ETSI TS 119 182-1 V1.1.1 (2021-03)

Electronic Signatures and Infrastructures (ESI);
JAdES digital signatures;
Part 1: Building blocks and JAdES baseline signatures

Reference
DTS/ESI-0019182-1

Keywords
electronic signature, JSON

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

Foreword

This Technical Specification (TS) has been produced by ETSI Technical Committee Electronic Signatures and Infrastructures (ESI).

The present document is part 1 of a multi-part deliverable covering JAdES digital signatures, as identified below:

Part 1: "Building blocks and JAdES baseline signatures";

Part 2: "Extended JAdES signatures".

One JSON schema file, whose location is detailed in clause B.1 and which contain JSON Schema definitions complements the present document.

The present document has taken as starting point the paper [i.18] "Bringing JSON signatures to ETSI AdES framework: meet JAdES signatures", by Juan Carlos Cruellas, in Computer Standards and Interfaces, Volume 71, August 2020.

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Introduction

Electronic commerce has emerged as a frequent way of doing business between companies across local, wide area and global networks. Trust in this way of doing business is essential for the success and continued development of electronic commerce. It is therefore important that companies using this electronic means of doing business have suitable security controls and mechanisms in place to protect their transactions and to ensure trust and confidence with their business partners. In this respect digital signatures are an important security component that can be used to protect information and provide trust in electronic business.

The present document is intended to cover digital signatures supported by PKI and public key certificates, and aims to meet the general requirements of the international community to provide trust and confidence in electronic transactions, including, amongst other, applicable requirements from Regulation (EU) No 910/2014 [i.1].

The present document can be used for any transaction between an individual and a company, between two companies, between an individual and a governmental body, etc. The present document is independent of any environment. It can be applied to any environment e.g. smart cards, SIM cards, special programs for electronic signatures, etc.

The present document is part of a rationalized framework of standards (see ETSI TR 119 000 [i.4]).

1 Scope

The present document:

1) Specifies a JSON [1] format for AdES signatures (JAdES signatures hereinafter) built on JSON Web Signatures (JWS hereinafter) as specified in IETF RFC 7515 [2]. For this, the present document:

   - Extends the JSON Web Signatures specified in IETF RFC 7515 [2] by defining an additional set of JSON header parameters that can be incorporated in the JOSE Header (either in its JWS Protected Header or its JWS Unprotected Header parts). Many of these new header parameters have the same semantics as the attributes/properties defined in CAdES [i.2] and XAdES [4] digital signatures. Other header parameters are defined to meet specific requirements that current JSON Web Signatures cannot meet (e.g. for explicitly referencing detached JWS Payload). These new header parameters and their corresponding types are defined in a JSON schema.

   - Specifies the mechanisms for incorporating the aforementioned JSON components in JSON Web Signatures [2] to build JAdES signatures, offering the same features as CAdES and XAdES in JSON syntax, and therefore fulfilling the same requirements (such as the long-term validity of digital signatures).

2) Defines four levels of JAdES baseline signatures addressing incremental requirements to maintain the validity of the signatures over the long term. Each level requires the presence of certain JAdES header parameters, suitably profiled for reducing the optionality as much as possible. The aforementioned levels provide the basic features necessary for a wide range of business and governmental use cases for electronic procedures and communications to be applicable to a wide range of communities when there is a clear need for interoperability of digital signatures used in electronic documents.

EXAMPLE: An example of requirements raised in specific domains is signing HTTP messages exchanged by parties in certain environments, which require signing both the HTTP body and some specific http headers. The format specified in IETF RFC 7515 [2] does not provide any native mechanism for individually identifying a detached JWS Payload. Clause 5.2.8 of the present document defines `sigD`, a new JSON header parameter that allows to identify one or more detached data objects which, suitably processed and concatenated, form the detached JWS Payload.

Procedures for creation, augmentation, and validation of JAdES digital signatures are out of scope.

NOTE 1: ETSI EN 319 102-1 [i.3] specifies procedures for creation, augmentation and validation of other types of AdES digital signatures.

The present multi-part deliverable aims at supporting electronic signatures independent of any specific regulatory framework.

NOTE 2: Specifically, but not exclusively, it is the aim that JAdES digital signatures specified in the present multi-part deliverable can be used to meet the requirements of electronic signatures, advanced electronic signatures, qualified electronic signatures, electronic seals, advanced electronic seals, and qualified electronic seals as defined in Regulation (EU) No 910/2014 [i.1].

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] IETF RFC 8259 (December 2017): "The JavaScript Object Notation (JSON) Data Interchange Format".

[2] IETF RFC 7515 (May 2015): "JSON Web Signature (JWS)".

[3] IETF RFC 3061 (February 2001): "A URN Namespace of Object Identifiers".

[4] ETSI EN 319 132-1: "Electronic Signatures and Infrastructures (ESI); XAdES digital signatures; Part 1: Building blocks and XAdES baseline signatures".

[5] IETF RFC 5035 (August 2007): "Enhanced Security Services (ESS) Update: Adding CertID Algorithm Agility".

[6] Recommendation ITU-T X.509: "Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks".

[7] IETF RFC 3161 (August 2001): "Internet X.509 Public Key Infrastructure Time Stamp Protocol (TSP)".

[8] IETF RFC 5280 (May 2008): "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile".

[9] IETF RFC 6960 (June 2013): "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP".

[10] IETF RFC 5816 (April 2010): "ESSCertIDv2 Update for RFC 3161".

[11] IETF RFC 3494 (March 2003): "Lightweight Directory Access Protocol version 2 (LDAPv2) to Historic Status".

[12] IETF RFC 4648 (October 2006): "The Base16, Base32, and Base64 Data Encodings".

[13] IETF RFC 3230 (January 2002): "Instance Digests in HTTP".

[14] IETF RFC 7797 (February 2016): "JSON Web Signature (JWS) Unencoded Payload Option".

[15] IETF RFC 3339 (July 2002): "Date and Time on the Internet: Timestamps".

[16] IETF RFC 7518 (May 2015): "JSON Web Algorithms (JWA)".

[17] IETF RFC 3986 (January 2005): "Uniform Resource Identifier (URI): Generic Syntax".

[18] IETF RFC 2616 (June 1999): "Hypertext Transfer Protocol - HTTP/1.1".

[19] draft-handrews-json-schema-01 (March 2018): "JSON Schema: A Media Type for Describing JSON Documents".

[20] draft-handrews-json-schema-validation-01 (March 2018): "JSON Schema Validation: A Vocabulary for Structural Validation of JSON".

[21] ETSI TS 119 312 (V1.3.1): "Electronic Signatures and Infrastructures (ESI); Cryptographic Suites".

[22] FIPS Publication 180-4 (August 2015): "Secure Hash Standard (SHS)", National Institute of Standards and Technology.

[23] FIPS Publication 202 (August 2015): "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", National Institute of Standards and Technology.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] Regulation (EU) No 910/2014 of the European Parliament and of the Council on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC. OJ L 257, 28.08.2014, p. 73-114.

[i.2] ETSI EN 319 122-1: "Electronic Signatures and Infrastructures (ESI); CAdES digital signatures; Part 1: Building blocks and CAdES baseline signatures".

[i.3] ETSI EN 319 102-1: "Electronic Signatures and Infrastructures (ESI); Procedures for Creation and Validation of AdES Digital Signatures; Part 1: Creation and Validation".

[i.4] ETSI TR 119 000: "Electronic Signatures and Infrastructures (ESI); The framework for standardization of signatures: overview".

[i.5] ETSI TR 119 001: "Electronic Signatures and Infrastructures (ESI); The framework for standardization of signatures; Definitions and abbreviations".

[i.6] ETSI TR 119 100: "Electronic Signatures and Infrastructures (ESI); Guidance on the use of standards for signature creation and validation".

[i.7] ETSI TS 119 172-1: "Electronic Signatures and Infrastructures (ESI); Signature Policies; Part 1: Building blocks and table of contents for human readable signature policy documents".

[i.8] OASIS Standard: "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0".

[i.9] ETSI TS 101 533-1: "Electronic Signatures and Infrastructures (ESI); Data Preservation Systems Security; Part 1: Requirements for Implementation and Management".

[i.10] IETF RFC 4998: "Evidence Record Syntax (ERS)".

[i.11] W3C Recommendation (19 November 2019): "Verifiable Credentials Data Model 1.0".

[i.12] draft-cavage-http-signatures-10 (May 2018): "Signing HTTP Messages".

[i.13] JSON Schema Specification in json-schema.org website.

NOTE: Available at https://json-schema.org/specification.html.

[i.14] draft-handrews-json-schema-02 (September 2019): "JSON Schema: A Media Type for Describing JSON Documents".

[i.15] draft-handrews-json-schema-validation-02 (September 2019): "JSON Schema Validation: A Vocabulary for Structural Validation of JSON".

[i.16] IETF RFC 7517 (May 2015): "JSON Web Key (JWK)".

[i.17] ISO 3166-1: "Codes for the representation of names of countries and their subdivisions -- Part 1: Country code".

[i.18] Juan Carlos Cruellas: "Bringing JSON signatures to ETSI AdES framework: meet JAdES signatures". Computer Standards and Interfaces, Volume 71, August 2020.

3 Definition of terms, symbols, abbreviations and terminology

3.1 Terms

For the purposes of the present document, the terms given in ETSI TR 119 001 [i.5], IETF RFC 7515 [2] and the following apply:

JAdES signature: JSON Web Signature

NOTE: As specified in IETF RFC 7515 [2], or other parts of this multi-part deliverable.

JWS Signature Value: digital signature cryptographic value calculated over a sequence of octets derived from the JWS Protected Header and data to be signed

NOTE 1: IETF RFC 7515 [2] uses the term JWS Signature for this concept. The present document does not use this term, but the JWS Signature Value, for the sake of terminological coherence of other AdES specifications.

NOTE 2: The present document uses the term **JSON Web Signature,** as defined by IETF RFC 7515 [2], i.e. for denoting the JSON data structure for representing a digitally signed message.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

ASCII	American Standard Code For Information Interchange
ASN.1	Abstract Syntax Notation 1
CA	Certification Authority
CRL	Certificate Revocation List
FIPS	Federal Information Processing Standards
HTTP	Hyper Text Transfer Protocol
IETF	Internet Engineering Task Force
ISO	International Organization for Standardization
ITU-T	International Telecommunication Union Telecommunication Standardization Sector
JOSE	JSON Object Signing and Encryption
JSON	JavaScript Object Notation
JWS	JSON Web Signature
OCSP	Online Certificate Status Protocol
OID	Object IDentifier
PKI	Public Key Infrastructure
RFC	Request For Comments
SAML	Security Assertion Markup Language

SHA	Secure Hash Algorithm
SIM	Subscriber Identification Module
SPO	Service Provision Option
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
UTC	Coordinated Universal Time

3.4 Terminology

The present document adopts, wherever it is possible the same terminology as the terminology used in IETF RFC 7515 [2] and in IETF RFC 8259 [1].

Therefore, within the present document, the term "JSON Web Signature" shall denote the JSON structure specified in IETF RFC 7515 [2].

The present document uses the term "JSON value" for denoting JSON objects, or JSON arrays, or JSON numbers, or JSON strings, i.e. a subset of the potential meanings of "JSON value" listed in clause 3 of IETF RFC 8259 [1].

The present document uses the term "header parameter" for denoting a JSON object, JSON array, JSON number, or JSON string, which is member either of the JWS Protected Header or the JWS Unprotected Header specified in IETF RFC 8259 [1].

The present document uses the term "member" for denoting a JSON object's member, as specified in clause 4 of IETF RFC 8259 [1].

The present document uses the term "element" or "element of the array" for denoting the contents of a position within a JSON array (specified in clause 5 of IETF RFC 8259 [1]).

NOTE: These last terms will be used for denoting each of the JSON values that will be added to the `etsiU` JSON array (specified in clause 5.3.1 of the present document), which will be incorporated in the JWS Unprotected header as a header parameter. Therefore, these JSON values will play, within the present document, an equivalent role to the role played by the unsigned attributes in CAdES and the unsigned qualifying properties in XAdES.

The present document uses the term "JAdES component" or "component" for denoting any JAdES signature constituent, regardless it is a header parameter, a member of a JSON Object, an element of a JSON array, or any other JSON Value.

The present document uses this special `font` for denoting tags of JAdES components.

As for the names of the header parameters and elements of the `etsiU` JSON array, the following criteria and conventions have been used:

1) The names have been selected to have a maxim length of 8 characters; most of the names are shorter.

2) The names of header parameters qualifying the signature itself use to start with "sig".

3) The names of header parameters qualifying the signer use to start with "sr".

4) The names of header parameters qualifying the data to be signed use to start with "sd".

5) The names of header parameters dealing with time-stamp tokens use to finalize with "tst".

6) The names of header parameters dealing with certificates use to start or contain "x" (following the convention of IETF RFC 7515 [2], which defines the header parameters `x5u`, `x5c`, `x5t`, and `x5t#S256`).

7) The names of header parameters dealing with revocation values (CRLs or OCSP responses) use to start or contain "r".

8) The names of header parameters dealing with attribute certificates or the corresponding revocation values use to start "a".

9) The names of header parameters dealing with values (of certificates or revocation values) use to contain "Vals".

10) The names of header parameters dealing with references (to certificates or revocation values) use to contain "Refs" (except `x5t`, and `x5t#S256`, which have been defined in IETF RFC 7515 [2], contain references to certificates, and do not include it).

4 General Requirements

The JAdES components defined in the present document shall be carried within the JOSE header as specified in IETF RFC 7515 [2].

All the JAdES signed header parameters specified in clause 5.2 of the present document, as well as: `cty`, `kid`, `crit`, and `x5u` header parameters specified in IETF RFC 7515 [2] and further profiled in clause 5.1 of the present document, if required to be present, shall be incorporated as header parameters of the JWS Protected Header of the JSON Web Signature, specified in IETF RFC 7515 [2].

JAdES signatures may be serialized using either JWS Compact Serialization or JWS JSON Serialization as specified in clause 3 of IETF RFC 7515 [2].

JWS Unprotected Header in JAdES signatures shall contain only one header parameter, namely the `etsiU` header parameter (specified in clause 5.3 of the present document), which is defined as a JSON array.

NOTE 1: The rationale for this is that the JWS Unprotected Header is a JSON object, and no order may be inferred in its different members. This is the reason why the present document defines `etsiU` header parameter as a JSON array.

