Data model for remote signature applications

version 1.0.0

Contents

Foreword

This document is a work by members of the Cloud Signature Consortium, a nonprofit association founded by industry and academic organizations for building upon existing knowledge of solutions, architectures and protocols for Cloud-based Digital Signatures, also defined as “remote” Electronic Signatures.

The Cloud Signature Consortium has developed the present specification to make these solutions interoperable and suitable for uniform adoption in the global market, in particular – but not exclusively – to meet the requirements of:

- the European Union's Regulation 910/2014 on Electronic Identification and Trust Services (eIDAS) [i.1], which formally took effect on 1 July 2016, amended by Regulation 2024/1183 on the European Digital Identity Framework [i.2].

Revision history

Version	Date	Version change details
0.1.0	15/03/2025	Pre-release for early feedback
0.2.0	28/03/2025	Pre-release, refined, for public feedback
0.3.0	27/06/2025	Pre-release for early expert feedback on specific changes
0.4.0	18/07/2025	Pre-release for more expert feedback on specific changes
0.5.0	01/08/2025	Pre-release for public feedback
1.0.0	16/10/2025	Public release

Acknowledgements

This work is the result of the contributions of several individuals from the Technical Working Group of the Cloud Signature Consortium and some additional contributors.

Introduction

This specification defines data models for use in standard APIs such as those within the OAuth 2.0 [4] (henceforth: OAuth) and OpenID for Verifiable Credentials [i.6] (henceforth: OID4VC) frameworks, in a way that complements the Cloud Signature Consortium APIs.

Intellectual Property Rights

The Intellectual Property Rights Policy (IPR Policy) of the Cloud Signature Consortium is available at https://cloudsignatureconsortium.org/about-us/intellectual-property/.

Trademark notice

The Cloud Signature Consortium logo is a Registered Trademark of the Cloud Signature Consortium:
EU Trademark number 015579048.

Essential Patents

IPRs essential or potentially essential to the present document may have been declared to the Cloud Signature Consortium. The information pertaining to these essential IPRs, if any, is available on request from the Cloud Signature Consortium secretariat at info@cloudsignatureconsortium.org.

No investigation, including IPR searches, has been carried out by the Cloud Signature Consortium. No guarantee can be given as to the existence of other IPRs not referenced in the present document which are, or may be, or may become, essential to the present document.

Legal notices

The Cloud Signature Consortium seeks to promote and encourage broad and open industry adoption of its standard.

1 Scope

These data models target several use cases in which the creation of electronic signatures is distributed across multiple distributed applications. These applications include business applications, (mobile) identity wallets, and trust service applications.

This specification defines data models for:

- Requesting a signature
- Requesting a signing operation
- Authorization of a signing operation

The following are out of scope or this specification:

- Application programming interfaces (APIs).

2 Interpretation of requirement levels

The keywords “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119 [1].

3 References

3.1 Normative references

The following documents, in whole or in part, are normatively referenced in this specification and are indispensable for its application. For dated references, only the edition cited applies. For undated references (regardless if a specific version is linked or not), the latest edition of the referenced document (including any amendments or errata) applies.

[1] IETF RFC 2119: “Key words for use in RFCs to Indicate Requirement Levels”.

[2] IETF RFC 4648: “The Base16, Base32, and Base64 Data Encodings”.

[3] IETF RFC 2397: “The ‘data’ URL scheme”.

[4] IETF RFC 6749: “The OAuth 2.0 Authorization Framework”.

[5] IETF RFC 7515: “JSON Web Signature (JWS)”.

[6] IETF RFC 8259: “The JavaScript Object Notation (JSON) Data Interchange Format”.

[7] IETF RFC 9112: “HTTP/1.1”.

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

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

[10] ETSI EN 319 142-1: “Electronic Signatures and Infrastructures (ESI); PAdES digital signatures; Part 1: Building blocks and PAdES baseline signatures”.

[11] ETSI TS 119 182-1: “Electronic Signatures and Infrastructures (ESI); JAdES digital signatures; Part 1: Building blocks and JAdES baseline signatures”.

[12] IETF RFC 8017: “PKCS #1: RSA Cryptography Specifications Version 2.2”.

[13] ISO 3166-1: “Codes for the representation of names of countries and their subdivisions — Part 1: Country codes”. A list of country codes can be accessed on ISO Online Browsing Platform (OBP).

[14] IETF RFC 8610: “Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures”.

[15] ETSI EN 319 102-1 “Electronic Signatures and Trust Infrastructures (ESI); Procedures for Creation and Validation of AdES Digital Signatures; Part 1: Creation and Validation”.

