted.

· The <ds:DigestMethod> element is set to the hash method used.

· The <ds:DigestValue> element is set to the hash value that is to be calculated as per **[XMLDSIG]**.

· The <ds:Transforms> element is set to the sequence of transforms applied by the server in step b. « This sequence MUST describe the effective transform as a reproducible procedure from parsing until hash. » [DSS-5.1.2-4]

· « References resulting from processing of optional inputs MUST be included. » [DSS-5.1.2-5] In doing so, the server MAY reflect the ordering of the Document elements.

The server creates an XML signature using these <ds:Reference> elements according to the processing rules in **[XMLDSIG]**.

The last task 'insert ds:Object' handles the creation of an enveloping signature. If one or more optional input elements IncludeObject are present they will cause the inclusion of an object inside the signature being created.

5.1.2.1 XML Signatures Variant Optional Input IncludeObject

An enveloping signature is a signature having <ds:Object>s which are referenced by <ds:Reference>s having a same-document URI.

For each <IncludeObject> the server creates a new <ds:Object> element containing the document, as identified using the WhichDocument element, as its child. This object is carried within the enveloping signature. The ordering of the <IncludeObject> optional inputs MAY be ignored by the server.

« This <Document> MUST include a “same-document” RefURI attribute (having a value starting with “#”) which references either:

·   The whole newly-created <ds:Object>.

·   The relevant parts of the newly-created <ds:Object>’s contents to be covered/protected by the signature.

» [DSS-5.1.2.1-1] « f the result of evaluating the expression included in the RefURI element doesn’t fit in any of the options described above, the server MUST reject the request using a ResultMajor RequesterError which MAY be qualified by a ResultMinor

urn:oasis:names:tc:dss:1.0:resultminor:InvalidRefURI » [DSS-5.1.2.1-2]

Note: If the server does not support the ordering of <ds:Object>, it is recommended either to use ID-based referencing to the <ds:Object> (using the client-generated ID included in the ObjId attribute) or to rely on expressions based on <ds:Object>'s contents that allow to unambiguously refer to the included object or their relevant parts.

The URI in the RefURI element of this <Document> should at least reference the relevant parts of the Object to be included in the calculation for the corresponding reference. « Clients MUST generate requests in a way that some <ds:Reference>’s URI values actually will reference the <ds:Object> generated by the server once this element will have been included in the <ds:Signature> produced by the server. » [DSS-5.1.2.1-3]

« For each IncludeObject the server MUST carry out the following steps before performing Basic Processing:

1.   The server identifies the Document that is to be placed into a <ds:Object> as indicated by the WhichDocument element.

2.   The data to be carried in the enveloping signature is extracted and decoded.

3.   if the hasObjectTagsAndAttributesSet element is false or not present the server builds the <ds: Object> as follows:

a.   The server generates the new <ds:Object> and sets its Id attribute to the value indicated in ObjId element of the optional input if present.

b.   In the case of the Document pointed at by WhichDocument having Base64Data, <ds:Object>('s) MIME Type is to be set to the value of Base64Data('s) MIME Type value and the Encoding is to be set to http://www.w3.org/TR/xmlschema-2/#base64Binary

4.   The server splices the to-be-enveloped documents as <ds:Object>(s) into the <ds:Signature>, which is to be returned.

5.   If CreateReference is set to true generate a ds:Reference element referencing the spliced <ds: Object> and exclude this <Document> from the set of <Document>s ready for further processing. Otherwise just exclude this <Document> from the set of <Document>s ready for further processing.

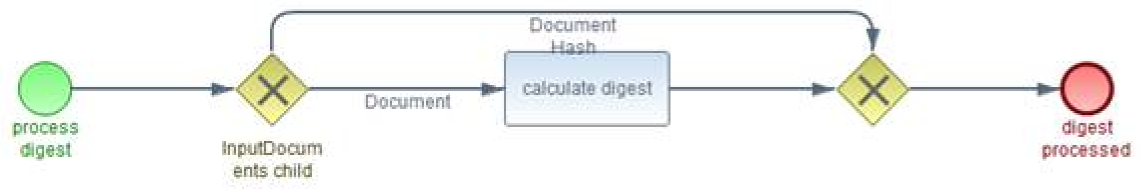
» [DSS-5.1.2.1-4]

5.2 Processing for CMS Signatures

5.2.1 Sub process ‘process digest’

The following process diagram illustrates the processing steps required to calculate the digest for a CMS signature.

Figure 5: Process Digest



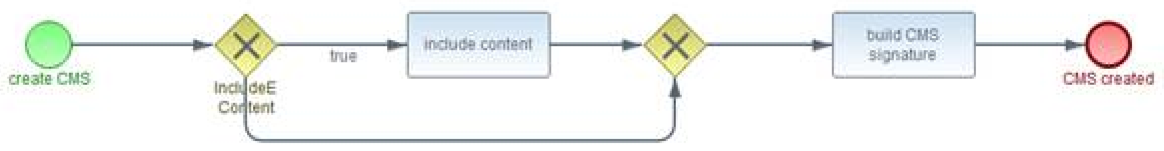
The SignRequest component MUST contain either a single Document (not having RefURI or RefType set) or a single DocumentHash component not having RefURI, RefType, Transforms.

If the InputDocuments component contains a Document element, the server hashes the octet stream represented by the Document. This is performed by the task ‘calculate digest’ If the InputDocuments component contains a DocumentHash element, the server uses the hash values as an input for the following steps. The DocumentHash MAY contain digests of different algorithms. The server selects the appropriate hash algorithm.

5.2.2 Sub process ‘create CMS signature’

The following process diagram illustrates the processing steps to create a CMS signature.

Figure 6: Create CMS signature



If the InputDocuments component contains a Document element and the IncludeEContent element of the OptionalInputsSign component is set to true then the task ‘include content’ creates a CMS structure with the document enveloped within the signature. For CMS details in this context please refer to [RFC 3852] sections 5.1 “SignedData Type” and 5.2 “EncapsulatedContentInfo Type”.

« Otherwise the resulting signature MUST be detached (aka. external or “without eContent”). » [DSS-5.2.2-1]

The following task ‘build CMS signature’ builds a SignedData structure containing the SignerInfo computed as follows:

The server forms a SignerInfo structure based on the input document. The components of the SignerInfo are set as follows:

· The digestAlgorithm field is set to the OID value for the hash method that was used in the previous processing step.

· The signedAttributes field’s message-digest attribute contains the hash value that was calculated / provided in previous processing step. Other signedAttributes MAY be added by the server, according to its profile or policy, or according to the Properties optional input.

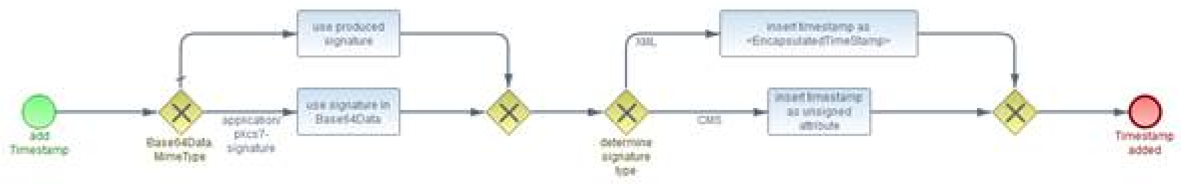
· The remaining fields (sid, signatureAlgorithm, and signature) are filled in as per a normal CMS signature.

5.3 General Timestamp Processing

5.3.1 Sub process ‘add Timestamp’

The following process diagram illustrates the processing steps to insert a timestamp.