NOTE 2: The elements of this JSON array will contain JSON values that play for JAdES signatures the same role as the role played by the unsigned attributes for CAdES signatures, and the role played by the unsigned qualifying properties for XAdES signatures.

NOTE 3: An immediate consequence is that a time-stamp token present within the `arcTst` object specified in clause 5.3.6.2 of the present document, protects the JWS Payload, the JWS Protected Header, the JAdES Signature Value, and the `etsiU` header parameter within the JWS Unprotected Header.

Header parameters defined by IETF RFC 7515 [2] and IETF RFC 7797 [14] not further profiled within the present document may be added as header parameters within the JAdES signature, following the requirements specified in the present document.

In JAdES signatures, the JWS Payload may be attached or detached.

Detached JWS Payload may either be one detached object, or result from the concatenation of more than one detached data objects. See the specification of `sigD` signed header parameter in clause 5.2.8 of the present document.

NOTE 4: At the moment of producing the present document, JSON Schema was under development. The working draft being used at the present document was the one specified by draft-handrews-json-schema-01 [19], and draft-handrews-json-schema-validation-01 [20]. These documents, though, do not correspond to the latest version (draft-handrews-json-schema-02 [i.14], and draft-handrews-json-schema-validation-02 [i.15]) due to the fact that tools checking correctness of JSON schema files have not been yet completed. The drafts of JSON schema specifications may be accessed at JSON Schema Specification in json-schema.org website [i.13].

NOTE 5: Although at the moment of producing the present document there exist several proposals for JSON canonicalization algorithms, none have been formally adopted by any standardisation organization. Nevertheless, the present document uses placeholders for identifiers of canonicalization algorithms in a number of components that could use them if such algorithms are standardized in the future.

5 Header parameters semantics and syntax

5.1 Use of header parameters defined in IETF RFC 7515 and IETF RFC 7797

5.1.1 Introduction

This clause defines additional requirements for the use of some of header parameters specified in IETF RFC 7515 [2].

JAdES signatures may incorporate any of the header parameters specified in IETF RFC 7515 [2] and IETF RFC 7797 [14].

NOTE: Clause 6.3 also specifies requirements (mainly of presence and cardinality), for the use of some of the header parameters specified in IETF RFC 7515 [2] for JAdES baseline signatures.

5.1.2 The `alg` (X.509 URL) header parameter

Semantics

The `alg` header parameter shall be a signed header parameter that qualifies the signature.

The `alg` header parameter shall have the semantics specified in IETF RFC 7515 [2], clause 4.1.1.

Syntax

The `alg` header parameter shall have the syntax specified in IETF RFC 7515 [2], clause 4.1.1.

Its value should be one of the algorithms for digital signatures recommended by in ETSI TS 119 312 [21].

5.1.3 The `cty` (content type) header parameter

Semantics

The `cty` header parameter shall be a signed header parameter that qualifies the JWS Payload.

The `cty` header parameter shall have the semantics specified in IETF RFC 7515 [2], clause 4.1.10.

The `cty` header parameter should not be present if the `sigD` header parameter, specified in clause 5.2.8 of the present document, is present within the JAdES signature.

The `cty` header parameter should not be present if the content type is implied by the JWS Payload.

The `cty` header parameter shall not be present if the JWS Payload is a (counter-signed) signature.

NOTE: The `sigD` header parameter has one member that contains information of the format and type of the constituents of the JWS Payload.

Syntax

The `cty` header parameter shall have the syntax specified in IETF RFC 7515 [2], clause 4.1.10.

5.1.4 The `kid` (key identifier) header parameter

Semantics

The `kid` header parameter shall be a signed header parameter that qualifies the signature.

The `kid` header parameter shall have the semantics specified in IETF RFC 7515 [2], clause 4.1.4.

The content of `kid` header parameter shall be the base64 (IETF RFC 4648 [12]) encoding of one DER-encoded instance of type `IssuerSerial` type defined in IETF RFC 5035 [5].

The header parameter `kid` shall be used as a hint that can help to identify the signing certificate if other header parameters referencing or containing the signing certificate are present in the JAdES signature.

Syntax

The `kid` header parameter shall have the syntax specified in IETF RFC 7515 [2], clause 4.1.4.

5.1.5 The x5u (X.509 URL) header parameter

Semantics

The `x5u` header parameter shall be a signed header parameter that qualifies the signature.

The `x5u` header parameter shall have the semantics specified in IETF RFC 7515 [2], clause 4.1.5.

The `x5u` member shall be used as a hint, as implementations can have alternative ways for retrieving the referenced certificate if it is not found at the referenced place.

Syntax

The `x5u` header parameter shall have the syntax specified in IETF RFC 7515 [2], clause 4.1.5.

5.1.6 The x5t (X.509 Certificate SHA-1 Thumbprint) header parameter

JAdES signatures shall not contain the `x5t` header parameter specified in clause 4.1.7 of IETF RFC 7515 [2].

5.1.7 The x5t#S256 (X.509 Certificate SHA-256 Thumbprint) header parameter

Semantics

The `x5t#S256` shall be a signed header parameter that qualifies the signature.

The `x5t#S256` header parameter shall have the semantics specified in IETF RFC 7515 [2], clause 4.1.8.

Syntax

The `x5t#S256` header parameter shall have the syntax specified in IETF RFC 7515 [2], clause 4.1.8.

A JAdES signature shall have at least one of the following header parameters in its JWS Protected Header: `x5t#S256` (specified in clause 4.1.8 of IETF RFC 7515 [2]), `x5c` (specified in clause 4.1.6 of IETF RFC 7515 [2]), `sigX5ts` (specified in clause 5.2.2.3 of the present document), or `x5t#o` (specified in clause 5.2.2 of the present document).

NOTE 1: The simultaneous presence of `x5t#S256` and `x5t#o` header parameters is allowed for facilitating interoperability whilst implementations migrate from `x5t#S256` to `x5t#o`.

NOTE 2: Profiles of JAdES can allow some of the combinations of the aforementioned header parameters.

5.1.8 The x5c (X.509 Certificate Chain) header parameter

Semantics

The `x5c` header parameter shall be a signed header parameter.

The `x5c` header parameter shall have the semantics specified in IETF RFC 7515 [2], clause 4.1.6.

Syntax

The `x5c` header parameter shall have the syntax specified in IETF RFC 7515 [2], clause 4.1.6.

5.1.9 The `crit` (critical) header parameter

Semantics

The `crit` header parameter shall be a signed header parameter that qualifies the signature.

The `crit` header parameter shall have the semantics specified in IETF RFC 7515 [2], clause 4.1.11.

The JAdES signatures incorporating any signed header parameter specified in clause 5.2 shall incorporate the signed `crit` header parameter.

Syntax

The `crit` header parameter shall have the syntax specified in IETF RFC 7515 [2], clause 4.1.11.

The elements of the `crit` JSON array shall be the names of all the signed header parameters that are present in the JAdES signatures and specified in clause 5.2.

5.1.10 The `b64` header parameter

Semantics

The `b64` header parameter shall be a signed header parameter.

The `b64` header parameter shall have the semantics specified in IETF RFC 7797 [14], clause 3.

Syntax

The `b64` header parameter shall have the syntax specified in IETF RFC 7797 [14], clause 3.

If the `sigD` header parameter is present with its member set to `"http://uri.etsi.org/19182/HttpHeaders"` then the `b64` header parameter shall be present and set to `"false"`.

5.2 New signed header parameters

5.2.1 The `sigT` (claimed signing time) header parameter

Semantics

The `sigT` header parameter shall be a signed header parameter that qualifies the signature.

The `sigT` header parameter's value shall specify the time at which the signer claims to have performed the signing process.

Syntax

The `sigT` header parameter shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information:

```
"sigT": {"type": "string", "format": "date-time"},
```

The contents of the string:

1) Shall be formatted as specified in IETF RFC 3339 " [15].

2) Shall be the UTC time for date and time.

3) Shall not contain the part corresponding to fraction of seconds.

EXAMPLE: "2019-11-19T17:28:15Z".

5.2.2 Header parameters for certificate references

5.2.2.1 Introduction

The present clause defines two new signed parameters namely, the `x5t#o`, which extends the semantics of `x5t#S256` for allowing different digest algorithms than the SHA256, and the `sigX5ts`, which contains references to several certificates within the certification path, including one reference to the signing certificate, computed using a certain arbitrary digest algorithm.

5.2.2.2 The `x5t#o` (X509 certificate digest) header parameter

Semantics

The `x5t#o` header parameter shall be a signed header parameter that qualifies the signature.

The `x5t#o` header parameter shall contain an identifier of a digest algorithm different than the identifier of SHA-256, and the digest value of the referenced certificate.

NOTE 1: For instance, the signature validation policy can mandate other certificates to be present which can include all the certificates up to the trust anchor.

NOTE 2: The identifier of SHA-256 is not allowed because for this algorithm, x5t#256 header parameter has been already specified in IETF RFC 7515 [2].

The `x5t#o` header parameter shall not contain any other information.

Syntax

The `x5t#o` header parameter shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information.

```
"x5t#o": {
    "type": "object",
    "properties": {
        "digAlg": {"type": "string"},
        "digVal": {"type": "string", "contentEncoding": "base64"}
    },
    "required": ["digAlg","digVal"],
    "additionalProperties": false
},
```

The `digAlg` member shall identify the digest algorithm.

The `digVal` member shall contain the base64url-encoded value of the digest computed on the DER-encoded certificate.

5.2.2.3 The `sigX5ts` (X509 certificates digests) header parameter

Semantics

The `sigX5ts` header parameter shall be a signed header parameter that qualifies the signature.

The `sigX5ts` header parameter shall contain several references of certificates within the certification path of the signing certificate, each one formed by the identifier of a digest algorithm and the digest value of the referenced certificate.

NOTE: This header parameter is not used when only the reference to the signing certificate is required. Instead, the `x5t#o` header parameter is used in such occasions.

This element may contain digest values computed with algorithm SHA-256.

The first reference within the `sigX5ts` header parameter shall be the reference of the signing certificate.

The `sigX5ts` header parameter shall not contain any other information.

Syntax

The `sigX5ts` header parameter shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information:

```
"sigX5ts": {
    "type": "array",
    "items": {"$ref": "#/definitions/x5t%23o"},
    "minItems": 2
},
```

5.2.3 The `srCms` (signer commitments) header parameter

Semantics

The `srCms` header parameter shall be a signed header parameter that qualifies JWS Payload.

The `srCms` header parameter shall indicate the commitment made by the signer when signing.

The `srCms` header parameter shall express the commitment type with a URI.

The `srCms` header parameter may contain a sequence of qualifiers providing more information about the commitment.

NOTE 1: The commitment type can be:

- defined as part of the signature policy, in which case, the commitment type has precise semantics that are defined as part of the signature policy; or
- be a registered type, in which case, the commitment type has precise semantics defined by registration, under the rules of the registration authority. Such a registration authority can be a trading association or a legislative authority.

NOTE 2: The specification of commitment type identifiers is outside the scope of the present document. For a list of predefined commitment type identifiers, see ETSI TS 119 172-1 [i.7].

Syntax

This header parameter shall be carried in the JWS Protected Header.

The `srCms` header parameter shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information:

```
"srCms":{
    "type": "array",
    "items": {
        "type": "object",
        "properties":{
            "commId": {"$ref": "#/definitions/oId"},
            "commQuals":{
                "type": "array",
                "items": {"type":"object"},
                "minItems": 1
            }
        },
        "required": ["commId"],
    "additionalProperties": false
    },
    "minItems": 1
},
```

Each item of the `srCms` header parameter shall indicate one commitment made by the signer, which may be further qualified.

The `commId` member of every array item is an instance of `oId` type, which is specified in clause 5.4.1 of the present document, whose `id` member shall have a URI as value, uniquely identifying one commitment made by the signer.

The `commQuals` member provides means to include additional qualifying information on the commitment made by the signer.

5.2.4 The `sigPl` (signature production place) header parameter

Semantics

The `sigPl` header parameter shall be a signed header parameter that qualifies the signer.

The `sigPl` header parameter shall specify an address associated with the signer at a particular geographical (e.g. city) location.

Syntax

This header parameter shall be carried in the JWS Protected Header.

The `sigPl` header parameter shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information.

NOTE: Its definition follows the specification of `PostaAddress` type in schema.org (https://schema.org/PostalAddress).

```
"sigPl":{
    "type": "object",
    "properties":{
        "addressCountry": {"type": "string"},
        "addressLocality": {"type": "string"},
        "addressRegion": {"type": "string"},
        "postOfficeBoxNumber": {"type": "string"},
        "postalCode": {"type": "string"},
        "streetAddress": {"type": "string"}
    },
    "minProperties": 1,
    "additionalProperties": false
},
```

This `addressCountry` member shall contain may contain either the name of the country or its two-letter ISO 3166-1 [i.17] alpha-2 country code.

This header parameter shall be carried in the JWS Protected Header.

5.2.5 The `srAts` (signer attributes) header parameter

Semantics

The `srAts` header parameter shall be a signed header parameter that qualifies the signer.

The `srAts` header parameter shall encapsulate signer attributes (e.g. role). This header parameter may encapsulate the following types of attributes:

- attributes claimed by the signer;
- attributes certified in attribute certificates issued by an Attribute Authority; or/and
- assertions signed by a third party.

Syntax

This header parameter shall be carried in the JWS Protected Header.

The `srAts` header parameter shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information:

```
"qArrays":{
    "type": "array",
    "items": {
        "type": "object",
        "properties": {
            "mediaType": {"type": "string"},
            "encoding": {"type": "string"},
            "qVals": {
                "type": "array",
```

```
                "minItems": 1
            }
        },
        "required": ["mediaType", "encoding", "qVals"],
        "additionalProperties": false
    },
    "minItems": 1
},

"srAts":{
    "type": "object",
    "properties": {
        "certified":{
            "type": "array",
            "items": {"$ref": "#/definitions/certifiedAttrs"},
            "minItems": 1
        },
        "claimed": {"$ref": "#/definitions/qArrays"},
        "signedAssertions": {"$ref": "#/definitions/qArrays"}
    },
    "minProperties": 1,
    "additionalProperties": false
},

"certifiedAttrs": {
    "type": "object",
    "properties": {
        "x509AttrCert":{"$ref": "#/definitions/pkiOb"},
        "otherAttrCert":{"$ref": "#/definitions/pkiOb"}
    },
    "oneOf":[
        {
            "required":["x509AttrCert"]
        },
        {
            "required":["otherAttrCert"]
        }
    ],
    "additionalProperties": false
},
```

EXAMPLE: W3C Recommendation [i.11] defines a JSON model for credentials that could become content of the `claimed` member.