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

3.2 Informative references

The following documents, in whole or in part, are informatively referenced in this specification and may be a useful contribution for its application. For dated references, only the edition cited applies. For undated references (regardless if a specific version is linked or not), the latest edition of the referenced document (including any amendments or errata) applies.

[i.1] Regulation (EU) No 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC.

[i.2] Regulation (EU) 2024/1183 of the European Parliament and of the Council of 11 April 2024 amending Regulation (EU) No 910/2014 as regards establishing the European Digital Identity Framework.

[i.3] CSC Architectures and protocols for remote signature applications.

[i.4] IETF RFC 9396: “OAuth 2.0 Rich Authorization Requests”.

[i.5] OpenID for Verifiable Presentations, Version 1.0.

[i.6] OpenID for Verifiable Credentials.

[i.7] IETF RFC 3739: “Qualified Certificates Profile”.

[i.8] void

[i.9] ETSI TS 119 001: “Electronic Signatures and Infrastructures (ESI); The framework for standardization of signatures; Definitions and abbreviations”.

[i.10] IETF RFC 5280: “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile”.

[i.11] ISO/IEC 18013-5: “Personal identification — ISO-compliant driving licence — Part 5: Mobile driving licence (mDL) application”.

[i.12] draft-ietf-oauth-sd-jwt-vc-08: “SD-JWT-based Verifiable Credentials (SD-JWT VC)”.

Note 1: The reference [i.12] will be updated with a final version before official publication.

[i.13] ETSI TS 119 312: "Electronic Signatures and Infrastructures (ESI); Cryptographic Suites".

[i.14] W3C Digital Credentials, Working Draft 10 July 2025.

[i.15] South African Electronic Communications and Transactions Act.

[i.16] OpenID Connect Core 1.0.

4 Terms, definitions and abbreviations

4.1 Terms and definitions

For the purposes of this specification, the following terms and definitions apply.

authorization server: server enabling users to authorize privileged operations.

Note 2: The authorization server is usually the endpoint described in CSC API [i.3] Section 8.4.

base64: Base64 as defined by RFC 4648 [2] Section 4, i.e. standard alphabet with padding SHALL be used. See also paragraph about Base64 in Conventions section of this document.

base64url: Denotes the URL-safe Base64 encoding as defined in RFC 4648 [2] Section 5 and further precised in RFC 7515 [5] Section 2 and Appendix C. Padding SHALL NOT be used.

digital signature: data appended to, or a cryptographic transformation of a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery e.g. by the recipient [i.9]

driving application: component that uses a signature creation application to sign a document as identified by interacting with the relying party and/or signer.

electronic signature: digital signature created by using a certificate issued to a natural person ensuring the integrity and origin of the document and the signatory commitment to the document content.

electronic seal: digital signature created by using a certificate issued to a legal person or business unit ensuring the integrity and origin of the document, without necessarily committing to the content.

identity wallet: electronic means for identification or presentation of electronic attestations of attributes, for example at an authorization server.

relying party: a party relying on a signature from the signer.

Note 3: Typically, one relying party requests the signature. The signer could be a relying party.

remote signing service provider: service provider managing a set of credentials on behalf of multiple users and allowing them to create a remote signature with a stored credential.

signature: shorthand for electronic signature or electronic seal.

signature creation application: application that accepts signer's original document and produces a signature or signed document in accordance with AdES [15].

signer's document representation: hash value of signer's formatted document. As defined in [15].

signer's original document: some document types (e.g. PDF, XML, and JSON) require formatting before SDR can be computed. signer's original document is the original document *before* any formatting.

signer's formatted document: some document types (e.g. PDF, XML, and JSON) require formatting before SDR can be computed. signer's original document is the original document *after* any formatting. SDR is the hash of signer's formatted document.

URL-encoded: encoded using the application/x-www-form-urlencoded format as defined in RFC 6749 [4] Appendix B.

4.2 Abbreviations

This section lists abbreviations used in this specification.

Authz: authorization server

DA: driving application

eIDAS: Regulation (EU) No 910/2014 [i.1] with ammendments of Regulation (EU) 2024/1183 [i.2]

OID4VC OpenID for Verifiable Credentials, see [i.6].

OpenID4VP OpenID for Verifiable Presentations, see [i.5].

RP: relying party

RSSP: remote signing service provider

SCA: signature creation application

SCD: signature creation device

SDR: signer's document representation

ECTA: South African Electronic Communications and Transactions Act [i.15]

4.3 JSON data types

The data model uses the following data types from JSON [6], unless otherwise specified.

- *Object*: a JSON object
- *Array*: a homogeneous JSON array
- *String*: a JSON string
- *Integer*: a non-negative JSON number
- *Boolean*: a JSON boolean

5 Conventions

5.1 Text conventions

This specification adopts the following text conventions to help identify various types of information.