Figure 7: Add Timestamp


add Timestamp
Base64Data MimeType
application/ pkcs7-signature
use produced signature
use signature in Base64Data
determine signature type
XML
CMS
insert timestamp as <EncapsulatedTimeStamp>
insert timestamp as unsigned attribute
Timestamp added

The AddTimestamp element of OptionalInputsSign indicates that the client wishes the server to embed a timestamp token as a property or attribute of the resultant or the supplied signature. The timestamp token will be applied to the signature value in the case of CMS/PKCS7 signatures or the <ds:SignatureValue> element in the case of XML signatures.

Note: Procedures for handling other forms of timestamp may be defined in profiles of the Core. In particular, the DSS AdES profile [DSS-AdES-P] defines procedures for generating timestamps over the content which is about to be signed (sometimes called content timestamps), and the DSS Timestamp profile [DSS-TS-P] defines procedures for handling standalone timestamps.

The Type element, if present, indicates what type of timestamp to apply. « Profiles that use this optional input MUST define the allowed values, and the default value, for the Type attribute (unless only a single type of timestamp is supported, in which case the Type attribute can be omitted). » [DSS-5.3.1-1]

Two scenarios for the timestamping of both CMS and XML signatures are supported by this Optional Input. They are as follows:

· Create and embed a timestamp token into the signature being created as part of this SignRequest.

· Create and embed a timestamp token into an existing signature, without verification, which is passed in the InputDocuments element of this SignRequest.

The following subsections specify the use of RFC 3161 timestamps with CMS signatures and the use of XML Timestamps or RFC 3161 timestamps with XML Signature. These subsections address both scenarios.

Note: The server SHOULD not verify the incoming signature before adding the timestamp. If a client wishes that its signatures be verified as a condition of time stamping, the client SHOULD use the AddTimestamp optional input of the Verify protocol.

5.3.1.1 Processing for CMS signatures time-stamping

If the MimeType element of the Base64Data component is set to ‘application/pkcs7-signature’ a timestamp token is created and embedded into the existing signature, without verification, which is passed in the InputDocuments component of this SignRequest. Otherwise a timestamp token is created and embedded into the signature being created as part of the processing of this SignRequest.

« In both scenarios, the timestamp token created by the server SHALL be created according to [RFC 3161]. » [DSS-5.3.1.1-1] The MessageImprint field within the TstInfo structure of the timestamp token will be derived from the signature value of the just-created or incoming signature depending on the scenario. « The timestamp SHALL be embedded in the CMS signature as an unsigned attribute with the object identifier (see Appendix A of [RFC 3161]):

{ iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 14} » [DSS-5.3.1.1-2]

The signature and its embedded timestamp is returned in the SignatureObject element of the SignResponse component.

5.3.1.2 Processing for XML Timestamps on XML signatures

« If the type attribute in the optional input AddTimestamp is urn:oasis:names:tc:dss:1.0:core:schema: XMLTimeStampToken

and signature being timestamped is an XML signature, then the XML signature MUST contain <dss: timestamp> as defined in [DSS1Core] section 5.1, placed in a <xades:XMLTimestamp> within a <xades: SignatureTimeStamp> as defined in [XAdES]. » [DSS-5.3.1.2-1]

« The <dss:timestamp> MUST contain <ds:Signature> with at least two <ds:Reference> elements:

· One with the Type attribute set to urn:oasis:names:tc:dss:1.0:core:schema:XMLTimeStampToken and referencing a <ds:Object> element whose content is a <TSTInfo> element.

· The other referencing the <ds:SignatureValue> being timestamped.

» [DSS-5.3.1.2-2]

The present specification defines a format for XML timestamp tokens. In addition, XAdES defines a mechanism for incorporating signature timestamps in XML signatures. « The present document mandates that signature timestamps in XML format MUST follow the syntax defined in [DSS1Core] section 5.1. » [DSS-5.3.1.2-3] « These time-stamp tokens MUST be added to XML signatures as specified by XAdES. » [DSS-5.3.1.2-4]

« The signature and its embedded timestamp SHALL be returned in the <SignatureObject> of the <SignResponse>. » [DSS-5.3.1.2-5]

5.3.1.3 Processing for RFC 3161 Timestamps on XML signatures

« If the type attribute in this optional input AddTimestamp is

urn:ietf:rfc:3161

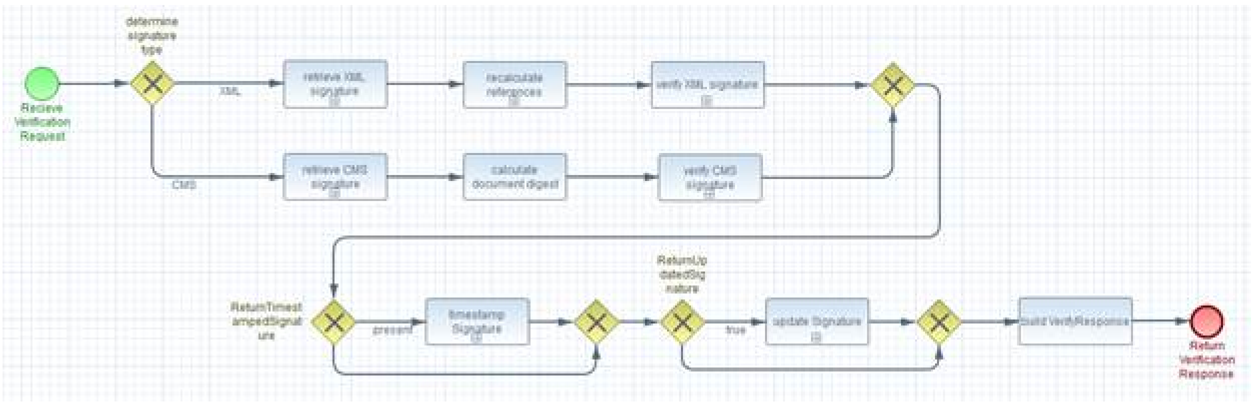
and signature being timestamped is an XML signature then the XML signature MUST contain an RFC 3161, placed in a <xades:EncapsulatedTimeStamp> within a <xades:SignatureTimeStamp> as defined in [XAdES]. » [DSS-5.3.1.3-1]

6 Data Processing Model for Verification

A DSS server that verifies XML signatures SHOULD perform the following steps, upon receiving a verification with the top-level component VerifyRequest (see section 4.2.10). These steps may be changed or overridden by the optional inputs, or by the profile or policy the server is operating under. The results of the vewrification process are return to the caller within a VerifyResponse component (see section 4.2.11). For more details on multi-signature verification, see section 6.1.1.1 [Multi-Signature Verification](#).

The following process diagram illustrates the major buildings blocks of the processing of a verification request. The sub processes are described in the following sub-chapters.

Figure 8: Verification Overview



The workflow splits into the sections for XMLDSig and CMS signature processing. « The processing path will be selected by the server considering a given SignatureType element of OptionalInputsVerify and its configuration and policies. » [DSS-6-1] Profiles may introduce additional signature types and MUST define the adequate processing steps.