The `certifiedAttrs` member shall contain a non-empty array of certified attributes, which shall be one of the following:

- the base64 encoding of DER-encoded X509 attribute certificates conformant to Recommendation ITU-T X.509 [6] issued to the signer, within the `X509AttrCert` member; or
- attribute certificates (issued, in consequence, by Attribute Authorities) in different syntax than the one specified in Recommendation ITU-T X.509 [6], within the `OtherAttrCert` member. The definition of specific `OtherAttrCert` is outside of the scope of the present document.

The `signedAssertions` member shall contain a non-empty array of assertions signed by a third party.

NOTE 1: A signed assertion is stronger than a claimed attribute, since a third party asserts with a signature that the attribute of the signer is valid. However, it is less restrictive than an attribute certificate.

The `claimed` member shall contain a non-empty array of attributes claimed by the signer.

Both the `signedAssertions` and the `claimed` members shall be instances of `qArrays` type. Each instance of this type shall be a JSON array whose elements are JSON objects. Each JSON object shall contain three members, namely:

a) The `mediaType` member, which shall contain a string identifying the type of the signed assertions or the claimed attributes present in `qVals` member, and shall meet the requirements specified in clause 8.4 of draft-handrews-json-schema-validation-01 [20].

b) The `encoding` member, which shall contain a string identifying the encoding of the signed assertions or the claimed attributes present in `qVals` member, and shall meet the requirements specified in clause 8.3 of draft-handrews-json-schema-validation-01 [20].

c) The `qVals` member, which shall be a JSON array of at least one item. The elements of `qVals` JSON array shall be the values of the signed assertions or the claimed attributes encoded as indicated within the `encoding` member.

NOTE 2: Instances of `qArrays` type allow to incorporate signed assertions and/or claimed attributes of different types and different encodings.

The definition of specific content types for `signedAssertions` and claimed attributes is outside of the scope of the present document.

NOTE 3: A possible content for `signedAssertions` can be a signed SAML [i.8] assertion.

Empty `srAts` header parameters shall not be generated.

5.2.6 The `adoTst` (signed data time-stamp) header parameter

Semantics

The `adoTst` header parameter shall be a signed header parameter that qualifies the JWS Payload.

The `adoTst` header parameter shall encapsulate one or more electronic time-stamps, generated before the signature production, whose message imprint computation input shall be the JWS Payload of the JAdES signature.

Syntax

This header parameter shall be carried in the JWS Protected Header.

The `adoTst` header parameter shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information.

```
"adoTst": {"$ref": "#/definitions/tstContainer"},
```

The message imprint computation input for the time-stamp token shall be an octet stream built as indicated below:

1) If the `sigD` header parameter, as specified in clause 5.2.8 of the present document, is absent then:

   a) If the `b64` header parameter specified in clause 3 of IETF RFC 7797 [14] is present and set to `"false"` then concatenate the JWS Payload value.

   b) If the `b64` header parameter specified in clause 3 of IETF RFC 7797 [14] is present and set to `"true"`, OR it is absent, then concatenate the base64url-encoded JWS Payload.

2) Else, if the `sigD` header parameter is present:

   a) If the value of its `mId` member is "`http://uri.etsi.org/19182/HttpHeaders`" then concatenate the bytes resulting from processing the contents of its `pars` member as specified in clause 5.2.8.2 of the present document except the "`Digest`" string element. The processing of the "`Digest`" string element in the `pars` array shall consist in retrieving the bytes of the body of the HTTP message.

NOTE 1: The rationale for this is that the body of an HTTP message is signed indirectly: the JWS signature value is computed, not on the its value, but on its digest value computed with a certain digest algorithm. Therefore, in order to protect the HTTP body against the risk of the digest value becoming weak, the input to the timestamp token's message imprint computation, should contain the digest value of the HTTP body computed with a strong digest algorithm, which may be different from the initial one if this is expected to become weak soon.

b) If the value of its `mId` member is "`http://uri.etsi.org/19182/ObjectIdByURI`" or "`http://uri.etsi.org/19182/ObjectIdByURIHash`" then concatenate the bytes resulting from processing the contents of its `pars` member as specified in clause 5.2.8.3.2 of the present document.

NOTE 2: The rationale for applying the processing specified in clause 5.2.8.3.2 of the present document to the case of the mechanism identified by "`http://uri.etsi.org/19182/ObjectIdByURIHash`" is the fact that this is an indirect signing mechanism, i.e. based on signing digest values of data objects, instead the data objects themselves. Time-stamping not the digest values but the retrieved data objects, protects against future weaknesses of the digest algorithms used in `sigD`.

If the JWS Payload is detached and the JAdES signature does not incorporate the `sigD` signed header parameter, then it is out of the scope to specify how to retrieve the JWS Payload.

The `adoTst` header parameter shall not contain the `canonAlg` member.

5.2.7 The `sigPId` (signature policy identifier) header parameter

5.2.7.1 Semantics and syntax

Semantics

The `sigPId` header parameter shall be a signed header parameter qualifying the signature.

The `sigPId` header parameter shall contain either an explicit identifier of a signature policy or an indication that there is an implied signature policy that the relying party should be aware of.

NOTE: ETSI TS 119 172-1 [i.7] specifies a framework for signature policies.

Syntax

This header parameter shall be carried in the JWS Protected Header.

The `sigPId` header parameter shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information.

```
"sigPId": {
    "type": "object",
    "properties": {
        "id": {"$ref": "#/definitions/oId"},
        "digAlg": {"type": "string"},
        "digVal": {"type": "string", "contentEncoding": "base64"},
        "digPSp": {"type": "boolean"},
        "sigPQuals": {
            "type": "array",
            "items": {"$ref":"#/definitions/sigPQual"},
            "minItems": 1
        }
    },
    "required": ["id"],
    "additionalProperties": false
},
```

The `id` member shall be used for referencing the signature policy explicitly. It shall uniquely identify a specific version of the signature policy.

The `digAlg` and `digVal` members shall contain, respectively, the identifier of the digest algorithm and the digest value of the object obtained after processing `id`.

The `digPSp` member shall be a boolean. When present and set to `"true"`, it shall indicate that the digest of the signature policy document has been computed as specified in a technical specification. Absence of this member shall be considered as if present and set to `"false"`. If this member is present and set to `"true"`, then the qualifier `spDSpec` qualifier shall be present and shall identify the aforementioned technical specification.

The `sigPQuals` member shall be a non-empty array of qualifiers of the signature policy.

The `sigPQuals` member may contain one or more qualifiers of the same type.

Clause 5.2.7.2 specifies three signature policy qualifiers.

5.2.7.2 Signature policy qualifiers

Semantics

This clause specifies the following three qualifiers for the signature policy:

- A URL where a copy of the signature policy document can be obtained (`spURI` choice).
- A user notice that should be displayed when the signature is validated (`spUserNotice` choice).
- An identifier of the technical specification that defines the syntax used for producing the signature policy document (`spDSpec` choice).

Syntax

The `spURI` , `spUserNotice` , and `spDSpec` qualifiers shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and are copied below for information.

```
"sigPQual": {
    "type": "object",
    "properties": {
        "spUserNotice": {"$ref": "#/definitions/spUserNotice"},
        "spURI": {"$ref": "#/definitions/spURI"},
        "spDSpec": {"$ref": "#/definitions/spDSpec"}
    },
    "minProperties": 1,
    "maxProperties": 1
},

"spURI": {"type": "string", "format": "uri"},

"spUserNotice": {
    "type": "object",
    "properties": {
        "noticeRef": {
            "type": "object",
            "properties": {
                "organization": {"type": "string"},
                "noticeNumbers": {
                    "type": "array",
                    "items": {"type": "integer"},
                    "minItems" : 1
                }
            },
            "required": ["organization","noticeNumbers"],
            "additionalProperties": false
        },
        "explText": {"type": "string"}
    },
    "minProperties": 1,
    "additionalProperties": false
},

"spDSpec": {"$ref": "#/definitions/oId"},
```

The `spURI` qualifier shall contain a URL value where a copy of the signature policy document can be obtained.

NOTE 1: This URL can reference, for instance, a remote site (which can be managed by an entity entitled for this purpose) from where (signing/validating) applications can retrieve the signature policy document.

The `spUserNotice` qualifier shall contain information that is intended for being displayed whenever the signature is validated.

The `explText` member shall contain the text of the notice to be displayed.

NOTE 2: Other notices can come from the organization issuing the signature policy.

The `noticeRef` member shall name an organization and shall identify by numbers (`noticeNumbers` member) a group of textual statements prepared by that organization, so that the application could get the explicit notices from a notices file.

The `spDSpec` member shall identify the technical specification that defines the syntax used for producing the signature policy document.

5.2.8 The `sigD` header parameter

5.2.8.1 Semantics and Syntax

Semantics

The `sigD` header parameter shall be a signed header parameter.

The `sigD` header parameter shall not appear in JAdES signatures whose JWS Payload is attached.

The `sigD` header parameter may appear in JAdES signatures whose JWS Payload is detached.

A JAdES signature shall have at most one `sigD` header parameter.

The `sigD` header parameter shall:

1) Reference one or more detached data objects.

2) Specify how the aforementioned references shall be processed for contributing to build the sequence of octets that shall be the JWS Payload of the JAdES signature.

3) Allow to define different mechanisms for meeting the two aforementioned requirements.

Chaining of references shall not be allowed. Only the data objects directly referenced within the `sigD` header parameter shall contribute to build the JWS Payload. If some referenced object contains in its turn references to other data objects, these last data objects shall not contribute to build the JWS Payload.

NOTE: This is for avoiding building trees of referenced and distributed data objects, which would complicate the validation of JAdES signatures.

The `sigD` header parameter may also incorporate base64url-encoded digest values of the referenced data objects within one string.

The `sigD` header parameter may also incorporate any additional information for meeting requirements 1) and 2) as required by the mechanisms mentioned in 3).

Syntax

This header parameter shall be carried in the JWS Protected Header.

The `sigD` header parameter shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information.

```
"sigD":{
    "type": "object",
    "properties": {
        "mId" : {"type":"string", "format": "uri"},
        "pars" : {
            "type": "array",
            "items": {"type": "string"},
            "minItems": 1
        },
        "hashM" : {"type":"string"},
        "hashV" : {
            "type": "array",
            "items": {"type": "string","contentEncoding": "base64"},
            "minItems": 1
        },
        "ctys" : {
            "type": "array",
```

```
            "items": {"type": "string"},
            "minItems": 1
        }
    },
    "required": ["mId"],
    "additionalProperties": false
},
```

The `mId` member shall be present. It shall be an URI identifying the mechanism used for referencing and processing each referenced data object for building the JWS Payload. The present document defines 3 referencing mechanisms with their corresponding identifiers in clauses 5.2.8.2, 5.2.8.3.2, and 5.2.8.3.3.

The `pars` member shall be present. It shall be a non-empty array of strings. Each element of the array shall contain a reference to one data object, as required by the identification mechanism identified in the `mId` member.

The `hashM` member shall be a string identifying a digest algorithm. Its value shall be one of the identifiers defined in IETF RFC 7518 [16], or any future specification that amends, complements, or supersedes it. The presence of this member shall be conditional on the definition of the identification mechanism. If this member is present, then `hashV` member shall be present.

The `hashV` member shall be a non-empty array of strings. Each element of the array shall contain:

1) The base64url-encoded digest value of the data object referenced by the parameter value that is present in the same position of the `pars` array if the `b64` header parameter is present and set to `"false"`.

2) The base64url-encoded digest value of the base64url-encoded data object referenced by the parameter value that is present in the same position of the `pars` array if the `b64` header parameter is absent or it is present and set to `"true"`.

The presence of the `hashV` member shall be conditional on the definition of the identification mechanism. If this member is present, then `hashM` member shall be present.

The `ctys` member shall be a non-empty an array of strings. The contents of each element of this array shall have the same semantics of the `cty` header parameter specified in clause 4.1.10 of IETF RFC 7515 [2].

There shall be as many elements within the `ctys` array as elements within the array `pars`. Each element of the `ctys` array shall contain the information corresponding to the data object referenced by the parameter value that is present in the same position of the `pars` array, except if the content type is implied by the data object or the data object is a counter-signed signature: in these cases, the element of the `ctys` array shall have as value an empty string.

5.2.8.2 Mechanism HttpHeaders

The URL identifying this referencing mechanism shall be `"http://uri.etsi.org/19182/HttpHeaders"`.

If this mechanism is used, then the `b64` header parameter shall be present and set to `"false"`.

For this referencing mechanism, neither `hashV`, neither `hashM` member, nor `ctys` shall be present.

Using this referencing mechanism, a JAdES signature may explicitly reference several HTTP headers and sign them, as well as the HTTP message body.

For this referencing mechanism, the contents of the `pars` member shall be an array of lowercased names of HTTP header fields.

The HTTP message body may also be signed by incorporating into the HTTP message the `Digest` HTTP header specified in clause 4.3.2 of IETF RFC 3230 [13], whose content is the digest of the message body.

The HTTP headers shall be processed and concatenated in the order their lowercased names appear within the `pars` member to form the JWS Payload, as follows;

a) If the HTTP header name is "(request target)" then generate the header field value by concatenating the lower-cased method (e.g. get, put), a space character, and the path and query parts of the target URI (the "path-absolute" production and optionally a '?' character followed by the "query" production see clauses 3.3 and 3.4 of IETF RFC 3986 [17]).

b) For other HTTP header names create the header field string by concatenating the lowercased header field name followed with a colon ':', a space character, and the header field value. Any leading and trailing white spaces are removed. If there are multiple instances of the same header field, all header field values associated with the header field shall be concatenated, separated by a ASCII comma and an ASCII space ', ', and used in the order in which they will appear in the transmitted HTTP message.

c) Insert newline character after all but the last HTTP header value.

NOTE 1: The above are equivalent to the steps required for signature string construction as defined in clause 2.3 of Internet draft draft-cavage-http-signatures-10 [i.12].