Table 1 – Text conventions

Text convention	Example
The vertical bar (|) indicates a possible value for selection or outcome and SHALL be interpreted as "exclusive or".	YES | NO
Text in colored boxes is example code.	`POST /csc/v2/credentials/info HTTP/1.1`
Italic text indicates the name of a data component or data type.	A *documentInfo* object contains a *String* parameter named *hash*. ... the *meta* parameter in the credential query.
Inline code blocks are used for other code than names of a data component or data type.	The *String* value `"public"` or `"OTP"` can be used. ... a remotely hosted document or a `data:` URL ... ... by calling the `signatures/signHash` endpoint ...

In general, data types and data components defined in this specification use the "camelCase" notation, like the data type *documentInfo* or the data component (parameter) *authType*. In case the name contains an abbreviation, this abbreviation is written with uniform casing, like *hashAlgorithmOID* ("OID" in uppercase, because it is not the first word) and *qesRequest* ("qes" in lowercase, because it is the first word).

However, names and parameters that are defined in other standards, like those in the domain of authentication and related to OAuth, are used here in their original format to facilitate understanding and interoperability, using "snake_case", like *refresh_token*, i.e., two names separated by an underscore. This specification is self-contained and in general does not refer to any external data types other than JSON data types. When this specification refers to external data types, that reference is specified in the respective section. Therefore, no general conventions are defined for external data types.

Whenever a data type is specified as an *Object* with particular parameters or attributes, instances with this type can be referred to as "object" in lowercase. For example, a *documentInfo* object is an *Object* that conforms to the requirements specified in the *documentInfo* section.

5.2 Base64

Where possible, and not contradicted by other conventions such as code examples, we follow the recommended typesetting for references to the different Base64 variants as defined in RFC 4648 [2]. As a result, the (lowercase) usage of 'base64' or 'base64url' strictly follows the specifications provided in Terms and definitions.

For data defined by this specification, when it is required to be Base64 encoded, data SHALL be encoded and decoded as defined in RFC 4648 [2] Section 4, which defines the usage of the standard alphabet and the use of padding. To avoid JSON representation issues line breaks SHALL NOT be used within base64-encoded data.

Within this document, it is stated different (e.g. by referencing "base64url-encoding") on a specific parameter only where a relying standard other normative already defined it different.

Example: For an external container which is defined in another normative, where CSC data elements are embedded, base64url may apply for the encoding of the external container. Contained data elements defined by this document use base64 encoding. Other data elements which originate from other normative may be base64url-encoded.

Service implementations MAY additionally accept, based on auto-detection and for improved compatibility with non-compliant client applications, the base64url-encoding (with and without padding) where specification requires base64. Relying Party implementations MAY additionally accept, based on auto-detection and for improved compatibility with non-compliant service implementations, the base64url-encoding (with and without padding) where base64 is requested according to the specification.

6 Architectures and use cases

The present document is complementary to the CSC API [i.3]. The CSC API specifies interfaces for cloud services that provide signing of documents, signing of document digests, and management of credentials (keys with certificates). The CSC API supports a wide range of architectures.

The purpose of this document is to define a set of data structures for signature and signature creation requests, signing authorization, and signing responses as well as associated data for use in invocation of the CSC API, while having these data structures also usable in interactions between components surrounding the CSC APIs in more complex architectures. In particular, the data structures in this document could be used for:

- Interactions between an API client and the CSC API.
- Authorization details in an OAuth authorization flow before invoking the CSC API for performing the actual signing operation.
- Interactions between an authorization server and the signer.
- A relying party requesting a signature from the signer.

An important motivation for this document is the amendment [i.2] to the Regulation (EU) No 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market [i.1] ("eIDAS"). This regulation lays down rules for qualified signatures and the European Digital Identity Wallet. In the terminology of eIDAS, the CSC API provides interfaces for remote "signature creation applications" (`signatures/signDoc`) and remote "signature creation devices" (`signatures/signHash`).

In a signing flow a replying party (RP) requests a signer to sign a document. In some cases, the RP may be the signer.

The signer interacts with a driving application (DA) when signing. The DA could, for instance, be:

- A service provided by the relying party (RP),
- Provided by a separate service provider (e.g. a signing portal) that both the signer and RP interacts with, or
- An application running on the signers device (e.g. an identity wallet).

The DA sends a signature creation request to a signature creation application (SCA), expecting either a signed document or a signature object. The SCA handles signature formatting according to e.g. AdES (see [8], [9], [10], and [11]). The SCA could, for instance, be:

- Part of the DA, or
- Part of the signer's mobile identity wallet, or
- Provided as a remote (proxy) signing service implementing the CSC `signatures/signDoc` interface.