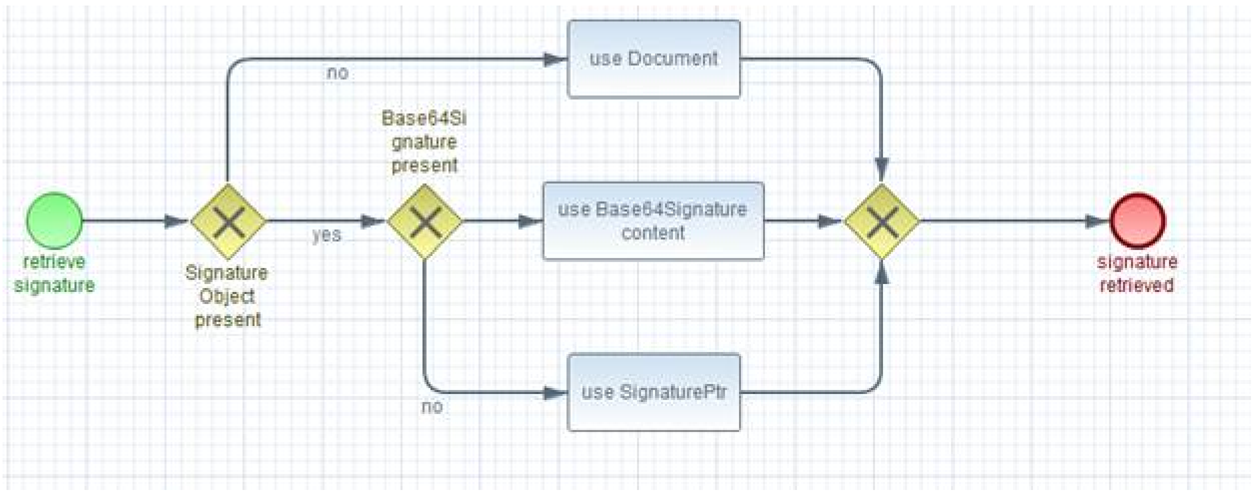
· If the element ReturnTimestampedSignature of OptionalInputsVerify is present, the sub-process 'timestamp Signature' adds a timestamp to the signature.

· If the element ReturnUpdatedSignature of OptionalInputsVerify is 'true' the sub-process 'update Signature' inserts the updated signature into the OptionalOutputsVerify.

6.1 [Processing for XML Signature Verification](#)

6.1.1 Sub process 'retrieve XML signature'

Figure 9: Retrieve XML Signature



The server retrieves one or more <ds:Signature> objects as follows:

· If the SignatureObject is present, the server retrieves either the <ds:Signature> that is a child element of the SignatureObject (see: Note at the end of this section), or those <ds:Signature> objects which are pointed to by the SignaturePtr in the SignatureObject.

· If the SignaturePtr points to an input document but not a specific element in that document, the pointed-to input document must be a Document element containing XML.

· « If the SignatureObject is omitted, there MUST be only a single Document element. » [DSS-6.1.1-1] This case is handled as if a SignaturePtr pointing to the single Document was present: the server will search and find every <ds:Signature> element in this input document and verify each <ds:Signature> according to the steps below.

6.1.1.1 Multi-Signature Verification

« If a client requests verification of an entire input document, either using a SignaturePtr without an XPath or a missing SignaturePtr (see section 6.1 step 1), then the server MUST determine whether the input document contains zero, one, or more than one <ds:Signature> elements. » [DSS-6.1.1.1-1] If zero, the server SHOULD return a ResultMajor code of RequesterError.

« If more than one <ds:Signature> elements are present, the server MUST either reject the request with a ResultMajor code of RequesterError and a ResultMinor code of NotSupported, or accept the request and try to verify all of the signatures. » [DSS-6.1.1.1-2]

If the server accepts the request in the multi-signature case (or if only a single signature is present) and one of the signatures fails to verify, the server should return one of the error codes in section 6.2, reflecting the first error encountered.

If all of the signatures verify correctly, the server should return the Success ResultMajor code and the following ResultMinor code:

urn:oasis:names:tc:dss:1.0:resultminor:ValidMultiSignatures

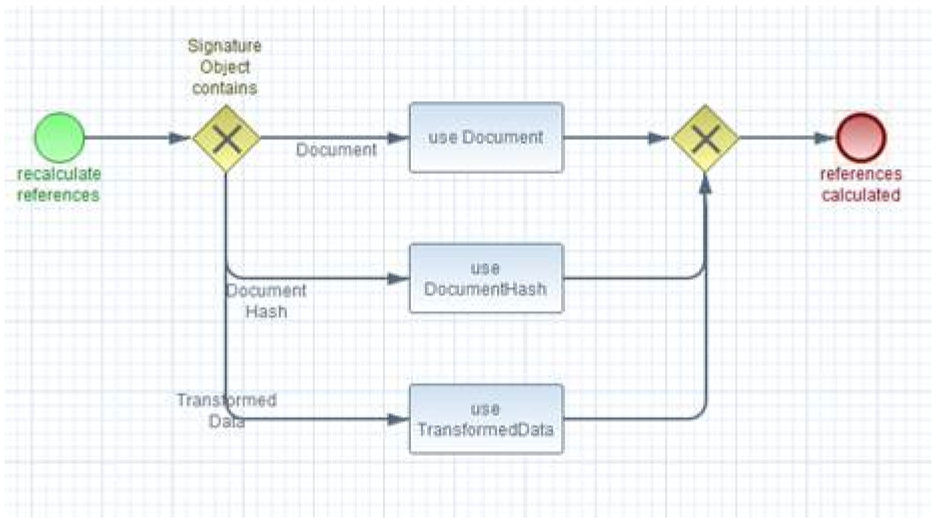
Non-normative Note:

These procedures only define procedures for handling of multiple signatures on one input document. The procedures for handling multiple signatures on multiple documents are not defined in this core specification, but however such procedures, along with any optional elements that may be required, may be defined in profiles of this specification.

Only certain optional inputs and outputs are allowed when performing multi-signature verification. See section 5.3 General processing for details.

6.1.2 Sub process 'recalculate references'

Figure 10: Recalculate References



For each <ds:Reference> in the <ds:Signature>, the server finds the input document with matching RefURI and RefType values (omitted attributes match omitted attributes).

· If the <ds:Reference> uses a same-document URI, the XPointer should be evaluated against the input document the <ds:Signature> is contained within, or against the <ds:Signature> itself if it is contained within the SignatureObject element.

· The SchemaRef element or optional input Schema of the input document or SignatureObject will be used, if present, to identify ID attributes when evaluating the XPointer expression.

· If the <ds:Reference> uses an external URI and the corresponding input document is not present, the server will skip the <ds:Reference>, and later return a result code such as ReferencedDocumentNotPresent to indicate this. The RefURI MAY be omitted in at most one of the set of Input documents.

· If the input document is a Document, the server extracts and decodes as described in 4.2.3 Component Document onwards depending of the form of the input document).