NOTE 2: Clauses 5.2.6 and 5.3.6.2.2 of the present document specify the processing of the "`Digest`" string element in the `pars` array in clauses for building its contribution to the message imprint computation input when generating time-stamp tokens encapsulated within the `adoTst` and `arcTst` respectively. That processing is required because in this mechanism the body of the HTTP message is indirectly signed (what is signed is its digest, not the HTTP message body itself).

5.2.8.3 Mechanisms supported by URI references

5.2.8.3.1 General requirements

This clause specifies two mechanisms that use URIs for referencing the data objects contributing to build the JWS Payload.

For these referencing mechanisms, the contents of the `pars` member shall be an array of strings. Each string shall be an URI, appertaining to the group of URIs that can be classified as locators according to clause 1.1.3 of IETF RFC 3986 [17]. Each URI shall refer one data object.

NOTE: According to IETF RFC 3986 [17], URIs that can be classified as locators (URLs are the obvious example) "provide a means of locating the resource by describing its primary access mechanism".

If the URI-reference does not include a scheme, then HTTP scheme shall be assumed.

Dereferencing URIs in the HTTP scheme shall be supported. Dereferencing an URI in the HTTP scheme shall comply with the Status Code Definitions specified in clause 10 of IETF RFC 2616 [18].

Dereferencing URIs in other locator schemes may be supported. Dereferencing URIs within one of such schemes shall be conducted as defined in the corresponding scheme specification.

5.2.8.3.2 Mechanism ObjectIdByURI

The URI identifying this referencing mechanism shall be "`http://uri.etsi.org/19182/ObjectIdByURI`".

For this referencing mechanism, neither `hashV` , nor `hashM` shall be present. Member `ctys` may be present.

The semantics and syntax of `ctys` shall be as specified in clause 5.2.8.1 of the present document.

The stream of octets corresponding to the contribution of the JWS Payload to the computation of the JWS Signature Value shall be generated as indicated below:

1) Initialize the stream of octets to an empty stream.

2) While there are URIs in the `pars` array not visited:

   - Take the next one.

   - Dereference the URI, as specified in clause 5.2.8.3.1 of the present document.

   - If the `b64` header parameter specified in clause 3 of IETF RFC 7797 [14] is absent or is present and set to "true", then base64url encode the retrieved object octets.

   - Concatenate the resulting octets to the stream of octets to be signed.

5.2.8.3.3 Mechanism ObjectIdByURIHash

The URL identifying this referencing mechanism shall be "http://uri.etsi.org/19182/ObjectIdByURIHash".

For this referencing mechanism, the hashV, and the hashM members shall be present. Member ctys may be present.

The semantics and syntax of hashM, hashV ,and ctys shall be as specified in clause 5.2.8.1 of the present document.

For computing the digest values, whose base64url encodings appear within the hashV member, each data object referenced within the pars member, shall be retrieved as specified in clause 5.2.8.3.1 of the present document.

When using this mechanism, the JWS Payload shall contribute as an empty stream to the computation of the JWS Signature Value.

NOTE 1: As this sigD is a signed header parameter, and it already includes the digest of the components of the JWS Payload, the JWS Payload is indirectly signed by signing the sigD signed header parameter, and consequently, this referencing mechanism does not require that the JWS Payload directly contributes to the computation of the JWS Signature Value.

If the JWS Payload is required for other purposes than computing the JWS Signature Value when this mechanism is used, it shall be generated as specified in clause 5.2.8.3.2.

NOTE 2: The generation of this JWS Payload is required, for instance, for generating the adoTst or the arcTst header parameters.

5.3 New unsigned header parameter

5.3.1 The etsiU header parameter

Semantics

The etsiU unprotected header parameter shall be a JSON array whose elements contain JSON values that are not signed by the JAdES signature.

NOTE 1: The rationale for this is as follows: the computation of certain time-stamp tokens message imprints is performed by digesting the concatenation of sets of unsigned header parameters, and this concatenation needs to be performed following an order; the JSON array allows to define such an order: the unsigned header parameters are concatenated following the order of appearance within the JSON array.

NOTE 2: As it has been specified in clause 4 of the present document etsiU header parameter is incorporated in the JWS Unprotected Header specified in clause 3.2 of IETF RFC 7515 [2]. Consequently, all its elements will also be unprotected, and its elements will play in JAdES signatures the same role as the role played by the unsigned attributes for CAdES signatures, and the role played by the unsigned qualifying properties for XAdES signature.

The etsiU header parameter shall contain JSON values that qualify the JAdES signature itself, or the signer, or the JWS Payload.

NOTE 3: Because the etsiU header parameter is present within the JWS Unprotected Header, then JWS JSON Serialization as specified in IETF RFC 7515 [2], clause 3.2, needs to be employed as the alternative to JWS Compact Serialization.

The components present within the etsiU header parameter shall appear as clear instances of unsigned components or as their corresponding base64url encodings.

NOTE 4: While clear instances of unsigned components require some type of canonicalization if they contribute to the computation of a time-stamp message imprint, their base64url-encoded values will not require such canonicalization. The present document is neutral about which alternative should be used.

NOTE 5: The contents of the components of the `etsiU` header parameters can appear as clear instances of unsigned components or as their corresponding basee64url encodings regardless the presence or absence of the `b64` header parameter and its value if present, as the `b64` header parameter only affects to the JWS Payload representation and the input to the JWS Signature Value computation.

The present document specifies:

1) A JSON object (`sigPSt`) containing details for facilitating access to a signature policy document, in clause 5.3.3.

2) A JSON object (`cSig`) containing details for containing a counter-signature of the JAdES signature itself, in clause 5.3.2.

3) A JSON object (`sigTst`) containing a time-stamp token on the JWS Signature Value, in clause 5.3.4.

4) A JSON array (`xVals`) containing CA certificates required for validating the signature, in clause 5.3.5.1.

5) A JSON object (`rVals`) containing values of revocation data required for validating the signature, in clause 5.3.5.2.

6) A JSON array (`axVals`) containing certificates of Attribute Authorities required for validating the signature, in clause 5.3.5.3.

7) A JSON object (`arVals`) containing values of revocation data of Attribute Authorities required for validating the signature, in clause 5.3.5.4.

8) A JSON object (`tstVD`) containing validation data (certificate and values of revocation data) for time-stamp tokens present in the signature, in clause 5.3.6.1.

9) A JSON object (`arcTst`) containing one or more time-stamp tokens on all the components of the JAdES signature, in clause 5.3.6.2.

10) A JSON array (`xRefs`) containing references to certificates required for validating the signature, in clause A.1.1.

11) A JSON object (`rRefs`) containing references to revocation data required for validating the signature, in clause A.1.2.

12) A JSON array (`axRefs`) containing references to certificates of Attribute Authorities required for validating the signature, in clause A.1.3.

13) A JSON object (`arRefs`) containing references to revocation data of Attribute Authorities required for validating the signature, in clause A.1.4.

14) A JSON object (`sigRTst`) containing a time-stamp token on the references to the validation material and the JWS Signature Value, in clause A.1.5.1.

15) A JSON object (`rfsTst`) containing a time-stamp token on the references to the validation material, in clause A.1.5.2.

All the JSON arrays and JSON objects listed above shall be placed within the `etsiU` header parameter if they are incorporated into the JAdES signature.

Syntax

The `etsiU` header parameter shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information.

```
"etsiU": {
    "type": "array",
    "oneOf": [
        {
            "items": {"$ref": "#/definitions/etsiUClearInstance"}
        },
        {
            "items": {"type": "string", "contentEncoding": "base64"}
```

```
        }
    ],
    "minItems": 1
},

"etsiUClearInstance": {
    "type": "object",
    "properties": {
        "sigPSt": {"$ref": "#/definitions/sigPSt"},
        "sigTst": {"$ref": "#/definitions/sigTst"},
        "xVals": {"$ref": "#/definitions/xVals"},
        "rVals": {"$ref": "#/definitions/rVals"},
        "axVals": {"$ref": "#/definitions/axVals"},
        "arVals": {"$ref": "#/definitions/arVals"},
        "tstVD": {"$ref": "#/definitions/tstVD"},
        "arcTst": {"$ref": "#/definitions/arcTst"},
        "xRefs": {"$ref": "#/definitions/xRefs"},
        "rRefs": {"$ref": "#/definitions/rRefs"},
        "axRefs": {"$ref": "#/definitions/axRefs"},
        "arRefs": {"$ref": "#/definitions/arRefs"},
        "sigRTst": {"$ref": "#/definitions/sigRTst"},
        "rfsTst": {"$ref": "#/definitions/rfsTst"},
        "cSig": {"$ref": "rfcs/rfc7515.json#/definitions/jws"}
    },
    "minProperties": 1,
    "maxProperties": 1
}
```

The `etsiU` header parameter shall be a non-empty array.

The `etsiU` header parameter shall be incorporated as member of the `header` JSON object of the JSON Web Signature.

NOTE 6: The json schema file `rfc7515.json`, referenced within the `etsiUClearInstance`, is distributed by ETSI within subfolder `rfcs`. See clause B.1 for details on the location of the JSON schema files.

NOTE 7: The `header` member is the place reserved by IETF RFC 7515 [2] for unsigned header parameters in JSON Web Signatures. Clause 3.2 of IETF RFC 7515 [2] leaves its content open. The present document suitably profiles its contents.

NOTE 8: The `cSig` member, specified in clause 5.3.2 of the present document, is a countersignature of the JAdES signature. Its inner structure is defined in a separate JSON schema file whose details appear in clause B.1 of the present document, which is provided for helping implementations to validate the inner structure of JSON Web Signatures.

The content of any element of the `etsiU` array shall be either an unsigned JSON value in clear (clear JSON incorporation), or its base64url encoding (base64url incorporation).

The array shall not contain JSON values in clear in some positions, and base64url encoded unsigned JSON values in others. Either all of them shall be incorporated in clear or shall be incorporated base64url encoded.

The `etsiU` header parameter should be the only header parameter incorporated to the JWS Unprotected Header. Any unprotected JSON value that is not specified in the present document should be incorporated as an element of the `etsiU` header parameter.

NOTE 9: Adding these components into the `etsiU` header parameter allows to properly secure them in the long-term using `arcTst`.

If the `etsiU` header parameter contains JSON values in clear, instances of `tstContainer` type shall have the `canonAlg` member, except for the `sigTst` JSON object.

If the `etsiU` header parameter contains bas64url-encoded JSON values, instances of `tstContainer` type shall not have the `canonAlg` member.

5.3.2 The cSig (counter signature) JSON object

Semantics

The cSig JSON object shall contain one counter signature of the JAdES signature where cSig is incorporated. This counter signature may also be a JAdES signature.

Syntax

The cSig JSON object contains either a JSON Web Signature or a JAdES signature that signs the JWS Signature Value of the embedding JAdES signature.

One JAdES counter signature may itself be counter signed using a cSig JSON object, signing the JWS Signature Value of the first counter signature, built as described above.

NOTE: This is an alternative way of constructing arbitrarily long series of counter signatures, each one signing the JWS Signature Value of the one where it is directly embedded.

5.3.3 The sigPSt JSON object

Semantics

The sigPSt JSON object shall contain either:

- the signature policy document which is referenced in the sigPId JSON object so that the signature policy document can be used for offline and long-term validation; or
- a URI referencing a local store where the signature policy document can be retrieved.

Syntax

This JSON object shall be carried in the JWS Unprotected Header.

The sigPSt shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and are copied below for information.

```
"sigPSt": {
    "type":"object",
    "properties": {
        "sigPolDoc": {"type": "string", "contentEncoding": "base64"},
        "sigPolLocalURI": {"type": "string", "format": "uri-reference"},
        "spDSpec": {"$ref": "#/definitions/oId"}
    },
    "oneOf": [
        {
            "required": ["sigPolDoc"]
        },
        {
            "required": ["sigPolLocalURI"]
        }
    ],
    "minProperties": 1,
    "additionalProperties": false
},
```

The sigPolDoc member shall contain the base64 encoded signature policy.

The sigPolLocalURI member shall have as value the URI referencing a local store where the present document can be retrieved.

NOTE 1: Contrary to the spURI, the sigPolLocalURI points to a local file.

The spDSpec member shall identify the technical specification that defines the syntax used for producing the signature policy document.

NOTE 2: It is the responsibility of the entity incorporating the signature policy to the signature-policy-store to make sure that the correct document is securely stored.

NOTE 3: Being an unsigned JSON object, it is not protected by the digital signature. If the `sigPId` JSON object is incorporated into the signature and contains the `hashAV` member with the digest value of the signature policy document, any alteration of the signature policy document present within `sigPSt` or within a local store, would be detected by the failure of the digests comparison.

5.3.4 The `sigTst` JSON object

Semantics

The `sigTst` JSON object shall encapsulate one or more electronic time-stamps time-stamping the JWS Signature Value.

Syntax

This JSON object shall be carried in the JWS Unprotected Header.

The `sigTst` JSON object shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information.

```
"sigTst": {"$ref": "#/definitions/tstContainer"},
```

The input of the message imprint computation for the time-stamp tokens encapsulated by `sigTst` JSON object shall be the base64url-encoded JWS Signature Value.

NOTE: In a signature serialized with JWS JSON Serialization, this is the same as the content of the `signature` member.

The `sigTst` JSON object shall not contain the `canonAlg` member.

5.3.5 JSON objects for validation data values

5.3.5.1 The `xVals` JSON array

Semantics

The `xVals` JSON array:

1) Shall contain the certificate of the trust anchor, if such certificate does exist and if it is not already present within other component of the underlying JSON Web Signature. If this certificate is present within another component of the underlying JSON Web Signature, it should not be included.

2) Shall contain the CA certificates within the signing certificate path that are not already present within other component of the underlying JSON Web Signature. The certificates present within other component of the underlying JSON Web Signature should not be included.

3) Shall contain the signing certificate if it is not already present within other component of the underlying JSON Web Signature. If this certificate is present within other component of the underlying JSON Web Signature, it should not be included.

4) Shall contain certificates used to sign revocation status information (e.g. CRLs or OCSP responses) of certificates in 1), 2) and 3), and certificates within their respective certificate paths that are not present in the signature. Certificate values present within the signature, including certificate values within the revocation status information themselves should not be included.