Signature creation is performed by a signature creation device (SCD). For remote signing the SCD is hosted by a remote signing service provider (RSSP). The SCA interacts with SCD to create the signature value. The RSSP may support either:

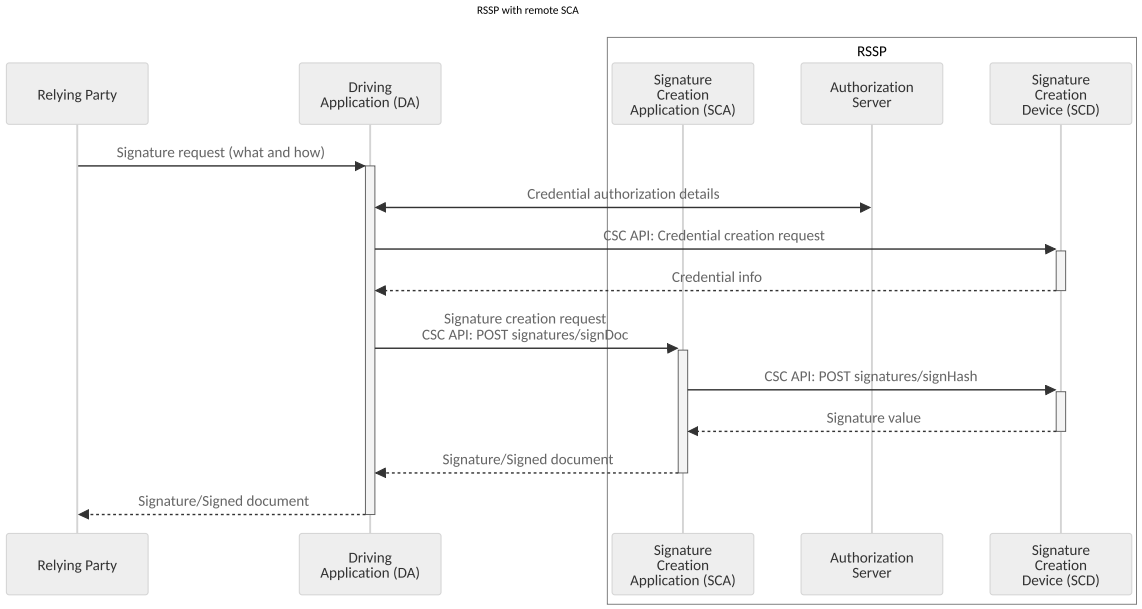
- only the `signatures/signHash` endpoint,
- the `signatures/signDoc` endpoint with access to a "signature creation device" used to sign the document, or
- the `signatures/signDoc` endpoint, implemented by calling the `signatures/signHash` endpoint from a different service provider.

When the RSSP implements the `signatures/signDoc` endpoint it is effectively a SCA. In that case, the DA may use the RSSP directly, or it may use a secondary SCA as proxy for the SCA implemented by the RSSP.

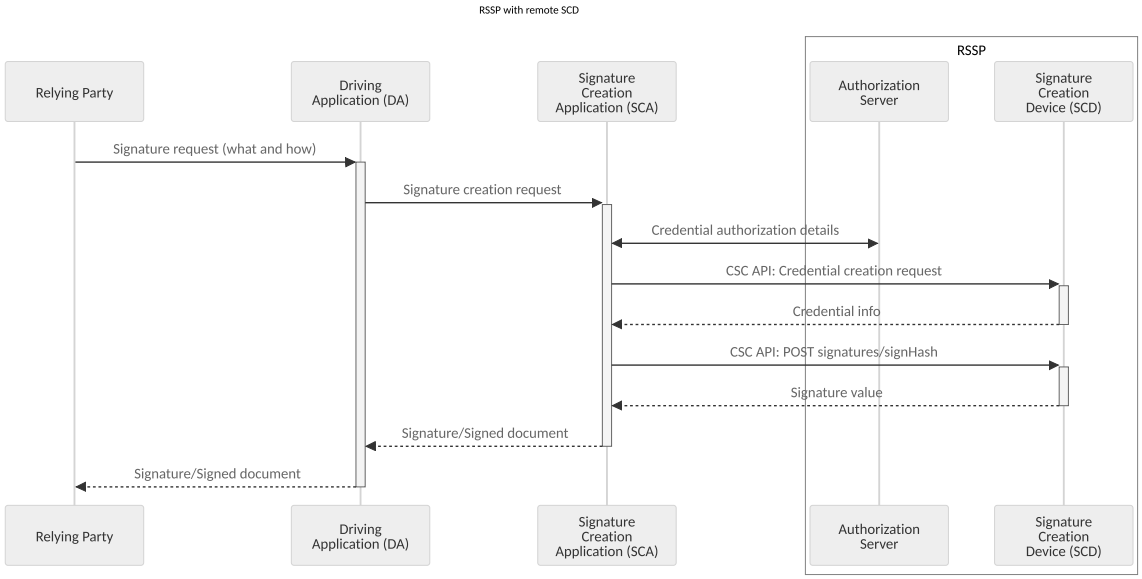
The data model described in the present document supports the interactions described above.