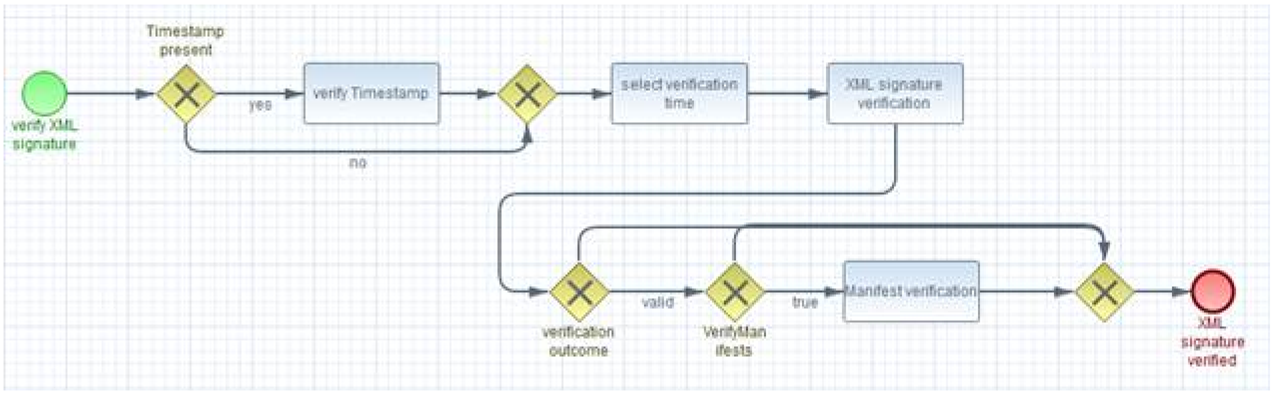
· If the input document is a TransformedData, the server MAY check that the <ds:Transforms> (if supplied) match between the TransformedData and the <ds:Reference> and then hashes the resultant data object according to <ds:DigestMethod>. « The server MUST check that the result matches <ds: DigestValue>. » [DSS-6.1.2-1]

· If the input document is a DocumentHash, the server MAY check that the <ds:Transforms>, <ds: DigestMethod> (if supplied) and <ds:DigestValue> elements match between the DocumentHash and the <ds:Reference>.

· « If the combination of RefURI and RefType matches more than one input document all of them MUST be either a TransformedData or a DocumentHash otherwise a RequesterError is issued qualified by result minor of ReferencedDocumentNotPresent. » [DSS-6.1.2-2] Only one of them is allowed to have a WhichReference value that matches the order of the <ds:Reference> within the <ds:SignedInfo> in question otherwise a RequesterError is issued qualified by result minor of ReferencedDocumentNotPresent.

6.1.3 Sub process ‘verify XML signature’

Figure 11: Verify XML Signature



« If one or more timestamps are present on the given signature this / these timestamps MUST be verified. » [DSS-6.1.3-1] The 'time of existence' asserted by the timestamp MAY be used to decide the verification time. For details see section 6.1.3.1 and 6.1.3.2 .

The server verifies the validity of the signature at a particular time (i.e. current time, assumed signing time or other time), depending on the server policy. This behavior MAY be altered by using the optional input UseVerificationTime.

If the VerifyManifests element of OptionalInputsVerify is set to 'true' the server validates the manifests in an XML signature. In accordance with [XMLDSIG] section 5.1, DSS Manifest validation does not affect a signature's core validation. The results of verifying individual <ds:Reference>'s within a <ds: Manifest> are returned in the VerifyManifestResults within the OptionalOutputsVerify. If the optional input VerifyManifests is set to 'true' and the XMLSig core validation succeeds, then the returned ResultMinor is

urn:oasis:names:tc:dss:1.0:resultminor:valid:hasManifestResults

In case of a negative XMLSig core validation no attempt is made to verify manifests.

If the signature validates correctly, the server returns one of the first three ResultMinor codes listed in section 6.2, depending on the relationship of the signature to the input documents (not including the relationship of the signature to those XML elements that were resolved through XPointer evaluation; the client will have to inspect those relationships manually). If the signature fails to validate correctly, the server returns some other code; either one defined in section 6.2 of this specification, or one defined by some profile of this specification.

6.1.3.1 Processing for RFC 3161 timestamp tokens on XML Signatures

The present section describes the processing rules for verifying an RFC 3161 timestamp token embedded within an XML signature as an unsigned property. This XML signature may be passed in on a Verify call within the SignatureObject or embedded within a Document's child.

The server shall verify the timestamp token performing the steps detailed below. If any one of them results in failure, then the timestamp token SHOULD be rejected.

1. Extract the timestamp token embedded in the incoming signature as defined in 5.3.1.2 Processing for XML Timestamps on XML signatures.

2. Verify that the token's public verification certificate is authorized for time stamping by examining the Extended Key Usage field for the presence of the time stamping OID "1.3.6.1.5.5.7.3.8".

3. Process the signature timestamp as defined in **[XAdES]** Annex G.2.2.16.1.3.

4. Verify that the public verification certificate conforms to all relevant aspects of the relying-party's policy including algorithm usage, policy OIDs, time accuracy tolerances, and the Nonce value.

5. Set the Result element as appropriate. urn:oasis:names:tc:dss:1.0:resultminor:valid:signature: InvalidSignatureTimestamp MAY be used to indicate that the signature is valid but the timestamp against that signature is invalid.

6.1.3.2 Processing for XML timestamp tokens on XML signatures

The present section describes the processing rules for verifying and XML Signature timestamp token embedded within an XML signature using the incorporation mechanisms specified in XAdES (i.e., in the <xades:XMLTimeStamp> <xades:SignatureTimeStamp> element's child). This XML signature may be passed in on a Verify call within the SignatureObject or embedded within a Document's child.

The server shall verify the timestamp token performing the steps detailed below. If any one of them results in failure, then the timestamp token SHOULD be rejected.

1. Extract the timestamp token embedded in the incoming signature as defined in 5.3.1.2 Processing for XML Timestamps on XML signatures.

2. Verify that the verification key and algorithms used conforms to all relevant aspects of the applicable policy. Should this key come within a public certificate, verify that the certificate conforms to all relevant aspects of the applicable policy including algorithm usage, policy OIDs, and time accuracy tolerances.

3. Verify that the aforementioned verification key is consistent with the ds:SignedInfo /SignatureMethod/@Algorithm attribute value.

4. Verify the timestamp token signature in accordance with the rules defined in **[XMLDSIG]**.

5. Verify that the <ds:SignedInfo> element contains at least two <ds:Reference> elements.

6. Verify that one of the <ds:Reference> elements has its Type attribute set to "urn:oasis:names: tc:dss:1.0:core:schema:XMLTimeStampToken". Take this one and proceed as indicated below:

a. Retrieve the referenced data object. Verify that it references a <ds:Object> element, which in turn envelopes a <TSTInfo> element.

b. Verify that the <TSTInfo> element has a valid layout as per the present specification.

c. Extract the digest value and associated algorithm from its <ds:DigestValue> and <ds: DigestMethod> elements respectively.

d. Recalculate the digest of the retrieved data object as specified by **[XMLDSIG]**with the digest algorithm indicated in <ds:DigestMethod>, and compare this result with the contents of <ds: DigestValue>.

7. Take each of the other <ds:Reference> elements and for each validate the hash as specified in **[XMLDSIG]**.

8. Check that for one of the <ds:Reference> elements the retrieved data object is actually the <ds: SignatureValue> element and that it contains its digest after canonicalization.

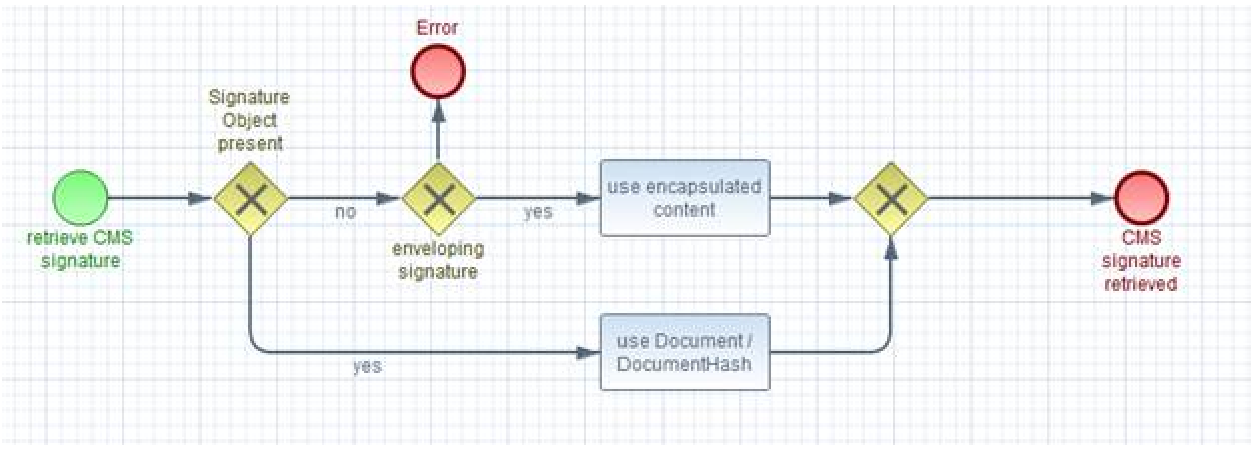
9. Set the Result element as appropriate. Minor Error urn:oasis:names:tc:dss:1.0:resultminor: valid:signature:InvalidSignatureTimestamp MAY be used to indicate that the signature is valid but the timestamp against that signature is invalid.