5) Shall not contain CA certificates that pertain exclusively to the certificate paths of certificates used to sign attribute certificates or signed assertions within `srAts`, or electronic time-stamps. And

6) May contain a set of certificates used to validate any countersignature incorporated into the JAdES signature that are not present in other components of the JAdES signature or its countersignatures. This set may include any of the certificates listed in 1), 2), 3) and 4) referred to signing certificates of countersignatures instead of the signing certificate of the JAdES signature. The certificates present elsewhere in the JAdES signature or its countersignatures should not be included.

Syntax

The xVals array parameter shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information.

```
"xVals": {
    "type": "array",
    "items": {
        "type":"object",
        "properties": {
            "x509Cert": {"$ref": "#/definitions/pkiOb"},
            "otherCert": {"$ref": "#/definitions/pkiOb"}
        },
        "oneOf": [
            {
                "required": ["x509Cert"]
            },
            {
                "required": ["otherCert"]
            }
        ],
        "additionalProperties": false
    },
    "minItems": 1
},
```

An x509Cert item shall contain the base64 encoding of one DER-encoded X.509 certificate.

An otherCert item is a placeholder for potential future new formats of certificates.

5.3.5.2 The rVals JSON object

Semantics

The rVals JSON object:

1) shall contain revocation values corresponding to CA certificates within the signing certificate path if they are not already present within another component of the underlying JSON Web Signature. It shall not contain a revocation value for the trust anchor. The revocation values present within another component of the underlying JSON Web Signature should not be included;

2) shall contain a revocation value for the signing certificate if it is not already present within another component of the underlying JSON Web Signature. If it is present within another component of the underlying JSON it should not be included;

3) may contain revocation values corresponding to certificates used to sign CRLs or OCSP responses of 1) and 2) and certificates within their respective certificate paths. The revocation values present already present within another component of the underlying JSON Web Signature should not be included;

4) shall not contain revocation values corresponding to CA certificates that pertain exclusively to the certificate paths of certificates used to sign attribute certificates or signed assertions within srAts, or electronic time-stamps; and

5) may contain revocation values corresponding to the signing certificate of any countersignature incorporated into the JAdES signature as well as to the CA certificates in its certificate path. This set may include any of the revocation values listed in 1), 2) and 3) referred to signing certificates of countersignatures instead of the signing certificate of the JAdES signature. However, those revocation values among the aforementioned ones that are already present in other components of the JAdES signature should not be included.

Syntax

The rVals JSON object shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information.

```
"rVals": {
    "type": "object",
    "properties":{
        "crlVals": {
            "type": "array",
```

```
            "items": {"$ref":"#/definitions/pkiOb"},
            "minItems": 1
        },
        "ocspVals": {
            "type": "array",
            "items": {"$ref":"#/definitions/pkiOb"},
            "minItems": 1
        },
        "otherVals": {
            "type": "array",
            "items": {"type":"object"},
            "minItems": 1
        }
    },
    "minProperties": 1 ,
    "additionalProperties": false
},
```

`crlVals` member shall be a non-empty array of encoded X.509 CRLs [8].

Each item of `crlVals` array shall contain the base64 encoding of a DER-encoded X.509 CRLs [8].

If the validation data contain one or more Delta CRLs, the `crlVals` member shall contain the set of CRLs required to provide complete revocation lists.

`ocspVals` member shall be a non-empty array of encoded OCSP responses [9].

Each item of `ocspVals` array shall contain the base64 encoding of a DER-encoded `OCSPResponse` defined in IETF RFC 6960 [9].

The `otherVals` member provides a placeholder for other revocation information that can be used in the future. Their semantics and syntax are outside the scope of the present document.

5.3.5.3 The `axVals` JSON array

Semantics

The `axVals` JSON array:

1) shall contain the value(s) of the signing certificate(s) of the attribute certificate(s) and signed assertion(s) incorporated into the XAdES signature;

2) shall contain, if not present within the signature, the value(s) of the certificate(s) for the trust anchor(s) if such certificates exist, and the CA certificate values within path of the signing certificate(s) of the attribute certificate(s) and signed assertion(s) incorporated into the XAdES signature. Certificate values present within the signature should not be included; and

3) may contain the certificate values used to sign CRLs or OCSP responses and the certificates values within their respective certificate paths, used for validating the signing certificate(s) of the attribute certificate(s) and signed assertion(s) incorporated into the XAdES signature. Certificate values present within the signature, including certificate values within the revocation status information themselves should not be included.

Syntax

The `axVals` JSON array shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information.

```
"axVals": {"$ref": "#/definitions/xVals"},
```

5.3.5.4 The `arVals` JSON object

Semantics

The `arVals` JSON object:

1) shall contain the revocation value(s) of the certificate(s) that sign the attribute certificate(s) and signed assertion(s) incorporated into the XAdES signature;

2) shall contain, if not incorporated into the signature, the revocation values corresponding to CA certificates within the path(s) of the signing certificate(s) of the attribute certificate(s) and signed assertion(s) incorporated into the XAdES signature. It shall not contain revocation values for the trust anchors. Values already incorporated into the signature should not be included; and

3) may contain the revocation values on certificates used to sign CRLs or OCSP responses and certificates within their respective certificate paths, which are used for validating the signing certificate(s) of the attribute certificate(s) and signed assertion(s) incorporated into the XAdES signature. Revocation values already incorporated into the signature should not be included.

Syntax

The `arVals` JSON object shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information.

```
"arVals": {"$ref": "#/definitions/rVals"},
```

If the validation data contain one or more Delta CRLs, this JSON object shall include the set of CRLs required to provide complete revocation lists.

5.3.6 JSON values for long term availability and integrity of validation material

5.3.6.1 The `tstVD` JSON object

Semantics

The `tstVD` JSON object shall be a container for validation data required for carrying a full verification of the electronic time-stamps embedded within any of the different electronic time-stamp container JSON objects defined in the present document.

The `tstVD` JSON object shall allow incorporating certificate values.

The `tstVD` JSON object shall allow incorporating revocation values.

Syntax

The tstVD JSON object shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information.

```
"tstVD": {
    "type": "object",
    "properties": {
        "xVals": {"$ref": "#/definitions/xVals"},
        "rVals": {"$ref": "#/definitions/rVals"}
    },
    "minProperties": 1 ,
    "additionalProperties": false
},
```

The `xVals` member shall contain certificates used in the full verification of electronic time-stamps.

The `xVals` member may contain all the certificates required for a full verification of the electronic time-stamps.

The `xVals` member may also contain only some of the certificate values if the rest are present elsewhere in the JAdES signature (for instance within the electronic time-stamp itself, or in other `tstVD` created for other electronic time-stamps).

The `rVals` member shall contain revocation values used in the full verification of electronic time-stamps embedded in one JAdES time-stamp container.

The `rVals` member may contain all the revocation values required for a full verification of the electronic time-stamps.

The rVals member may also contain only some of the revocation values if the rest are present elsewhere in the JAdES signature (for instance within the electronic time-stamp itself, or in other tstVD created for other electronic time-stamps).

If the tstVD JSON object contains validation data for time-stamp tokens encapsulated in the adoTst header parameter then it shall be added at the beginning of the array within the etsiU header parameter.

NOTE: The incorporation of the aforementioned tstVD JSON object cannot take place after the incorporation of the first arcTst JSON object, as this would break the verification of its message imprint.

If the tstVD JSON object contains validation data for time-stamp tokens that are encapsulated in a JSON object different than the adoTst header parameter, then it shall be added in the array of the etsiU header parameter immediately after the item containing the aforementioned JSON object containing the electronic time-stamp.

5.3.6.2 The arcTst JSON object

5.3.6.2.1 Semantics and syntax

Semantics

The arcTst JSON object shall encapsulate electronic time-stamps computed on the JWS Payload, the JWS Protected Header, the JAdES Signature Value, and the etsiU JSON array within the JWS Unprotected Header at the time of generating each electronic time-stamp.

NOTE 1: The purpose of this JSON object is to tackle the long-term availability and integrity of the validation material.

NOTE 2: As it has been anticipated in clause 4 any header parameter different than etsiU JSON array present within the JWS Unprotected Header is not protected by the time-stamps encapsulated by this JSON object.

Syntax

The arcTst JSON object shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information.

```
"arcTst": {"$ref": "#/definitions/tstContainer"},
```

If the JAdES signature incorporates a cSig JSON object, all the required material for conducting the validation of the counter-signature shall be incorporated into the JAdES signature before generating the first arcTst JSON object. This may be done within the counter-signature itself or within the containers available within the counter-signed JAdES signature.

The contents of the cSig JSON object should not be changed, once it has been time-stamped by the arcTst.

NOTE 3: If a cSig JSON object is time-stamped by the arcTst, any ulterior change of its contents (by addition of unsigned JSON values if the countersignature is a JAdES signature, for instance) would make the validation of the arcTst fail.

The tstContainer member shall be as specified in clause 5.4.3.3 of the present document.

5.3.6.2.2 Computation of message-imprint

5.3.6.2.2.1 Processing

The message imprint computation input shall be an octet stream resulting from the concatenation of the components in the order they are listed below:

1) If the sigD header parameter is absent then:

   a) If the b64 header parameter specified in clause 3 of IETF RFC 7797 [14] is present and set to "false" then concatenate the JWS Payload value.

b) If the `b64` header parameter specified in clause 3 of IETF RFC 7797 [14] is present and set to `"true"`, OR it is absent, then concatenate the base64url-encoded JWS Payload.

2) If the `sigD` header parameter is present:

a) If the value of its `mId` member is "`http://uri.etsi.org/19182/HttpHeaders`" then concatenate the bytes resulting from processing the contents of its `pars` member as specified in clause 5.2.8.2 of the present document except the "`Digest`" string element. The processing of the "`Digest`" string element in the `pars` array shall consist in retrieving the bytes of the body of the HTTP message.

NOTE 1: The rationale for this is that the body of an HTTP message is signed indirectly: the JWS signature value is computed, not on the its value, but on its digest value computed with a certain digest algorithm. Therefore, in order to protect the HTTP body against the risk of the digest value becoming weak, the input to the timestamp token's message imprint computation, should contain the digest value of the HTTP body computed with a strong digest algorithm, which may be different from the initial one if this is expected to become weak soon.

b) Else if the value of its `mId` member is "`http://uri.etsi.org/19182/ObjectIdByURI`" or "`http://uri.etsi.org/19182/ObjectIdByURIHash`" then concatenate the bytes resulting from processing the contents of its `pars` member as specified in clause 5.2.8.3.2 of the present document.

NOTE 2: The rationale for applying the processing specified in clause 5.2.8.3.2 of the present document to the case of the mechanism identified by "`http://uri.etsi.org/19182/ObjectIdByURIHash`" is the fact that this is an indirect signing mechanism, i.e. based on signing digest values of data objects, instead the data objects themselves. Time-stamping not the digest values but the retrieved data objects, protects against future weaknesses of the digest algorithms used in `sigD`.

3) The character '.'.

4) The value of the JWS Protected Header, base64url encoded, followed by the character '.'.

5) The value of the JAdES Signature Value, base64url encoded.

6) The character '.'.

7) If the elements of the `etsiU` array appear as the base64url encodings of the unsigned components, then proceed as specified in clause 5.3.6.2.2.2 of the present document. If the elements of the `etsiU` array appear as clear instances of unsigned components, then proceed as specified in clause 5.3.6.2.2.3 of the present document.

5.3.6.2.2.2 Contribution of `etsiU` with base64url incorporation

The contribution of the `etsiU` array when its elements contain the base64url encodings of the unsigned components, shall be generated as follows:

1) Perform the following steps:

a) the `xVals` JSON array shall be incorporated, base64url encoded, into the signature if it is not already present and the signature misses some of the certificates listed in clause 5.3.5.1 that are required to validate the JAdES signature;

b) the `rVals` JSON object shall be incorporated, base64url encoded, into the signature if it is not already present and the signature misses some of the revocation data listed in clause 5.3.5.2 that are required to validate the JAdES signature;

c) the `axVals` JSON array shall be incorporated, base64url encoded, into the signature if not already present and the following conditions are true: attribute certificate(s) or signed assertions have been incorporated into the signature, and the signature misses some certificates required for their validation; and

d) the `arVals` JSON object shall be incorporated, base64url encoded, into the signature if not already present and the following conditions are true: attribute certificates or signed assertions have been incorporated into the signature, and the signature misses some revocation values required for their validation.

2) Take the components of the etsiU array in the order they appear within the array, and concatenate them to the final octet stream.

5.3.6.2.2.3 Contribution of etsiU with clear JSON incorporation

The contribution of the `etsiU` array when its elements contain clear instances of unsigned components, shall be generated as follows:

1) Perform the following steps:

a) the `xVals` JSON array shall be canonicalized and incorporated, in clear JSON, into the signature if it is not already present and the signature misses some of the certificates listed in clause 5.3.5.1 that are required to validate the JAdES signature;

b) the `rVals` JSON object shall be canonicalized and incorporated, in clear JSON, into the signature if it is not already present and the signature misses some of the revocation data listed in clause 5.3.5.2 that are required to validate the JAdES signature;

c) the `axVals` JSON array shall be canonicalized and incorporated, in clear JSON, into the signature if not already present and the following conditions are true: attribute certificate(s) or signed assertions have been incorporated into the signature, and the signature misses some certificates required for their validation; and

d) the `arVals` JSON object shall be canonicalized and incorporated, in clear JSON, into the signature if not already present and the following conditions are true: attribute certificates or signed assertions have been incorporated into the signature, and the signature misses some revocation values required for their validation.

2) Take the components of the `etsiU` array in the order they appear within the array, canonicalize each one of them using the canonicalization algorithm identified in `canonAlg` member, and concatenate each resulting octet stream to the final octet stream.

5.4 Generally useful syntax

5.4.1 The `oId` data type

Semantics

Instances of `oId` data type shall contain a unique and permanent identifier of one data object.

Instances of `oId` data type may contain a textual description of the nature of the data object qualified by the instance of the `oId` data type.

Instances of `oId` data type may contain a number of references to documents where additional information about the nature of the data object qualified by the instance of the `bjectId` data type, can be found.