In some deployments, some of the interactions may be internal (e.g. to a single service provider, or a service providers with a pre-existing agreement). Where interoperability is not required, the interactions may not be as described in this section. Such interactions are out of scope of this document.

In some instances, the RSSP will implement the full CSC API. This enables a DA to use a remote SCA provided by an RSSP.



In other instances, the RSSP may only implement the `sighHash` functionality. In this case, a SCA can use a remote SCD provided by an RSSP.



The protocol for requesting a signature from a component in the environment of the signer (such as an identity wallet) is out of scope for the CSC, although one suitable protocol could be the CSC API [i.3] with its `signatures/signDoc` endpoint. For other protocols, this document provides data structures that can be used to convey the information required to request a signature.

7 Basic data types

This section specifies basic data components that are applied in remote signing-related processes specified in the CSC API [i.3]. How these data components are applied, is specified in separate API or data formatting specifications.

7.1 *adesParameters*

The *adesParameters* object expresses the AdES format of a signature.

The *adesParameters* object is composed of the following parameters.

Parameter	Presence	Value	Description
signature_format	OPTIONAL	*String*	The required signature format: • "C" SHALL be used to request the creation of a CAdES signature; • "X" SHALL be used to request the creation of a XAdES signature. • "P" SHALL be used to request the creation of a PAdES signature. • "J" SHALL be used to request the creation of a JAdES signature.
conformance_level	OPTIONAL	*String*	The required signature conformance level: • "AdES-B-B" SHALL be used to request the creation of a baseline 191x2 level B signature; • "AdES-B-T" SHALL be used to request the creation of a baseline 191x2 level T signature; • "AdES-B-LT" SHALL be used to request the creation of a baseline 191x2 level LT signature; • "AdES-B-LTA" SHALL be used to request the creation of a baseline 191x2 level LTA signature; • "AdES-B" SHALL be used to request the creation of a baseline etsits level B signature; • "AdES-T" SHALL be used to request the creation of a baseline etsits level T signature; • "AdES-LT" SHALL be used to request the creation of a baseline etsits level LT signature; • "AdES-LTA" SHALL be used to request the creation of a baseline etsits level LTA signature. If a timestamp is needed its request and inclusion is managed by the signing server according to signing server configuration and policies.

signed_envelope_property	OPTIONAL	*String*	The required property concerning the signed envelope whose possible values depend on the value of the *signature_format* parameter.
			According to the type of selected *signature_format* a client application may specify the following signature properties.
			• CAdES
			  • Detached
			  • Attached
			  • Parallel
			• PAdES
			  • Certification
			  • Revision
			• XAdES
			  • Enveloped
			  • Enveloping
			  • Detached
			• JAdES
			  • Detached
			  • Attached
			  • Parallel
			The default values are the following ones.
			• CAdES: Attached
			• PAdES: Certification
			• XAdES: Enveloped
			• JAdES: Attached

signed_props	OPTIONAL	*Array of attribute*	List of signed attributes other than SDR. The attributes that may be included depend on the signature format and the signature creation policy. Other attributes with non-colliding identifiers MAY be included.
referenceUri	OPTIONAL	*String*	If "signature_format" is "X" and "signed_envelope_property" is set to "DETACHED":

Parameter	Presence	Value	Description
			• If present, *referenceUri* MUST be used as the "URI" attribute of the corresponding signature reference. • If present, *referenceUri* MUST be a valid URI. • If not present, the "URI" attribute of the corresponding reference MUST be omitted. If "signature_format" is "J" and "signed_envelope_property" is set to "DETACHED": • If present, *referenceUri* MUST be used in the "sigD" header parameter. • If present, *referenceUri* MUST be a valid URI. • If not present, the "sigD" header parameter MUST be omitted. Otherwise: • This parameter MUST be ignored.

Note 4: The `referenceUri` will be part of the signature and should only be resolved during signature validation. The `referenceUri` may not always resolve to the document to be signed during the signing operation.