6.2 Processing for CMS Signature Verification

A DSS server that verifies CMS signatures SHOULD perform the following steps, upon receiving a VerifyRequest. These steps may be changed or overridden by the optional inputs, or by the profile or policy the server is operating under.

6.2.1 Sub process 'retrieve CMS signature'

Figure 12: Retrieve CMS Signature



1. The server retrieves the CMS signature by decoding the Base64Signature child of SignatureObject.

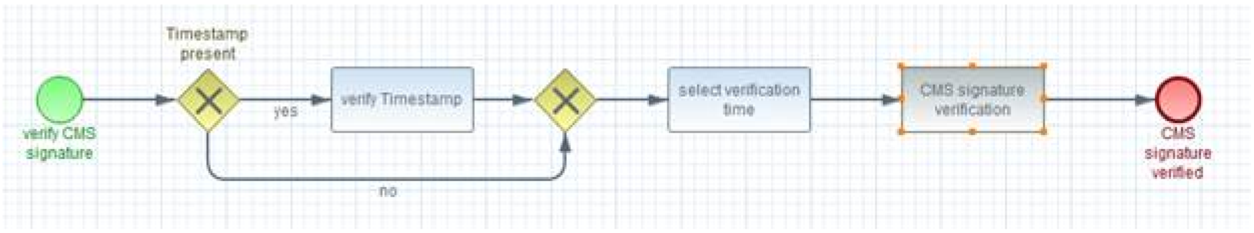
2. The server retrieves the input data. « If the CMS signature is detached, there MUST be a single input document: i.e. a single Document or DocumentHash element. » [DSS-6.2.1-1] « Otherwise, if the CMS signature is enveloping, it contains its own input data and there MUST NOT be any input documents present. » [DSS-6.2.1-2]

3. The CMS signature and input data are verified in the conventional way (see **[RFC 5652]** for details).

4. If the signature validates correctly, the server returns the first ResultMinor code listed in section 6.2. If the signature fails to validate correctly, the server returns some other code; either one defined in section 6.2 of this specification, or one defined by some profile of this specification.

6.2.2 Sub process 'verify CMS signature'

Figure 13: Verify CMS Signature



« If one or more timestamps are present on the given signature this / these timestamps MUST be verified. » [DSS-6.2.2-1] The 'time of existence' asserted by the timestamp MAY be used to decide the verification time. For details see section 6.2.2.1 .

The server verifies the validity of the signature at a particular time (i.e. current time, assumed signing time or other time), depending on the server policy. This behavior MAY be altered by using the optional input UseVerificationTime.

If the signature validates correctly, the server returns one of the first three ResultMinor codes listed in section 4.1.7 Component Result. If the signature fails to validate correctly, the server returns some other code; either one defined in section 4.1.7 Component Result of this specification, or one defined by some profile of this specification.

6.2.2.1 Processing for RFC 3161 Timestamp tokens on CMS Signatures.

The present section describes the processing rules for verifying a CMS RFC3161 timestamp token passed in on a Verify call within the SignatureObject of the VerifyRequest element. In the CMS case, since the "signature timestamp" is embedded in the signature as an unsigned attribute, only the time stamped signature is required for verification processing. As such, no additional input is required.

The processing by the server is broken down into the following steps:

1. The signature timestamp is embedded in the incoming signature as an unsigned attribute whose object identifier is 1.2.840.11359.1.9.16.2.14. Extract and verify the timestamp token.

2. Verify that the token's public verification certificate is authorized for time stamping by examining the Extended Key Usage field for the presence of the time stamping OID "1.3.6.1.5.5.7.3.8".

3. Validate that the TstInfo structure has a valid layout as defined in **[RFC 3161]**.

4. Extract the MessageImprint hash value and associated algorithm from the TstInfo structure which will be compared against the hash value derived in the next step.

5. Recalculate the hash of the signature value field of the signature in which the timestamp is embedded.

6. Compare the hash values from the two previous steps, and if they are equivalent, then this timestamp is valid for the signature that was time stamped.

7. Verify that the public verification certificate conforms to all relevant aspects of the relying-party's policy including algorithm usage, policy OIDs, time accuracy tolerances, and the Nonce value.

8. Set the Result element as defined in this specification. Minor Error urn:oasis:names:tc:dss:1.0:resultminor:valid:signature:InvalidSignatureTimestamp MAY be used to indicate that the signature is valid but the timestamp against that signature is invalid.

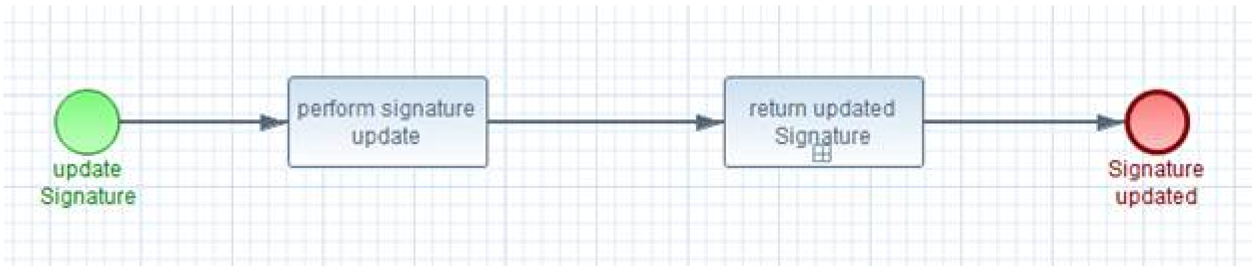
6.3 General Processing

The following steps are shared between all signature types.

6.3.1 Sub process 'update Signature'

The presence of the ReturnUpdatedSignature element of OptionalInputsVerify instructs the server to return an UpdatedSignature output, containing a new or updated signature.