Syntax

The `oId` shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information.

```
"oId": {
    "type": "object",
    "properties": {
        "id": {"type": "string", "format": "uri"},
        "desc": {"type": "string"},
        "docRefs":{
            "type": "array",
            "items": {"type": "string", "format": "uri"},
```

```
            "minItems": 1
        }
    },
    "required": ["id"],
    "additionalProperties": false
},
```

The `id` member shall contain a permanent identifier. Once the identifier is assigned, it shall not be re-assigned again.

The value of the `id` member shall be an URI. If the identifier of the object is an OID then the value of this member shall be encoded as an URN as specified by the IETF RFC 3061 [3].

If both an OID and a URI exist identifying one object, the URI value should be used in the `id` member.

The `desc` member shall contain an informal text describing the object.

The `docRefs` member shall contain an arbitrary number of URI values pointing to further explanatory documentation of the data object identified by the instance of this type.

5.4.2 The `pkiOb` data type

Semantics

The `pkiOb` data type shall be used to incorporate PKI objects, which can be non-JSON encoded, into the JAdES signature.

NOTE: Examples of such PKI objects, include X.509 certificates and revocation lists, OCSP responses, attribute certificates, and electronic time-stamps.

Syntax

The `pkiOb` type shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information.

```
"pkiOb": {
    "type": "object",
    "properties":{
        "encoding": {"type": "string", "format": "uri"},
        "specRef": {"type": "string"},
        "val": {"type": "string", "contentEncoding" : "base64"}
    },
    "required": ["val"],
    "additionalProperties": false
},
```

The content of this data type shall be the PKI object, base64 encoded.

The `encoding` member's value shall be a URI identifying the encoding used in the original PKI object. The values for the URI shall be one of the values defined in clause 5.1.3 of ETSI EN 319 132-1 [4].

If the `encoding` member is not present, then the contents of `val` member shall be the result of base64 encoding the DER-encoded ASN.1 data.

5.4.3 Container for electronic time-stamps

5.4.3.1 Introduction

The present document specifies JSON objects that act as electronic time-stamps containers.

Electronic time-stamps within the aforementioned containers may time-stamp isolated components or concatenations of several components of JAdES signatures.

This clause specifies a JSON type for containers of electronic time-stamps.

5.4.3.2 Containers for electronic time-stamps

Below follows the list of the electronic time-stamps containers that are defined by the present document:

- Containers for electronic time-stamps proving that the JWS Payload has been created before certain time instant: `adoTst`.
- Container for electronic time-stamps proving that the signature value has been computed before a certain time instant (to protect against repudiation in case of a key compromise): `sigTst`.
- Container for electronic time-stamps time-stamping the signature and validation data values, for providing long term JAdES signatures: `arcTst`.
- Containers for electronic time-stamps on components that contain references to validation data, namely: `rfsTst` and `sigRTst`. (specified in clause A.1.5 of the present document).

5.4.3.3 The `tstContainer` type

Semantics

The `tstContainer` type shall:

- allow encapsulating IETF RFC 3161 [7] electronic time-stamps as well as electronic time-stamps in other formats;
- provide means for managing electronic time-stamps computed on a concatenation of JAdES components (including detached JWS Payload); and
- allow encapsulating more than one electronic time-stamp generated for the same set of JAdES components (including detached JWS Payload), each one issued by different TSAs, for instance.

Syntax

The `tstContainer` type shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information.

```
"tstContainer":{
    "type": "object",
    "properties": {
        "canonAlg": {"type": "string", "format": "uri"},
        "tstTokens": {
            "type": "array",
            "items": {"$ref":"#/definitions/tstToken"},
            "minItems": 1
        }
    },
    "required": ["tstTokens"],
    "additionalProperties": false
},

"tstToken":{
    "type": "object",
    "properties":{
        "type": {"type": "string"},
        "encoding": {"type": "string", "format": "uri"},
        "specRef": {"type": "string"},
        "val": {"type": "string", "contentEncoding" : "base64"}
    },
    "required": ["val"],
    "additionalProperties": false
},
```

The `tstContainer`'s `tstTokens` member shall contain a non-empty array of JSON objects each one encapsulating one electronic time-stamp token.

The `tstToken`'s `type` member shall identify the type of the time-stamp token. For IETF RFC 3161 [7] time-stamp tokens this member shall not be present.

The `tstToken`'s `encoding` member shall be an URI and shall identify the encoding used for the time-stamp token. For IETF RFC 3161 [7] time-stamp tokens this member shall not be present.

The `tstToken`'s `specRef` member shall identify the technical specification that has defined the used time-stamp token. For IETF RFC 3161 [7] time-stamp tokens this member shall not be present.

Finally the `tstToken`'s `val` member shall contain the base64 encoding of the electronic time-stamp token itself. For IETF RFC 3161 [7] time-stamp tokens this member shall contain the base64 encoding of the DER-encoded electronic time-stamp token.

In JAdES signatures, the containers of time-stamp tokens time-stamping components within the `etsiU` unsigned header parameter, implicitly identify what components are time-stamped and how they contribute to the input of the message imprint's computation. No further information in the time-stamp token container is required.

NOTE: This is because all the components of a JAdES signature are placed within JAdES signature itself.

The `tstContainer's canonAlg` member shall contain the identifier of a canonicalization algorithm.

If the `tstContainer's canonAlg` member is present, then the bytes concatenated for building the time-stamp's message imprint input, shall be the bytes resulting from applying the canonicalization algorithm to all the time-stamped JAdES components.

If the `tstContainer's canonAlg` is absent then the bytes concatenated for building the time-stamp's message imprint input, shall be the bytes of each of the time-stamped JAdES components themselves.

6 JAdES baseline signatures

6.1 Signature levels

Clause 6 defines four levels of JAdES baseline signatures, intended to facilitate interoperability and to encompass the life cycle of JAdES signature, namely:

a) B-B level provides requirements for the incorporation of signed header parameters and some unsigned components within the `etsiU` unsigned header parameter when the signature is generated.

b) B-T level provides requirements for the generation and inclusion, for an existing signature, of a trusted token proving that the signature itself actually existed at a certain date and time.

c) B-LT level provides requirements for the incorporation of all the material required for validating the signature in the signature document. This level aims to tackle the long-term availability of the validation material.

d) B-LTA level provides requirements for the incorporation of electronic time-stamps that allow validation of the signature long time after its generation. This level aims to tackle the long-term availability and integrity of the validation material.

NOTE 1: ETSI TR 119 100 [i.6] provides a description on the life-cycle of a signature and the rationales on which level is suitable in which situation.

NOTE 2: The levels c) to d) are appropriate where the technical validity of signature needs to be preserved for a period of time after signature creation where certificate expiration, revocation and/or algorithm obsolescence is of concern. The specific level applicable depends on the context and use case.

NOTE 3: B-LTA level targets long term availability and integrity of the validation material of digital signatures over long term. The B-LTA level can help to validate the signature beyond many events that limit its validity (for instance, the weakness of used cryptographic algorithms, or expiration of validation data). The use of B-LTA level is considered an appropriate preservation and transmission technique for signed data.

NOTE 4: Conformance to B-LT level, when combined with appropriate additional preservation techniques tackling the long term availability and integrity of the validation material is sufficient to allow validation of the signature long time after its generation. The assessment of the effectiveness of preservation techniques for signed data other than implementing the B-LTA level are out of the scope of the present document. The reader is advised to consider legal instruments in force and/or other standards (for example ETSI TS 101 533-1 [i.9] or IETF RFC 4998 [i.10]) that can indicate other preservation techniques. Annex C defines what needs to be taken into account when using other techniques for long term availability and integrity of validation data and incorporating a new component in the `etsiU` unsigned header parameter derived from these techniques into the signature.

6.2 General requirements

6.2.1 Algorithm requirements

The algorithms and key lengths used to generate and augment digital signatures should be as specified in ETSI TS 119 312 [21].

NOTE: Cryptographic suites recommendations defined in ETSI TS 119 312 [21] can be superseded by national recommendations.

In addition, MD5 algorithm shall not be used as digest algorithm.

6.2.2 Notation for requirements

The present clause describes the notation used for defining the requirements of the different JAdES signature levels.

The requirements on the header parameters and certain other signature's components for each JAdES signature level are expressed in Table C.1. A row in the table either specifies requirements for a header parameter, other signature's component, or a service.

A service can be provided by different header parameters, by other signature's components, or by other mechanisms (service provision options hereinafter). In these cases, the specification of the requirements for a service is provided by three or more rows. The first row contains the requirements of the service. The requirements for the header parameters, other signature's components, and/or mechanisms used to provide the service are stated in the following rows.

Table 1 contains 8 columns. Below follows a detailed explanation of their meanings and contents:

1) Column "Header parameters/Elements in `etsiU` unsigned header parameter/Services":

   a) In the case where the cell identifies a Service, the cell content starts with the keyword "Service" followed by the name of the service.

   b) In the case where the header parameter or other signature's component provides a service, this cell contains "SPO" (for Service Provision Option), followed by the name of the header parameter or the other signature's component.

   c) Otherwise, this cell contains the name of the header parameter or the other signature's component.

2) Column "Presence in B-B level": This cell contains the specification of the presence of the header parameter or other signature's component, or the provision of a service, for JAdES-B-B signatures.

3) Column "Presence in B-T level": This cell contains the specification of the presence of the header parameter or other signature's component, or the provision of a service, for JAdES-B-T signatures.

4) Column "Presence in B-LT level": This cell contains the specification of the presence of the header parameter or other signature's component, or the provision of a service, for JAdES-B-LT signatures.

5) Column "Presence in B-LTA level": This cell contains the specification of the presence of the header parameter or other signature's component, or the provision of a service, for JAdES-B-LTA signatures. Below follow the values that can appear in columns "Presence in B-B", "Presence in B-T", "Presence in B-LT", and "Presence in B-LTA":

   - "shall be present": means that the header parameter or signature's component shall be incorporated to the signature, and shall be as specified in the document referenced in column "References", further profiled with the additional requirements referenced in column "Requirements", and with the cardinality indicated in column "Cardinality".

   - "shall not be present": means that the header parameter or signature's component shall not be incorporated to the signature.

   - "may be present": means that the header parameter or signature's component may be incorporated to the signature, and shall be as specified in the document referenced in column "References", further profiled with the additional requirements referenced in column "Requirements", and with the cardinality indicated in column "Cardinality".

   - "shall be provided": means that the service identified in the first column of the row shall be provided as further specified in the SPO-related rows. This value only appears in rows that contain requirements for services. It does not appear in rows that contain requirements for header parameters or signature's components.

   - "conditioned presence": means that the incorporation to the signature of the item identified in the first column is conditioned as per the requirements referenced in column "Requirements" and requirements in specifications and clauses referenced by column "References", with the cardinality indicated in column "Cardinality".

   - "*": means that the header parameter or signature's component (service) identified in the first column should not be incorporated to the signature (provided) in the corresponding level. Upper signature levels may specify other requirements.

NOTE: Incorporating an unsigned component within the `etsiU` header parameter that is marked with a "*" into a signature can lead to cases where a higher level cannot be achieved, except by removing the corresponding component.

6) Column "Cardinality": This cell indicates the cardinality of the header parameter or other signature's component. If the cardinality is the same for all the levels, only the values listed below appear. Otherwise the content specifies the cardinality for each level. See the example at the end of the present clause showing this situation. Below follows the values indicating the cardinality:

   - **0:** The signature shall not incorporate any instance of the header parameter or the signature's component.

   - **1:** The signature shall incorporate exactly one instance of the header parameter or the signature's component.

   - **0 or 1:** The signature shall incorporate zero or one instance of the header parameter or the signature's component.

   - **≥ 0:** The signature shall incorporate zero or more instances of the header parameter or the signature's component.

   - ≥ **1:** The signature shall incorporate one or more instances of the header parameter or the signature's component.

7) Column "References": This shall contain either the number of the clause specifying the header parameter in the present document, or a reference to the document and clause that specifies the other signature's component.

8) Column "Additional requirements and notes": This cell contains numbers referencing notes and/or letters referencing additional requirements on the header parameter or the other signature's component. Both notes and additional requirements are listed in Table 1.

6.3 Requirements on JAdES components and services

The four JAdES signature levels specified in the present clause shall be built as specified in clause 4 of the present document.

Table 1 shows the presence and cardinality requirements on the signature header parameters, other components, and services indicated in the first column for the four JAdES baseline signature levels, namely: JAdES-B-B, JAdES-B-T, JAdES-B-LT, and JAdES-B-LTA). Additional requirements are detailed below the table suitably labelled with the letter indicated in the last column.

NOTE 1: JAdES-B-B signatures that incorporate only the header parameters and other components that are mandatory in Table C.1, and that implement the mandatory requirements, contain the lowest number of header parameters and other components, with the consequent benefits for interoperability.

In JAdES baseline signatures the components that act as electronic time-stamps containers shall encapsulate only IETF RFC 3161 [7] updated by IETF RFC 5816 [10] time-stamp tokens.

Any header parameter specified in IETF RFC 7515 [2] or IETF RFC 7797 [14], and not further profiled in clause 5.1, may be present (cardinality of 0 or 1) in the four levels defined in Table 1.