7.2 *attribute*

The *attribute* object is composed of the following parameters:

Parameter	Presence	Value	Description
attribute_name	REQUIRED	*String*	Attributes and/or properties listed in the tables in clauses 6.3 of ETSI EN 319 122-1 [8], ETSI EN 319 132-1 [9], ETSI EN 319 142-1 [10], and ETSI TS 119 182-1 [11], respectively, MAY be supported by the RSSP. The RSSP MAY define additional attributes. The RSSP SHOULD list supported and required attributes/properties in a signature creation policy.
attribute_value	OPTIONAL	*String*	When some element of this parameter is not defined the signing server SHALL calculate it, if needed.

7.3 *certificatePolicy*

Parameter	Presence	Value	Description
certificatePolicy	REQUIRED Conditional	*String*	String containing the OID identifying the applicable certificate policy for the new signing certificate.

7.4 *hash*

The *hash* object represents the hash digest of data and information about the hashing algorithm used.

The *hash* object is composed of the following parameters.

Parameter	Presence	Value	Description
value	REQUIRED	*String*	Hash digest value with base64 encoding.
algorithmOID	REQUIRED	*String*	OID of the hashing algorithm used to generate the hash value (e.g. "2.16.840.1.101.3.4.2.1" for SHA-256).

7.5 signatureQualifier

The *signatureQualifier* type is a *String* identifying the kind of signature.

The table below lists a set of pre-defined values for the *signatureQualifier* type.

Note 5: Service providers may define and use their own identifiers.

Identifier	Description
`"eu_eidas_qes"`	This identifier refers to a qualified electronic signature under eIDAS.
`"eu_eidas_aes"`	This identifier refers to an advanced electronic signature under eIDAS.
`"eu_eidas_aesqc"`	This identifier refers to an advanced electronic signature with qualified certificate under eIDAS.
`"eu_eidas_qeseal"`	This identifier refers to a qualified electronic seal under eIDAS.
`"eu_eidas_aeseal"`	This identifier refers to an advanced electronic seal under eIDAS.
`"eu_eidas_aesealqc"`	This identifier refers to an advanced electronic seal with qualified certificate under eIDAS.
`"za_ecta_aes"`	This identifier refers to an advanced electronic signature defined by the South African ECT Act [i.15].
`"za_ecta_oes"`	This identifier refers to an ordinary electronic signature defined by the South African ECT Act [i.15].

Note 6: Signature qualifiers follow the syntax X_Y_Z (e.g. eu_eidas_qes) where: X: The ISO 3166-1 [13] Alpha-2 code of the country where the signature legislation is defined (e.g. eu for Europe). Y: The shortform name of the legislation (e.g. eidas for Electronic Identification And Trust Services). Z: The shortform name of the signature type defined by the legislation (e.g. qes for Qualified Electronic Signatures).

7.6 signingAlgorithm

The *signingAlgorithm* object represents the cryptographic algorithm use for a signing operation.

The *signingAlgorithm* object is composed of the following parameters.

Parameter	Presence	Value	Description
signAlgo	REQUIRED	*String*	The OID of the algorithm to use for signing. For example: 1.2.840.113549.1.1.1 = RSA encryption, 1.2.840.10045.4.3.2 = ECDSA with SHA-256. Typically, it will be one of the values contained in the list of supported key algorithms in CSC API [i.3] methods `credentials/info` or `credentials/list`
signAlgoParams	REQUIRED Conditional	*String*	The base64-encoded DER-encoded ASN.1 signature parameters, if required by the signature algorithm. Some algorithms like RSASSA-PSS, as defined in RFC 8017 [12], may require additional parameters.

7.7 subjectData

The *subjectData* object is a JSON object with information about a natural or legal person. The *subjectData* SHOULD contain data intended for inclusion in the certificate.

The *subjectData* object SHOULD include “Standard Claims” from OIDC[i.16], where applicable. Profiles of this data type MAY add additional properties.

Example:

```
{
    "name": "John Doe",
    "family_name": "Doe",
    "given_name": "John",
    "address": {
        "country": "DK"
    }
}
```

8 Documents

This section specifies document-related data components that are applied in remote signing-related processes specified in the CSC API [i.3]. How these data components are applied, is specified in separate API or data formatting specifications.

8.1 *documentData*

The *documentData* object contains a full document to be signed. The document can either be a “Signer’s original document” or a “Signer’s formatted document”.

The *documentData* object is composed of the following parameters.

Parameter	Presence	Value	Description
label	OPTIONAL	*String*	String containing a human-readable description of the respective document.
document	REQUIRED	*String*	base64-encoded document content to be signed. In case hashes were provided for the credential authorization, then the RSSP SHALL verify that the hash of the document in this parameter corresponds to one of the hashes provided in the credential authorization.
documentType	OPTIONAL	*String*	Indication of the document type. Can be either `"sod"` or `"sfd"`. Default value is `"sod"`.
circumstantialData	OPTIONAL	*String*	base64-encoded data required by the RSSP to deterministically compute SDR from signer’s original document.