Figure 14: Update Signature

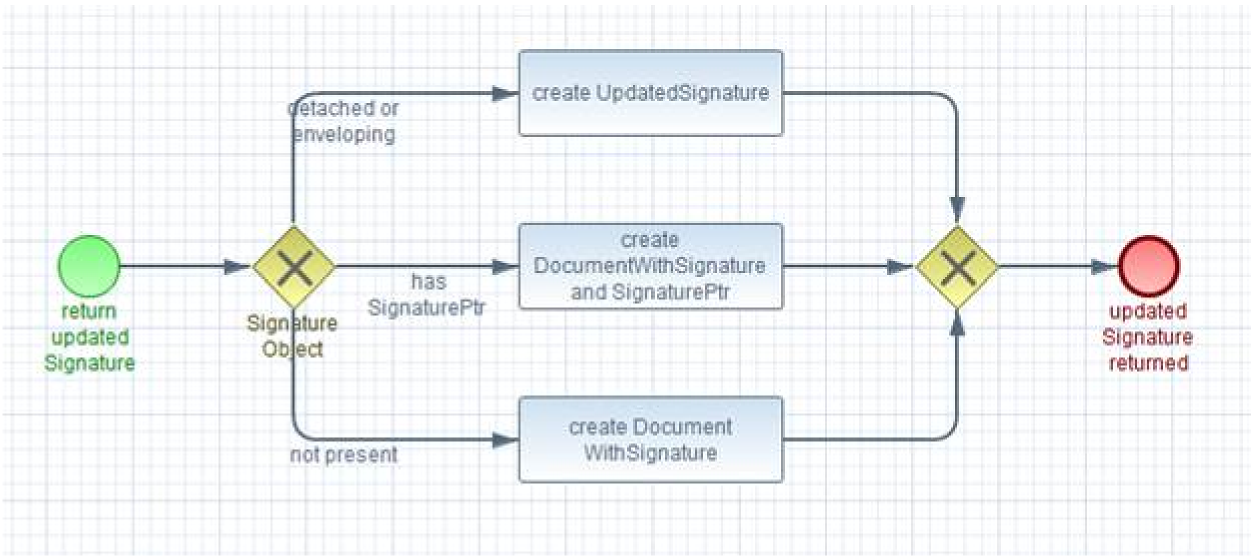


The Type element of ReturnUpdatedSignature defines the process of “updating” a signature. For example, the updated signature may be the original signature with some additional unsigned signature properties added to it (such as timestamps, counter-signatures, or additional information for use in verification), or the updated signature could be an entirely new signature calculated on the same input documents as the input signature. « Profiles that use this optional input MUST define the allowed values and their semantics, and the default value of ReturnUpdatedSignature (unless only a single type of updated signature is supported, in which case the element can be omitted). » [DSS-6.3.1-1]

Multiple occurrences of this optional input can be present in a single verify request message. « If multiple occurrences are present, each occurrence MUST have a different value. » [DSS-6.3.1-2] Each occurrence will generate a corresponding UpdatedSignature optional output. « These optional outputs SHALL be distinguishable based on their Type element, which will match each output with an input. » [DSS-6.3.1-3]

A DSS server SHOULD perform the following steps to return the updated signature appropriately. These steps may be changed or overridden by a profile or policy the server is operating under. (e.g. for PDF documents enveloping CMS signatures).

Figure 15: Select Update Target



· « If the detached or enveloping signature to be verified and updated appears within a Base64Signature then the UpdatedSignature optional output MUST contain the modified SignatureObject with the updated signature. » [DSS-6.3.1-4]

· « If the signature to be verified and updated is enveloped, and if the VerifyRequest contains a SignatureObject with a SignaturePtr pointing to an InputDocument enveloping the signature then the server MUST produce the following TWO optional outputs, first a DocumentWithSignature optional output containing the document that envelopes the updated signature, second an UpdatedSignature optional output containing a SignatureObject having a SignaturePtr element that MUST point to the former DocumentWithSignature. » [DSS-6.3.1-5]

· « If there is no SignatureObject included in the request then the server MUST produce a DocumentWithSignature optional output containing the document with the updated signature, only. » [DSS-6.3.1-6] No UpdatedSignature element will be generated.

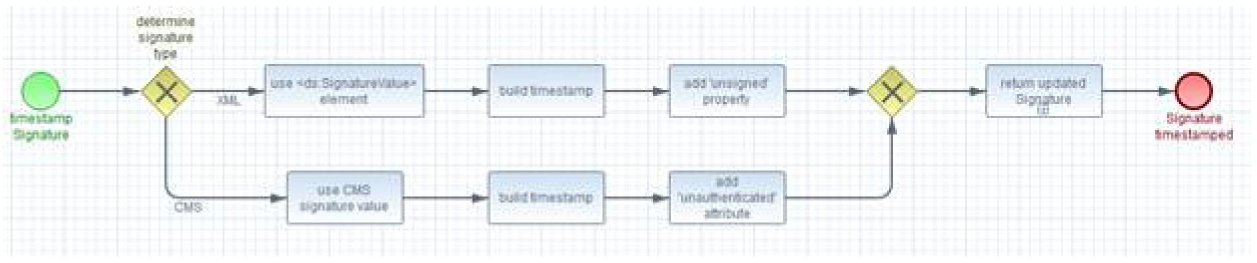
If created the DocumentWithSignature optional output contains the input document with the given signature inserted. The server places the signature in the document identified using the SignatureObject / SignaturePtr / WhichDocument element. « This Document MUST include a same-document RefURI element which references the data updated (e.g of the form RefURI). » [DSS-6.3.1-7]

6.3.2 Sub process ‘timestamp Signature’

If the ReturnTimestampedSignature element of OptionalInputsVerify is present the server updates the signature after its verification by embedding a signature timestamp token as an unauthenticated attribute (see "unauthAttrs" in section 9.1 [**RFC 5652**]) or *unsigned* property (see section 6.2.5 "The UnsignedSignatureProperties element" and section 7.3 "The SignatureTimeStamp element" [XAdES]) of the supplied signature.

The timestamp token will be on the signature value in the case of CMS/PKCS7signatures or the <ds: SignatureValue> element in the case of XML signatures.

Figure 16: Timestamp Signature



The Type element of ReturnTimestampedSignature, if present, indicates what type of timestamp to build. This document defines two values for it, namely:

· urn:ietf:rfc:3161 for generating a RFC 3161 timestamp token on the signature

· urn:oasis:names:tc:dss:1.0:core:schema:XMLTimeStampToken, for generating a XML timestamp token as defined in section 5.3 General Timestamp Processing of this document.

« Profiles that use this optional input MUST define the allowed values and the default value for the Type element (unless only a single type of timestamp is supported, in which case Type can be omitted). » [DSS-6.3.2-1]

The sub process of returning the updated signatures is the same as described in the sub process ‘update Signature’ (see section 6.3.1).

Note: Procedures for handling other forms of timestamp may be defined in profiles of the Core. In particular, the DSS XAdES profile [DSS-XAdES-P] defines procedures for handling timestamps against the document being signed, and the DSS Timestamp profile [DSS-TS-P] defines procedures for handling standalone timestamps.

6.3.3 Task ‘build VerifyResponse’

The task of building the VerifyResponse is shared between all signature formats. The OptionalInputsVerify element, server configuration and applied policies may affect the set of elements included in the OptionalOutputsVerify.

If the ReturnVerificationTimeInfo element of OptionalInputsVerify is set to ‘true’ the server returns the VerificationTimeInfo within the OptionalOutputsVerify. It contains the verification time and optionally other relevant time instants that may have been used when determining the verification time or that may be useful for its qualification.

If the ReturnSigningTimeInfo element of OptionalInputsVerify is set to ‘true’ the server returns the SigningTimeInfo within the OptionalOutputsVerify. It allows the client to obtain the time instant associated to the signature creation. Depending on the applicable server policy, this signing time needs to be qualified, in order to avoid unacceptable measurement errors or false claims, using time boundaries associated to trustworthy time values (based on timestamps or time-marks created using trusted time sources). In this case, the server MAY include these values in the LowerBoundary and UpperBoundary elements, respectively.

Criteria for determining when a time instant can be considered trustworthy and for determining the maximum acceptable delays between the signing time and their boundaries (if any) is outside the scope of this specification.