Table 1: Requirements for JAdES-B-B, JAdES-B-T, JAdES-B-LT, and JAdES-B-LTA signatures

Header parameters/Elements in etsiU unsigned header parameter/Services	Presence in B-B level	Presence in B-T level	Presence in B-LT level	Presence in B-LTA level	Cardinality	References	Additional requirements and notes
alg	shall be present	shall be present	shall be present	shall be present	1	Clause 5.1.2	
cty	conditioned presence	conditioned presence	conditioned presence	conditioned presence	0 or 1	Clause 5.1.3	2
kid	may be present	may be present	may be present	may be present	0 or 1	Clause 5.1.4	
x5u	may be present	may be present	may be present	may be present	0 or 1	Clause 5.1.5	
x5c	Conditioned presence	Conditioned presence	Conditioned presence	Conditioned presence	0 or 1	Clause 5.1.8	3
crit	Conditioned presence	Conditioned presence	Conditioned presence	Conditioned presence		Clause 5.1.9	4
sigT	shall be present	shall be present	shall be present	shall be present	1	Clause 5.2.1	a
Service: signing a reference of the signing certificate	Conditioned presence	Conditioned presence	Conditioned presence	Conditioned presence	1		3
SPO: x5t#256	conditioned presence	conditioned presence	conditioned presence	conditioned presence	0 or 1	Clause 5.1.7	
SPO: x5t#o	conditioned presence	conditioned presence	conditioned presence	conditioned presence	0 or 1	Clause 5.2.2	
SPO: sigX5ts	conditioned presence	conditioned presence	conditioned presence	conditioned presence	0 or 1	Clause 5.2.2	
sigD	may be present	may be present	may be present	may be present	0 or 1	Clause 5.2.8	
srAts	may be present	may be present	may be present	may be present	0 or 1	Clause 5.2.5	
srCms	may be present	may be present	may be present	may be present	≥ 0	Clause 5.2.3	
sigPl	may be present	may be present	may be present	may be present	0 or 1	Clause 5.2.4	
sigPId	may be present	may be present	may be present	may be present	0 or 1	Clause 5.2.7	
cSig	may be present	may be present	may be present	may be present	≥ 0	Clause 5.3.2	
adoTst	may be present	may be present	may be present	may be present	≥ 0	Clause 5.3.3	5
sigPSt	may be present	may be present	may be present	may be present	0 or 1	Clause 5.3.3	b
sigTst	*	shall be present	shall be present	shall be present	B-B: ≥ 0 B-T, B-LT, B-LTA: ≥ 1	Clause 5.3.4	c, d 5
xVals	*	*	conditioned presence	conditioned presence	0 or 1	Clause 5.3.5.1	e, 6
xRefs	*	*	shall not be present	shall not be present	B-B, B-T: 0 or 1 B-LT, B-LTA: 0	Clause A.1.1	f, g
axVals	*	*	conditioned presence	conditioned presence	0 or 1	Clause 5.3.5.3	e, 7
axRefs	*	*	shall not be present	shall not be present	B-B, B-T: 0 or 1 B-LT, B-LTA: 0	Clause A.1.3	f, g, h

Header parameters/Elements in `etsiU` unsigned header parameter/Services	Presence in B-B level	Presence in B-T level	Presence in B-LT level	Presence in B-LTA level	Cardinality	References	Additional requirements and notes
`rVals`	*	*	conditioned presence	conditioned presence	0 or 1	Clause 5.3.5.2	i, 8
`rRefs`	*	*	shall not be present	shall not be present	B-B, B-T: 0 or 1 B-LT, B-LTA: 0	Clause A.1.2	
`arVals`	*	*	conditioned presence	conditioned presence	0 or 1	Clause 5.3.5.4	i, 9
`arRefs`	*	*	shall not be present	shall not be present	B-B, B-T: 0 or 1 B-LT, B-LTA: 0	Clause A.1.4	h
`sigRTst`	*	*	shall not be present	shall not be present	B-B, B-T: ≥ 0 B-LT, B-LTA: 0	Clause A.1.5.1	
`rfsTst`	*	*	shall not be present	shall not be present	B-B, B-T: ≥ 0 B-LT, B-LTA: 0	Clause A.1.5.2	
Service: Incorporation of validation data for electronic time-stamps	*	*	shall be provided	shall be provided	-	-	j, k 10
SPO: `tstVD`	*	*	conditioned presence	conditioned presence	≥ 0	Clause 5.3.6.1	
SPO: certificate and revocation values embedded in the electronic time-stamp itself	*	*	conditioned presence	conditioned presence	≥ 0	-	
`arcTst`	*	*	*	shall be present	≥ 1	Clause 5.3.6.2	l, m

Additional requirements:

a) Requirement for `sigT`. The generator shall include the claimed UTC time when the signature was generated as content of the `sigT` header parameter.

b) Requirement for `sigPSt`. This header parameter may be incorporated into the JAdES signature only if the `sigPId` is also incorporated and it contains the `hashAV` member with the digest value of the signature policy document. Otherwise the `sigPSt` shall not be incorporated into the JAdES signature.

c) Requirement for `sigTst`. Each `sigTst` shall contain only one electronic time-stamp.

d) Requirement for `sigTst`. The electronic time-stamps encapsulated within the `sigTst` shall be created before the signing certificate has been revoked or has expired.

e) Requirement for `xVals` and `axVals`. Duplication of certificate values within the signature should be avoided.

f) Requirement for `xRefs` and `axRefs`. The references to certificates should not include the `kid` member.

g) Requirement for `xRefs` and `axRefs`. The references to certificates shall not include the `x5u` member.

h) Requirement for `axRefs` and `arRefs`. The `axRefs` and `arRefs` may be used when a at least an attribute certificate or a signed assertion is incorporated into the JAdES signature. Otherwise, `axRefs` and `arRefs` shall not be used.

i) Requirement for `rVals` and `arVals`. Duplication of revocation values within the signature should be avoided.

j) Requirement for service "incorporation of validation data for electronic time-stamps". The validation data for electronic time-stamps shall be present within the `tstVD` or embedded in the electronic time-stamp itself.

k) Requirement for service "incorporation of validation data for electronic time-stamps". The validation data for electronic time-stamps should be included within `tstVD`.

l) Requirement for `arcTst`. Each `arcTst` may contain more than one electronic time-stamp issued by different TSAs.

m) Requirement for `arcTst`. Before generating and incorporating a new `arcTst`, all the validation material required for validating the JAdES signature shall be included. This validation material shall include all the certificates and all certificate status information (like CRLs or OCSP responses) required for:

- validating the signing certificate;
- validating the signing certificate of any countersignature incorporated into the signature;
- validating any attribute certificate or signed assertion present in the signature; and
- validating the signing certificate of any previous electronic time-stamp already incorporated into the signature within any JAdES electronic time-stamp container component (including any `arcTst`).

NOTE 2: On `cty`, and `ctys` within `sigD`: see clauses 5.1.4 and 5.2.8.1 of the present document for details of their conditioned presence.

NOTE 3: On `x5c` and service "signing a reference of the signing certificate". Clause 5.1.7 specifies the conditions that decide the presence or absence of the `x5c`, `x5t#S256`, and `x5t#o` header parameters in a JAdES signature.

NOTE 4: On `crit`. Clause 5.1.9 specifies the conditions that decide the presence or absence of the `crit` header parameter in a JAdES signature.

NOTE 5: On `sigTst`, `adoTst`. Several instances of these components can be incorporated into the JAdES signature, coming from different TSAs.

NOTE 6: On `xVals`. Clause 5.3.5.1 specifies the conditions that decide the presence or absence of the `xVals` element of `etsiU` JSON array in a JAdES signature.

NOTE 7: On `axVals`. Clause 5.3.5.3 specifies the conditions that decide the presence or absence of the `axVals` element of `etsiU` JSON array in a JAdES signature.

NOTE 8: On `rVals`. Clause 5.3.5.2 specifies the conditions that decide the presence or absence of the `rVals` element of `etsiU` JSON array in a JAdES signature.

NOTE 9: On `arVals`. Clause 5.3.5.4 specifies the conditions that decide the presence or absence of the `arVals` element of `etsiU` JSON array in a JAdES signature.

NOTE 10: On service "incorporation of validation data for electronic time-stamps": the incorporation of the validation material of the electronic time-stamps ensures that the JAdES signature actually contains all the validation material needed.

Annex A (normative): Additional components Specification

A.1 Components for validation data

A.1.1 The xRefs JSON array

Semantics

The xRefs JSON array:

1) shall contain the reference to the certificate of the trust anchor if such certificate does exist, and the references to CA certificates within the signing certificate path;

2) shall not contain the reference to the signing certificate;

3) may contain references to certificates in the path of the certificates used for signing the electronic time-stamps already incorporated into the signature when the xRefs is incorporated, including references to the electronic time-stamps' signing certificates and references to certificates of trust anchors if such certificates do exist;

4) may contain references to the certificates used to sign CRLs or OCSP responses for certificates referenced by references in 1) and 3), and references to certificates within their respective certificate paths; and

5) shall not contain references to CA certificates that pertain exclusively to the certificate paths of certificates used to sign attribute certificates or signed assertions within srAts.

NOTE 1: The references to certificates exclusively used in the validation of attribute certificate or signed assertions are stored within axRefs (see clause A.1.3).

Syntax

The xRefs member shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information.

```
"x5Ids": {
    "type": "array",
    "items": {"$ref": "#/definitions/certId"},
    "minItems": 1
},

"certId":{
    "type": "object",
    "properties":{
        "digAlg": {"type": "string"},
        "digVal": {"type": "string", "contentEncoding": "base64"},
        "kid": {"type": "string", "contentEncoding" : "base64"},
        "x5u": {"type": "string", "format": "uri-reference"}
    },
    "required": ["digAlg","digVal"] ,
    "additionalProperties": false
},

"xRefs": {"$ref": "#/definitions/x5Ids"},
```

The digAlg member of shall identify the digest algorithm.

The digVal member shall contain the base64url-encoded value of the digest computed on the DER-encoded certificate.

The content of kid member shall be the base64 encoding of one DER-encoded instance of type IssuerSerial type defined in IETF RFC 5035 [5].

NOTE 2: The information in the `kid` member is only a hint, that can help to identify the certificate whose digest matches the value present in the reference. But the binding information is the digest of the certificate.

The `x5u` member shall provide an indication of where the referenced certificate can be found.

NOTE 3: It is intended that the `x5u` member is used as a hint, as implementations can have alternative ways for retrieving the referenced certificate if it is not found at the referenced place.

If at least one of the following: `xVals`, `axVals`, or the `arcTst`, is incorporated into the signature, all the certificates referenced in `xRefs` shall be present elsewhere in the signature.

A.1.2 The `rRefs` JSON object

Semantics

The `rRefs` JSON object:

1) shall contain a reference to a revocation value for the signing certificate;

2) shall contain the references to the revocation values (e.g. CRLs or OCSP values) corresponding to CA certificates within the signing certificate path. It shall not contain references to revocation values for the trust anchor;

NOTE 1: A trust anchor is by definition trusted, thus no revocation information for the trust anchor is used during the validation.

3) may contain references to revocation values (e.g. CRLs or OCSP values) corresponding to certificates in the path of signing certificates of electronic time-stamps already incorporated into the signature when the `rRefs` is incorporated. It shall not contain references to revocation values for the trust anchors of these certificates;

4) may contain references to the revocation values corresponding to certificates used to sign CRLs or OCSP responses referenced in references from 1), 2) and 3) and to certificates within their respective certificate paths; and

5) shall not contain references to the revocation values corresponding to CA certificates that pertain exclusively to the certificate paths of certificates used to sign attribute certificates or signed assertions within `srAts`.

NOTE 2: The references to revocation values exclusively used in the validation of attribute certificate or signed assertions are stored within `arRefs` (see clause A.1.4).

References within `rRefs` may be references to CRLs, OCSP responses and other type of revocation data.

Syntax

`rRefs` shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information.

```
"rRefs": {
    "type": "object",
    "properties": {
        "crlRefs": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "digAlg": {"type": "string"},
                    "digVal": {"type": "string", "contentEncoding": "base64"},
                    "crlId": {
                        "type": "object",
                        "properties": {
                            "issuer": {"type": "string", "contentEncoding" : "base64"},
                            "issueTime": {"type": "string", "format": "date-time"},
                            "number": {"type": "number"},
                            "uri": {"type": "string", "format": "uri-reference"}
                        },
                        "required": ["issuer","issueTime"],
                        "additionalProperties": false
                    }
```

```
                },
                "required": ["digAlg","digVal"] ,
                "additionalProperties": false
            },
            "minItems": 1
        },
        "ocspRefs":{
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "ocspId": {
                        "type": "object",
                        "properties": {
                            "responderId": {
                                "type": "object",
                                "properties": {
                                    "byName": {"type": "string", "contentEncoding" : "base64"},
                                    "byKey": {"type": "string", "contentEncoding" : "base64"}
                                },
                                "oneOf":[
                                    {
                                        "required": ["byName"]
                                    },
                                    {
                                        "required": ["byKey"]
                                    }
                                ],
                                "additionalProperties": false
                            },
                            "producedAt": {"type": "string", "format": "date-time"},
                            "uri": {"type": "string", "format": "uri-reference"}
                        },
                        "required": ["responderId", "producedAt"],
                        "additionalProperties": false
                    },
                    "digAlg": {"type": "string"},
                    "digVal": {"type": "string", "contentEncoding": "base64"}
                },
                "required": ["ocspId","digAlg","digVal"],
                "additionalProperties": false
            },
            "minItems": 1
        },
        "otherRefs": {
            "type": "array",
            "items": {"type":"object"},
            "minItems": 1
        }
    },
    "minProperties": 1,
    "additionalProperties": false
},
```

Empty `rRefs` shall not be incorporated.

The `crlRefs` member shall contain an array of references to CRLs.

Each item within the `CRLRefs` array shall contain one reference to one CRL.

The `digAlg` and `digVal` members of one item within the `crlRefs` array shall contain one indication of a digest algorithm, and the base64url encoding of the digest value of the DER-encoded referenced CRL, respectively.

The `crlId` member needs not to be present if the referenced CRL can be inferred from other information.

The `crlId` member of the items within the `crlRefs` array shall include the name issuer in its `issuer` member.

The value of `crlId`'s `issuer` member shall fulfil the requirements specified in IETF RFC 3494 [11] for strings representing Distinguished Names.

The `crlId` member of the items within the `crlRefs` array shall include the time when the CRL was issued in its `issueTime` member.

The `crlId` member of the items within the `crlRefs` array may include the number of the CRL in its `number` member.

NOTE 3: The `number` member is an optional hint helping to get the CRL whose digest matches the value present in the reference.

The `crlId`'s `uri` member shall indicate one place where the referenced CRL can be found.

NOTE 4: It is intended that this component be used as a hint, as implementations can have alternative ways for retrieving the referenced CRL if it is not found at the referenced place.

If one or more of the identified CRLs are a Delta CRL, this component shall include references to the set of CRLs required to provide complete revocation lists.

The `ocspRefs` member shall contain a non-empty array of references to OCSP responses.

Each item within the `ocspRefs` array shall contain one reference to one OCSP response.

The `ocspId` member of the items within the `ocspRefs` array shall include an identifier of the responder in its `responderID` member.

If the responder is identified by its name, then the `responderID`'s `byName` member shall contain the base64 encoding of the DER-encoded aforementioned name.

If the responder is identified by the digest of the server's public key computed as mandated in IETF RFC 6960 [9], then the base64 encoding of the DER-encoded of `byKey` field specified in IETF RFC 6960 [9] shall appear within the `responderID`'s `byKey` member.