8.2 *documentInfo*

The *documentInfo* object is used to authorize documents and properties to be signed.

The *documentInfo* object is composed of the following parameters.

Parameter	Presence	Value	Description
label	OPTIONAL	*String*	String containing a human-readable description of the respective document.
hash	REQUIRED	*String*	String containing the actual base64-encoded octet-representation of the hash of the document.
hashType	OPTIONAL	*String*	Indication of the type of hash. SHALL be either `"sdr"`, `"dtbsr"`, or `"sodr"`. If absent, the hash type SHOULD be interpreted as `"dtbsr"`.
signed_props	OPTIONAL	*Array of attribute*	List of signed attributes other than SDR. The attributes that may be included depend on the signature format and the signature creation policy. Other attributes with non-colliding identifiers MAY be included.
circumstantialData	OPTIONAL	*String*	base64-encoded data required by the RSSP to deterministically compute SDR from signer’s original document.

Note 7: Future versions of this data model may add additional options for *hashType* and may specify a default value. Future versions of this data model will support RSSP-defined hash types.

8.3 *documentReference*

The *documentReference* object represents a location where the document to sign can be retreived.

The *documentReference* object is composed of the following parameters.

Parameter	Presence	Value	Description
label	OPTIONAL	*String*	String containing a human-readable description of the respective document.
access	OPTIONAL	*accessControlMethod*	Object defining the method for access control to the resource located using *href*, as defined below.
href	REQUIRED	*String*	The URL that locates signer's original document. This can for example be an `https://` URL for a remotely hosted document. For transmission of document data implementors SHOULD prefer to use *documentData* over *documentReference* with a `data` URI scheme.
checksum	OPTIONAL	*hash*	Checksum protecting the integrity of signer's original document. SHA-256 (2.16.840.1.101.3.4.2.1) MUST be supported.
circumstantialData	OPTIONAL	*String*	base64-encoded data required by the RSSP to deterministically compute SDR from signer's original document.

8.3.1 *accessControlMethod*

The *accessControlMethod* object is composed of at least the following parameter:

- *type*

specified according to the following table:

Parameter	Presence	Value	Description
type	REQUIRED	*String*	String value identifying the type of the method to control access to a remote resource. This document specifies several common access control methods. Other access control methods MUST be identified using a collision-resistant identifier.

8.3.1.1 Public access

Such an *accessControlMethod* object has *type* `"public"` and no additional parameters.

In this case, no further authorization is needed to access a remote resource. This does not preclude the option that the resource locator is secret, and access is thereby restricted to clients who know it.

8.3.1.2 One-time password access

Note 8: This access control method has not been validated for security at the moment of writing. Just like with other access control methods, it is up to the implementer to ensure proper design and implementation. The OTP option is, however, included to enable implementers to rely on client's ability to render this information in a standard way. As with any data model for access control methods, the data model requirements by itself do not provide any protection.

Such an *accessControlMethod* object has *type* `"OTP"` and the following parameter:

- *oneTimePassword*

specified according to the following table:

Parameter	Presence	Value	Description
oneTimePassword	REQUIRED Conditional	*String*	The one-time password value. MUST be included only if the type is `"OTP"`. A one-time password value MUST be provided only once in the context of accessing the remote resource.

The use case for the one-time password involves three roles in a cross-device scenario:

- an application server, for example hosting a signer's original document;
- an authenticated client, for example a driving application front-end on a laptop;
- an unauthenticated client that is not yet authenticated, but needs to obtain document data, for example a signature creation application in an identity wallet.

In this scenario, the application server prepares a message for the client, for example a *signatureRequest* or a *signatureCreationRequest*. The message contains a document reference, for example encoded as a *documentReference*, for which the application server implements access controls. The application server can implement one-time passwords as a protection against shoulder-surfing attacks:

1. The server provides a message reference containing a document reference to the unauthenticated client, for example as a QR code.
2. The unauthenticated client requests the message from the server, using the message reference.
3. The server issues a one-time password and includes it in the document reference which it sends as part of the message to the unauthenticated client.
4. The unauthenticated client displays the one-time password to the user.
5. The user enters the displayed one-time password in the authenticated client.
6. The server releases the signer's original document to the unauthenticated client.

This way, the user of the authenticated client controls document access to unauthenticated clients obtaining the message reference that includes the document reference. A shoulder-surfing attacker who obtains the message reference receives a different one-time password than the original user does on their own device, so the original user is unlikely to enter the attacker's one-time password on the authenticated device.