« When there's no way for the server to determine the signing time, the server MUST omit the SigningTimeInfo output. » [DSS-6.3.3-1]

If the ReturnSignerIdentity element of OptionalInputsVerify is set to 'true' the server returns the SignerIdentity element within the OptionalOutputsVerify. The SignerIdentity optional output contains an indication of who performed the signature. This option is not allowed in multi-signature verification.

If the ReturnTransformedDocument element of OptionalInputsVerify is present the server returns an input document to which the XML signature transforms specified by a particular <ds:Reference> have been applied. The <ds:Reference> is indicated by the zero-based WhichReference element (0 means the first <ds:Reference> in the signature, 1 means the second, and so on). Multiple occurrences of this optional input can be present in a single verify request message. Each occurrence will generate a corresponding optional output. These options are not allowed in multi-signature verification.

The TransformedDocument element within the OptionalOutputsVerify contains a document corresponding to the specified <ds:Reference> after all the transforms in the reference have been applied. In other words, the hash value of the returned document should equal the <ds:Reference> element's <ds: DigestValue>. To match outputs to inputs, each TransformedDocument component will contain a WhichReference element which matches the corresponding ReturnTransformedDocument optional input element.

If the ReturnProcessingDetails element of OptionalInputsVerify is set to 'true' the server returns the ProcessingDetails element within the OptionalOutputsVerify. The ProcessingDetails element elaborates on what signature verification steps succeeded or failed. This option is not allowed in multi-signature verification.

7 Asynchronous Processing Model

The main functionality of the 'Asynchronous Processing Profile' [DSSAsync] is included in this version of the core.

The server MAY decide that the processing of a request cannot be performed within a reasonable timeframe and therefore return an instance of the 'Component ResponseBase' with the ResultMajor value of

urn:oasis:names:tc:dss:1.0:profiles:asynchronousprocessing:resultmajor:Pending

and the ResponseID element set to a server generated value.

The client MAY initiate a request with the top-level component PendingRequest (see section 4.2.12) from time to time with the ResponseID of the initial response included in the RequestID element.

If the server is still not able to return the requested response, it will respond with a ResultMajor of ‘Pending’ again. When the server finally succeeds with its processing the results will be delivered to the client with its next polling call. « In this case the ResultMajor value MUST NOT be ‘Pending’ but the ResultMajor resulting from the request processing. » [DSS-7-1]

Clients not able to perform the Asynchronous Processing Model MAY treat the ‘Pending’ response as an error or try to perform the operation at later point in time. Even without referring to the ResponseID the server MAY be able to respond with full result immediately.

8 DSS Core Bindings

Mappings from DSS messages into standard communications protocols are called DSS *bindings*. *Transport bindings* specify how DSS messages are encoded and carried over some lower-level transport protocol. *Security bindings* specify how confidentiality, authentication, and integrity can be achieved for DSS messages in the context of some transport binding. Below we specify an initial set of bindings for DSS. Future bindings may be introduced by the OASIS DSS TC or by other parties.

8.1 HTTP POST Transport Binding

In this binding, the DSS request/response exchange occurs within an HTTP POST exchange **[RFC 2616]** .

The following rules apply to the HTTP request:

1. The client may send an HTTP/1.0 or HTTP/1.1 request.

2. The Request URI may be used to indicate a particular service endpoint.

3. « The Content-Type header MUST be set to “application/xml” or “application/json”. » [DSS-8.1-1]

4. « The Content-Length header MUST be present and correct. » [DSS-8.1-2]

5. « The DSS request message MUST be sent in the body of the HTTP Request. » [DSS-8.1-3]

The following rules apply to the HTTP Response:

1. « The Content-Type header MUST be set to “text/xml” or “application/json”. » [DSS-8.1-4]

2. « The Content-Length header MUST be present and correct. » [DSS-8.1-5]

3. « The DSS response message MUST be sent in the body of the HTTP Response. » [DSS-8.1-6]

4. « The HTTP status code MUST be

a. either set to 200 if a DSS response message is returned.

b. or the status code can be set to

    i. either 3*xx* to indicate a redirection

    ii. or 4*xx* to indicate a low-level client error (such as a malformed request)

    iii. or 5*xx* to indicate a low-level server error » [DSS-8.1-7]

8.2 SOAP 1.2 Transport Binding

In this binding, the DSS request/response exchange occurs using the SOAP 1.2 message protocol **[SOAP]** .

The following rules apply to the SOAP request:

1. A single DSS SignRequest or VerifyRequest element will be transmitted within the body of the SOAP message.

2. « The client MUST NOT include any additional XML elements in the SOAP body. » [DSS-8.2-1]

3. « The character encoding UTF-8 MUST be used for the SOAP message. » [DSS-8.2-2]

4. Arbitrary SOAP headers may be present.

The following rules apply to the SOAP response:

1. « The server MUST return either a single DSS SignResponse or VerifyResponse element within the body of the SOAP message, or a SOAP fault code. » [DSS-8.2-3]

2. « The server MUST NOT include any additional XML elements in the SOAP body. » [DSS-8.2-4]

3. « If a DSS server cannot parse a DSS request, or there is some error with the SOAP envelope, the server MUST return a SOAP fault code » [DSS-8.2-5]. Otherwise, a DSS result code should be used to signal errors.

4. « The character encoding UTF-8 MUST be used for the SOAP message. » [DSS-8.2-6]

5. Arbitrary SOAP headers may be present.

« On receiving a DSS response in a SOAP message, the client MUST NOT send a fault code to the DSS server. » [DSS-8.2-7]

8.3 Security Bindings

It is good practice to use a security binding (e.g. TLS) to provide confidentiality, authentication and integrity.

The selection of security mechanism and the used parameters depends on many aspects of the usage scenario, for example:

· Required protection level of the content

· Technical limitations (e.g. introduced by mobile clients)

· Regulatory requirements

· Export restrictions

Moreover, these decisions always need to be reconsidered due to new results crypto analysis and known vulnerabilities. Therefore, details regarding protocols and cipher suites are out of scope of this document.

9 DSS-Defined Identifiers

The following sections define various URI-based identifiers. Where possible an existing URN is used to specify a protocol. In the case of IETF protocols the URN of the most current RFC that specifies the protocol is used (see **[RFC 2648]**). URI references created specifically for DSS have the following stem:

urn:oasis:names:tc:dss:2.0:

9.1 Signature Type Identifiers

The following identifiers MAY be used as the content of the <SignatureType> optional input (see section 3.5.1).

9.1.1 XML Signature

- **URI:** urn:ietf:rfc:3275
- This refers to an XML signature per **[XMLDSIG]**.

9.1.2 XML TimeStampToken

- **URI:** urn:oasis:names:tc:dss:2.0:core:schema:XMLTimeStampToken
- This refers to an XML timestamp containing an XML signature, per section 5.1.

9.1.3 RFC 3161 TimeStampToken

- **URI:** urn:ietf:rfc:3161
- This refers to an XML timestamp containing an ASN.1 TimeStampToken, per **[RFC 3161]**.

9.1.4 CMS Signature

- **URI:** urn:ietf:rfc:3369
- This refers to a CMS signature per **[RFC 5652****]** or prior versions of CMS.

9.1.5 PGP Signature

- **URI:** urn:ietf:rfc:2440
- This refers to a PGP signature per **[RFC 2440]**.

9.2 ResultMinors

The following list contains the values of ResultMinor that are used in this document.