The `ocspId` member of the items within the `ocspRefs` array shall include the generation time of the OCSP response in its `producedAt` member.

The value in `ocspId`'s `producedAt` member shall indicate the same time as the time indicated by the `ProducedAt` field of the referenced OCSP response.

The `ocspId`'s `uri` member shall indicate one place where the referenced OCSP response can be found.

NOTE 5: This value is not the address where the OCSP service can be reached. In addition to that, it is intended that this component be used as a hint, as implementations can have alternative ways for retrieving the referenced OCSP response if it is not found at the referenced place.

The `digAlg` and `digVal` members of the items within the `ocspRefs` array shall contain one indication of a digest algorithm, and the base64url encoding of the DER-encoded `OCSPResponse` field defined in IETF RFC 6960 [9], respectively.

References to alternative forms of validation data may be included in this component making use of the `otherRefs` member, a sequence whose items may contain any kind of information. Their semantics and syntax are outside the scope of the present document.

If at least one of the following: `rVals`, `arVals`, or the `arcTst`, is incorporated into the signature, all the revocation data referenced in `rRefs` shall be present elsewhere in the signature.

A.1.3 The `axRefs` JSON array

Semantics

The `axRefs` JSON array:

1) shall contain, if they are not present within `xRefs` or `x5t#o` header parameters, the references to the trust anchors if certificates exist for them, and the references to CA certificates within the path of the signing certificate(s) of the attribute certificate(s) and signed assertion(s) incorporated into the JAdES signature. References present within `xRefs` or `x5t#o` header parameters should not be included;

2) shall contain, if they are not present within `xRefs` or `x5t#o` header parameters, the reference(s) to the signing certificate(s) of the attribute certificate(s) and signed assertion(s) incorporated into the JAdES signature. References present within `xRefs` or `x5t#o` header parameters should not be included; and

3) may contain references to the certificates used to sign CRLs or OCSP responses and certificates within their respective certificate paths, which are used for validating the signing certificate(s) of the attribute certificate(s) and signed assertion(s) incorporated into the JAdES signature. References present within `xRefs` or `x5t#o` should not be included.

Syntax

`axRefs` shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information.

```
"axRefs": {"$ref": "#/definitions/x5Ids"},
```

If at least one of the following: `xVals`, `axVals`, or the `arcTst`, is incorporated into the signature, all the certificates referenced in `axRefs` shall be present elsewhere in the signature.

NOTE 1: The information in the `kid` member is only a hint, that can help to identify the certificate whose digest matches the value present in the reference. But the binding information is the digest of the certificate.

NOTE 2: It is intended that the `x5u` member is used as a hint, as implementations can have alternative ways for retrieving the referenced certificate if it is not found at the referenced place.

A.1.4 The `arRefs` JSON object

Semantics

The `arRefs` JSON object:

1) Shall contain, if they are not present within `rRefs`, the references to the revocation values corresponding to CA certificates within the path(s) of the signing certificate(s) of the attribute certificate(s) and signed assertion(s) incorporated into the JAdES signature. It shall not contain a revocation value for the trust anchors. References present within `rRefs` should not be included.

NOTE: A trust anchor is by definition trusted, thus no revocation information for the trust anchor is used during the validation.

2) Shall contain, if they are not present within the `rRefs`, the references to the revocation value(s) for the signing certificate(s) of the attribute certificate(s) and signed assertion(s) incorporated into the JAdES signature. References present within `rRefs` should not be included. And

3) May contain references to the revocation values on certificates used to sign CRLs or OCSP responses and certificates within their respective certificate paths, which are used for validating the signing certificate(s) of the attribute certificate(s) and signed assertion(s) incorporated into the JAdES signature. References present within `rRefs` component should not be included.

Syntax

`arRefs` shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information.

```
"arRefs": {"$ref": "#/definitions/rRefs"},
```

If one or more of the identified CRLs are a Delta CRL, this component shall include references to the set of CRLs required to provide complete revocation lists.

If at least one of the following: `rVals`, `arVals`, or the `arcTst`, is incorporated into the signature, all the revocation data referenced in `arRefs` shall be present elsewhere in the signature.

A.1.5 Time-stamps on references to validation data

A.1.5.1 The `sigRTst` JSON object

A.1.5.1.1 General

Semantics

The `sigRTst` JSON object shall encapsulate electronic time-stamps on the JWS Signature Value, the signature time-stamp, if present, and the JAdES components containing references to validation data.

Syntax

The `sigRTst` JSON object shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information.

```
"sigRTst": {"$ref": "#/definitions/tstContainer"},
```

This JSON object shall contain an electronic time-stamp that time-stamps the member encapsulating the JWS Signature Value, and the following components when they are present: `sigTst`, `xRefs`, `rRefs`, `axRefs`, and `arRefs`.

If none of the following components: `xRefs`, `rRefs`, `axRefs`, and `arRefs` is present, the `sigRTst` JSON object shall not be generated.

A.1.5.1.2 Computation of the message imprint with Base64url incorporation

The message imprint computation input shall be the concatenation of the components, in the order they are listed below:

1) The value of the base64url-encoded JWS Signature Value.

NOTE: If the JAdES signature is serialized with JWS JSON Serialization, this is the value within the member `signature`.

2) The character '.'.

3) Those among the following components that appear before `sigRTst`, in their order of appearance within the `etsiU` array, base64url-encoded:

   - `sigTst` if it is present;
   - `xRefs` if it is present;
   - `rRefs` if it is present;
   - `axRefs` if it is present; and
   - `arRefs` if it is present.

A.1.5.1.3 Computation of the message imprint with JSON clear incorporation

The message imprint computation input shall be the concatenation of the components, in the order they are listed below:

1) The value of the base64url-encoded JWS Signature Value.

NOTE: If the JAdES signature is serialized with JWS JSON Serialization, this is the value within the member `signature`.

2) The character '.'.

3) Those among the following components that appear before `sigRTst`, in their order of appearance within the `etsiU` array, canonicalized using the canonicalization algorithm identified in `canonAlg` member:

   - `sigTst` if it is present;

- `xRefs` if it is present;
- `rRefs` if it is present;
- `axRefs` if it is present; and
- `arRefs` if it is present.

A.1.5.2 The `rfsTst` JSON object

A.1.5.2.1 Semantics and syntax

Semantics

The `rfsTst` JSON object shall encapsulate electronic time-stamps on the JAdES components containing references to validation data.

Syntax

The `rfsTst` JSON object shall be defined as in the JSON Schema file whose location is detailed in clause B.1, and is copied below for information.

```
"rfsTst": {"$ref": "#/definitions/tstContainer"},
```

This JSON object shall contain an electronic time-stamp that time-stamps the following JAdES components when they are present: `xRefs`, `rRefs`, `axRefs`, and `arRefs`.

If none of the aforementioned JAdES components is present, the `rfsTst` JSON object shall not be generated.

A.1.5.2.2 Computation of the message imprint with Base64url incorporation

The message imprint computation input shall be the concatenation of the components listed below, base64url encoded, in their order of appearance within the `etsiU` array:

- `xRefs` if it is present;
- `rRefs` if it is present;
- `axRefs` if it is present; and
- `arRefs` if it is present.

A.1.5.2.3 Computation of the message imprint with clear JSON incorporation

The message imprint computation input shall be the concatenation of the components listed below, canonicalized using the canonicalization algorithm identified in `canonAlg` member, in their order of appearance within the `etsiU` array:

- `xRefs` if it is present;
- `rRefs` if it is present;
- `axRefs` if it is present; and
- `arRefs` if it is present.

Annex B (normative): JSON Schema files

B.1 JSON Schema files location for JAdES components

The file at https://forge.etsi.org/rep/esi/x19_182_JAdES/raw/v1.1.1/19182-jsonSchema.json (19182-jsonSchema.json) contains the definitions of the components specified in the present document.

The file at https://forge.etsi.org/rep/esi/x19_182_JAdES/raw/v1.1.1/19182-protected-jsonSchema.json (19182-protected-jsonSchema.json) may be used by implementers to validate the conformance of the JWS Protected Header of a JAdES signature against the JSON Schema definitions within 19182-jsonSchema.json.

The file at https://forge.etsi.org/rep/esi/x19_182_JAdES/raw/v1.1.1/19182-unprotected-jsonSchema.json (19182-unprotected-jsonSchema.json) may be used by implementers to validate the conformance of the JWS Unprotected Header of a JAdES signature against the JSON Schema definitions within 19182-jsonSchema.json.

Additionally, ETSI provides additional JSON schema files, for facilitating implementers to check the structure of JWS signatures. In case of conflicts between these JSON schemas and the IETF RFC 7515 [2], IETF RFC 7517 [i.16], IETF RFC 7797 [14], IETF RFC 7515 [2], IETF RFC 7517 [i.16] and IETF RFC 7797 [14] shall take precedence.

Below follows the list of these additional JSON schema files:

- The file at https://forge.etsi.org/rep/esi/x19_182_JAdES/raw/v1.1.1/rfcs/rfc7515.json (rfc7515.json). This file contains JSON schema definitions for the structures defined in IETF RFC 7515 [2].
- The file at https://forge.etsi.org/rep/esi/x19_182_JAdES/raw/v1.1.1/rfcs/rfc7515-jws.json (rfc7515-jws.json). This file may be used by implementers for checking conformance of a JWS signature against the JSON Schema definitions in file rfc7515.json.
- The file at https://forge.etsi.org/rep/esi/x19_182_JAdES/raw/v1.1.1/rfcs/rfc7515-protected.json (rfc7515-protected.json). This file may be used by implementers for checking conformance of the JWS Protected Header of a JWS signature against the JSON Schema definitions in file rfc7515.json.
- The file at https://forge.etsi.org/rep/esi/x19_182_JAdES/raw/v1.1.1/rfcs/rfc7515-unprotected.json (rfc7515-unprotected.json). This file may be used by implementers for checking conformance of the JWS Unprotected Header of a JWS signature against the JSON Schema definitions in file rfc7515.json.
- The file at https://forge.etsi.org/rep/esi/x19_182_JAdES/raw/v1.1.1/rfcs/rfc7517.json (rfc7517.json). This file contains JSON schema definitions for the structures defined in IETF RFC 7517 [i.16].
- The file at https://forge.etsi.org/rep/esi/x19_182_JAdES/raw/v1.1.1/rfcs/rfc7797.json (rfc7797.json). This file contains JSON schema definitions for the structures defined in IETF RFC 7797 [14].

Annex C (informative): Correspondence between XAdES tags and JAdES tags

C.1 Correspondence between XAdES qualifying properties tags and JAdES component tags

Table C.1 shows the correspondence between the tags used by the XAdES qualifying properties and the tags used by the JAdES components.

Table C.1: Correspondence between XAdES and JAdES tags

XAdES tag	JAdES tag
UnsignedProperties	etsiU
SigningTime	sigT
SigningCertificateV2 (reference to the signing certificate only -for extencing semantics of x5t#256 specified in IETF RFC 7515 [2])	x5t#o
SigningCertificateV2 (references to the signing certificate and other certificates within the cert path, for mimiking XAdES and CAdES).	sigX5ts
SigaturePolicyIdentifier	sigPId
SignatureProductionPlaceV2	sigPl
SignerRoleV2	srAts
DataObjectFormat	sdF
AllDataObjectsTimeStamp	adoTst
CommitmentTypeIndication	srCms
CounterSignature	cSig
IndividualDataObjectsTimeStamp	idoTst
SignaturePolicyStore	sigPSt
SignatureTimeStamp	sigTst
OIdentifier	oId
EncapsualtedPKIDataType	pkiOb
ArchiveTimeStamp	arcTst
RefsOnlyTimeStampV2	rfsTst
SigAndRefsTimeStampV2	sigRTst
CertificateValues	xVals
RevocationValues	rVals
AttrAuthoritiesCVals	axVals
AttributeRevocationValues	arVals
TimeStampValidationData	tstVD
CompleteCertificateRefs	xRefs
RevocationRefs	rRefs
AttributeCertificateRefsV2	axRefs
AttributeRevocationRefs	arRefs

Annex D (normative): Alternative mechanisms for long term availability and integrity of validation data

There may be mechanisms to achieve long-term availability and integrity of validation data different from the ones described in clause 5.3.6.

If such a mechanism is incorporated using an unsigned component into the signature, then for this mechanism shall be specified:

1) The clear specification of the semantics and syntax of the component including its unique identifier.

2) The strategy of how this mechanism guarantees that all necessary parts of the signature are protected by this component.

3) The strategy of how to handle signatures containing components defined in the present document.

EXAMPLE: The objects defined in IETF RFC 4998 [i.10], Annex A are examples of such alternative mechanisms but they only handle points 1) and 2).

Annex E (normative): Digest algorithms identifiers for JAdES signatures

This annex defines cryptographic digest algorithms and identifiers to be used with the JAdES signatures.

The digest algorithms are the digest algorithms identified in ETSI TS 119 312 [21].

Table E.1: Identifiers for digest algorithms

Short hash function name	Identifier	References
SHA-224	S224	FIPS Publication 180-4 [22]
SHA-256	S256	FIPS Publication 180-4 [22]
SHA-384	S384	FIPS Publication 180-4 [22]
SHA-512	S512	FIPS Publication 180-4 [22]
SHA-512/256	S512-256	FIPS Publication 180-4 [22]
SHA3-256	S3-256	FIPS Publication 202 [23]
SHA3-384	S3-384	FIPS Publication 202 [23]
SHA3-512	S3-512	FIPS Publication 202 [23]

Annex F (informative): Change History

Date	Version	Information about changes
January 2020	0.0.1	Version based on previous version circulated in October 2019 at ESI 68 (with a wrong TS number) after amendments of relevant parts.
January 2020	0.0.2	Consolidated version for getting feedback from ETSI ESI members and liaised entities.
May 2020	0.0.3	Consolidated version with changes implemented as per disposition to comments for version 0.0.2.
September 2020	0.0.4	Incorporated resolutions for all the comments received for v0.0.3.
November 2020	0.0.5	Incorporated resolutions for comments received for v0.0.4.
September 2020	0.0.6	Incorporated resolutions for comments received for v0.0.5. They include some changes in JSON schema. New JSON schema files have been added to the package, some referring to JWS.
January 2021	0.0.7	Incorporated resolutions for comments to v0.0.6 received during the Remote Consensus process.

History

Document history		
V1.1.1	March 2021	Publication