8.4 *documentRepresentations*

The *documentRepresentations* object represents the digests of one or more documents to sign (i.e. SDRs).

The *documentRepresentations* object is composed of the following parameters.

Parameter	Presence	Value	Description
label	OPTIONAL	*String*	String containing a human-readable description of the respective document.
hashes	REQUIRED	*Array of String*	One or more hash values representing one or more SDRs. This parameter SHALL contain the base64-encoded hash(es) of the documents to be signed.

9 Requests

This section specifies data components related to credentials and signatures that are applied in remote signing-related processes specified in the CSC API [i.3]. How these data components are applied, is specified in separate API or data formatting specifications.

9.1 *credentialCreationRequest*

The *credentialCreationRequest* object represents a request for creating a credential.

The *credentialCreationRequest* object is the union of the following:

- *certificatePolicy*
- The parameters in the table below

Parameter	Presence	Value	Description
subjectData	OPTIONAL	*subjectData*	Information to be included in the certificate associated with the new credential.

9.2 *credentialDeletionRequest*

The *credentialDeletionRequest* object represents a request to delete a credential and possibly revoke the corresponding certificate.

The *credentialDeletionRequest* object is composed of the following parameters.

Parameter	Presence	Value	Description
credentialID	REQUIRED	*String*	The *credentialID* to be deleted.
revoke	OPTIONAL	*Boolean*	If this parameter is present and is set to *true* the certificate corresponding to the deleted credential SHALL be revoked.
revocationReason	REQUIRED Conditional	*Integer*	This parameter MUST be present if *revoke* is present and has value *true*. This parameter MUST be ignored if *revoke* is not present or if *revoke* is false. When revoking the certificate, a CRL reason code MUST be specified as defined in RFC 5280 [16]. This MUST be a value between 0-10, with 7 being unused. See RFC 5280 [16], section 5.3.1.

9.3 *signatureCreationRequest*

The *signatureCreationRequest* object represents a request from a driving application to a signature creation application for signing a document. It contains information about what to sign (document and other signed attributes) and how to sign (formats and algorithms).

The *signatureCreationRequest* object is the union of the following:

- One of *documentData*, *documentReference*, *documentRepresentations*
- *adesParameters*
- *signingAlgorithm*

9.4 *signatureRequest*

The *signatureRequest* object represents a request from a relying party to a signer or driving application for signing a document.

The *signatureRequest* object is the union of the following:

- One of *documentData*, *documentReference*
- *adesParameters*
- The parameters in the table below

Parameter	Presence	Value	Description
signatureQualifier	REQUIRED	*signatureQualifier*	Identifier of the signature type to be created (e.g. “eu_eidas_qes” to denote a Qualified Electronic Signature according to eIDAS).
responseURI	OPTIONAL	*String*	URI describing where the response to this *signatureRequest* is expected to be sent to. The channel to the endpoint SHOULD be mutually authenticated.

10 Authorization

This section specifies authorization-related data components that are applied in remote signing-related processes specified in the CSC API [i.3]. How these data components are applied, is specified in separate API or data formatting specifications.

10.1 *signatureCreationApproval*

The *signatureCreationApproval* object represents information required by the signer to give informed consent to a signature creation.

The *signatureCreationApproval* object is composed of the following parameters.

Parameter	Presence	Value	Description
credentialID	REQUIRED Conditional	*String*	The identifier associated to the credential to authorize. At least one of the two values *credentialID* and *signatureQualifier* SHALL be present, both values MAY be present.
signatureQualifier	REQUIRED Conditional	*signatureQualifier*	Identifier of the signature type to be created (e.g. “eu_eidas_qes” to denote a Qualified Electronic Signature according to eIDAS). At least one of the two values *credentialID* and *signatureQualifier* SHALL be present, both values MAY be present.
numSignatures	REQUIRED	*Integer*	The number of signatures to authorize. Multi-signature transactions can be obtained by using a combination of array of hash values and by calling multiple times the RSSP function to sign a hash value.
documentDigests	REQUIRED	*Array of documentInfo*	An array of elements containing authorization details (such as hash values representing the documents and additional signed attributes) relevant for each document to be signed. The AS SHOULD use the *label* element in the user consent to designate the document. If *hashType* is absent, the RSSP SHOULD interpret the hash type as `"dtbsr"`. The *circumstantialData* parameter MAY be present if signature creation will be requested providing the signer’s original document. The RSSP SHOULD provide instructions for how the *circumstantialData* is created in e.g. the signature creation policy.
hashAlgorithmOID	REQUIRED	*String*	This hashing algorithm OID is used to generate the hash values in *documentDigests*.