Abbreviation

URI

OnAll Documents

urn:oasis:names:tc:dss:2.0:resultminor:valid:signature:OnAllDocuments

NotAll Documents Referenced

`urn:oasis:names:tc:dss:2.0:resultminor:valid:signature:NotAllDocumentsReferenced`

Incorrect Signature

`urn:oasis:names:tc:dss:2.0:resultminor:invalid:IncorrectSignature`

HasManifest Results

`urn:oasis:names:tc:dss:2.0:resultminor:valid:signature:HasManifestResults`

Invalid Signature Timestamp

`urn:oasis:names:tc:dss:2.0:resultminor:valid:signature:InvalidSignatureTimestamp`

Referenced DocumentNot Present

`urn:oasis:names:tc:dss:2.0:resultminor:ReferencedDocumentNotPresent`

KeyInfoNot Provided

`urn:oasis:names:tc:dss:1.0:resultminor:KeyInfoNotProvided`

MoreThanOne RefUriOmitted

`urn:oasis:names:tc:dss:2.0:resultminor:MoreThanOneRefUriOmitted`

InvalidRefURI

`urn:oasis:names:tc:dss:2.0:resultminor:InvalidRefURI`

NotSupported

`urn:oasis:names:tc:dss:2.0:resultminor:NotSupported`

Inappropriate Signature

`urn:oasis:names:tc:dss:2.0:resultminor:Inappropriate:signature`

General Error

`urn:oasis:names:tc:dss:2.0:resultminor:GeneralError`

KeyLookup Failed

`urn:oasis:names:tc:dss:2.0:resultminor:invalid:KeyLookupFailed`

CrlNot Availiable

urn:oasis:names:tc:dss:2.0:resultminor:CrlNotAvailiable

OcspNot Availiable

urn:oasis:names:tc:dss:2.0:resultminor:OcspNotAvailiable

Certificate Chain NotComplete

urn:oasis:names:tc:dss:2.0:resultminor:CertificateChainNotComplete

10 Security Considerations

There are several potential avenues for attack when processing incoming DSS documents. The following list is non-exhaustive and should not and can not replace a comprehensive security review.

A comprehensive security review considers the unique technology stack and processes specific to an implementation and not of all implementations.

10.1 Well-Known Attack Vectors

In the following subsections four well-known classes of attack vectors are highlighted:

1. XML Parsing Vulnerabilities

2. XML Canonicalization Vulnerabilities

3. Injection Attacks

4. JSON Deserialization Through Evaluation Attacks

The first two attack vector classes “XML Parsing Vulnerabilities” and “XML Canonicalization Vulnerabilities” can occur in any XML language and therefore do not rely on any specific DSS capabilities.

The third class, “Injection Attacks” applies to any format that is being processed in a deterministic way by an active processor with additional capabilities potentially being triggered by an unexpected and malicious payload.

“JSON Deserialization Through Evaluation Attacks” attack vectors consider processing programming languages and specifically collisions of constructs in the processing language used to consumer any JSON text and the allowed constructs in the JSON format.

In addition to the attack vectors listed and further detailed in the following non-normative subsections, DSS document processing requires interfaces to BASE64 and ASN.1 encoding and decoding in practical implementations, which MAY result in further attack vectors.

10.1.1 XML Parsing Vulnerabilities [non-normative]

There have been vulnerabilities in XML parsing libraries that can cause either denial of service or actual exploits. As an example, see Microsoft’s article on XML Denial of Service Attacks and Defenses. The best defense for these types of attacks is, in short, to keep the XML parser up-to-date and ensure to perform full validation prior to attempting to process the document.

10.1.2 XML Canonicalization Vulnerabilities [non-normative]

Exploitation of the use of canonicalization as content extractor MAY impact an implementation that offers e.g. inclusion and processing of XML Fragments in payloads as described e.g. in [JENSEN-2009].

10.1.3 Injection Attacks [non-normative]

Any DSS content MAY be processed somewhere, thus injection attacks MAY occur in many places which are not specific to DSS. The best defense known, is to sanitize untrusted output (and anything inside a DSS document received from outside the client or server system boundaries should be considered untrusted). For more explanation on injection attacks, see e.g. this OWASP article (https://www.owasp.org/index.php/Top_10-2017_A1-Injection).

10.1.4 JSON Deserialization Through Evaluation Attacks [non-normative]

Generally, there are security issues with processing languages that are capable to evaluate text in that processing language during runtime and dynamically.

Sample vector for JavaScript:

“JSON is a subset of JavaScript that excludes assignment and invocation. Since JSON's syntax is borrowed from JavaScript, it is possible to use that language's "eval()" function to parse most JSON texts (but not all; certain characters such as U+2028 LINE SEPARATOR and U+2029 PARAGRAPH SEPARATOR are legal in JSON but not JavaScript). This generally constitutes an unacceptable security risk, since the text could contain executable code along with data declarations. The same consideration applies to the use of eval()-like functions in any other programming language in which JSON texts conform to that language's syntax.” (cf. [RFC8259] section 12 “Security Considerations”.

11 Conformance

11.1 Conformance as a DSS version 2.0 document

To ease communication and subsequent resolution of any specific partial conformance violation, the preceding chapters already provide minimal requirements, that a specific instance component must fulfill, to permit conformance of the complete DSS version 2.0 document.

11.1.1 Conformance for JSON format

The following clause offers a simple two step process, to either prove or disprove the conformance of a complete JSON document (formulated in terms specific to that implementation language) to this version of DSS:

« Conformance Clause 1: “Valid JSON DSS Document” A JSON document instance conforms to this specification as a DSS document if it meets all of the following COUNT_ME conditions:

1. Is valid JSON

2. Validates against the JSON Schema

» DSS-11.1.1-1]

11.1.2 Conformance for XML format

The following clause offers a simple three step process, to either prove or disprove the conformance of a complete XML document (formulated in terms specific to that implementation language) to this version of DSS:

« Conformance Clause 1: “Valid XML DSS Document” An XML document instance conforms to this specification as a DSS document if it meets all of the following three conditions:

1. Is well-formed XML.

2. Consists of a single root element instance as defined in the namespace http://docs.oasis-open.org/dss-x/ns/core.

3. Is valid XML.

» [DSS-11.1.2-1]

11.1.3 Conformance for DSS Server

« Conformance Clause 1: “Conforming DSS Server” A DSS server instance conforms to this specification if the server fulfills all requirements on servers stated in the normative sections of this document. » [DSS-11.1.3-1]

11.1.4 Conformance for DSS Client

« Conformance Clause 1: “Conforming DSS Client” A DSS client instance conforms to this specification if the client fulfills all requirements on clients stated in the normative sections of this document. » [DSS-11.1.4-1]

Appendix A. Acknowledgments

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Andreas Kuehne, Individual

Detlef Huehnlein, Individual

Ernst Jan van Nigtevecht, Sonnenglanz Consulting

Juan Carlos Cruellas, Univ Politecnica de Cataluna

Stefan Hagen, Individual

Appendix B. Index of Components and Elements

Appendix C. List of Figures

Appendix D. Revision History

Revision	**Date**	**Editor**	**Changes Made**
WD06	2018-06-10	Andreas Kuehne and Stefan Hagen	Initial Draft version with feedback from the TC

WD07

2018-08-12

Stefan Hagen

Minor editorial fixes

WD08

2018-08-13

Andreas Kuehne

Editorial fixes to ease reading for newcomers (grouping of elements)

WD09

2018-08-20

Stefan Hagen

Revision of namespaces

WD10

2018-08-21

Andreas Kuehne and Stefan Hagen

Fix for JSON Schema URL Encoded ref attribute, alphabetical ordering of refereemces, application of OASIS conformance guidelines, addition of Security Considerations section, repair of broken links, and insertion of test assertion tags.