CSC data model bindings

version 1.0.0

Contents

Foreword
Revision history
Acknowledgements
Introduction
Intellectual Property Rights
    Trademark notice
    Essential Patents
Legal notices
1 Scope
2 Interpretation of requirement levels
3 References
    3.1 Normative references
    3.2 Informative references
4 Terms, definitions and abbreviations
    4.1 Terms and definitions
    4.2 Abbreviations
    4.3 JSON data types
5 Conventions
    5.1 Text conventions
6 Qualified electronic signature or seal request
    6.1 Binding to protocols using JWT
        6.1.1 *signatureRequest* for JWT
        6.1.2 Example application
        6.1.3 Security considerations
    6.2 Binding to OID4VC
        6.2.1 *qesRequest*
        6.2.2 *qesResponse*
        6.2.3 Example application
    6.3 Transaction data processing
    6.4 Transaction data rendering
7 Qualified electronic signature or seal approval
    7.1 qesApprovalRequest
        7.1.1 Transaction data parameters
        7.1.2 Transaction authorization request example
    7.2 *qesApproval*
        7.2.1 Transaction data binding to credential formats
    7.3 Example application
8 X.509 certificate
    8.1 *x509MetadataQuery*
        8.1.1 *keyInfo*
    8.2 *x509PresentationResponse*
        8.2.1 DCQL query and presentation response examples for X.509 credential

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
1.0.0	14/10/2025	Public release, based on data model pre-releases

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 bindings of the CSC data model [9] to various protocols that relate to, but are not part of, the CSC API. It is not mandatory to use these bindings together with the CSC API. They are examples of how the data model can be used in various scenarios. The bindings in this specification may be taken as is, or adopted for a particular use.

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

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

This document describes bindings of the CSC data model[9] for use cases that are out of scope for CSC but may be used in relation with the CSC API.

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

[9] Cloud Signature Consortium, “Data model for remote signature applications”.

[10] IETF RFC 7519: “JSON Web Token (JWT)”.

[11] 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”.

3.2 Informative references

[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 - version 2.

[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] W3C Subresource Integrity, Recommendation 23 June 2016.

[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.

4 Terms, definitions and abbreviations

4.1 Terms and definitions

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

authorization server: hereon abreviated as “AS”, server enabling users to authorize privileged operations.

Note 2: The AS 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 AS.

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 [11].

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.

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.

AS: authorization server

DA: driving application

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

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

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

QES Qualified Electronic Signature or Seal

RP: relying party

RSSP: remote signing service provider

SCA: signature creation application

SCD: signature creation device

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

4.3 JSON data types

Unless otherwise specified, the following data types from JSON [6] are used.

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

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 *qesRequest* object is an *Object* that conforms to the requirements specified in the *qesRequest* section.

6 Qualified electronic signature or seal request

A QES is an advanced electronic signature or seal created using a qualified certificate as specified in RFC 3739 [i.7], bound to a qualified signature or seal creation device, as defined in the associated legal framework.

The *signatureRequest* data object from the data model[9] is used when a relying party requests that a signer signs a document. The relying party must transfer the *signatureRequest* to the driving application. The driving application may return the signed document as part of a direct interaction between the relying party and the driving application. When there is no direct connection between the relying party and the driving application, the driving application may upload the signed document to the *responseURI* specified in the *signatureRequest*.

The transmission of the *signatureRequest* from the relying party to the driving application must be protected in integrity and confidentiality and the driving application must be able to authenticate the relying party.

6.1 Binding to protocols using JWT

One way to transmit the *signatureRequest* is by transmitting a JWT[10]. This section does not require any specific transport protocol for the JWT but gives a few examples. In some examples, the relying party posts the JWT to an endpoint provided by the driving application. In other examples, the relying party transmits a URL to a location where the JWT can be downloaded by the driving application. The relying party may, for instance, display a QR-code encoding the URL of the JWT to the signer. The signer can then transmit the URL to the driving application.

6.1.1 *signatureRequest* for JWT

6.1.1.1 Transaction data parameters

Parameter	Presence	Value	Description
iss	OPTIONAL	*String*	Issuer as defined in [10]. *iss* MUST identify the relying party.
signatureRequests	REQUIRED	Array of *signatureRequest*	An array of elements containing details about the requested signing operation. All elements SHALL be of type *signatureRequest* with *documentReference* as defined in [9]. The AS SHOULD use the *label* element in the user consent to designate the document. Signature applications that support this transaction data type MUST support `data:` URLs [3] with base64 encoding in the *href* parameter. They MAY restrict the supported media types and the acceptable body length.

Note 3: The *iss* claim is included to allow the driving application to validate that the relying party is authorized to request signatures. This specification does not define any method for validating the authorization of the relying party. Implementors should create a profile of this binding that describes methods for validating relying party authorization. Implementors may add additional claims (e.g. *x5c*) to support the authorization of the relying party.

6.1.2 Example application

In this non-normative example, the driving application is running on the device of the signer next to, or as part of, an identity wallet.

The relying party prepares the following *signatureRequest* JWT header:

```
{
  "alg": "ES384",
  "typ": "JWT"
}
```

and data:

```
{
  "iss": "https://example.com",
  "signatureRequests": [
    {
      "label": "Example Terms of Service",
      "access": { "type": "public" },
      "href": "https://example.com/terms-and-conditions.pdf",
      "checksum": "sha256-HZQzZmMAIWekfGH0/ZKW1nsdt0xg3H6bZYztgsMTLw0=",
      "signature_format": "P",
      "conformance_level": "AdES-B-B",
      "signed_envelope_property": "Certification",
      "signatureQualifier": "eu_eidas_qes",
      "responseURI": "https://example.com/signatureResponse/123/"
    }
  ]
}
```

After signing, the resulting JWT may look like this:

```
eyJhbGciOiJFUzM4NCIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJodHRwczovL2V4YW1wbGUuY29tIiwic
2lnbmF0dXJlUmVxdWVzdHMiOlt7ImxhYmVsIjoiRXhhbXBsZSBUZXJtcyBvZiBTZXJ2aWNlIiwiYWN
jZXNzIjp7InR5cGUiOiJwdWJsaWMifSwiaHJlZiI6Imh0dHBzOi8vZXhhbXBsZS5jb20vdGVybXMtY
W5kLWNvbmRpdGlvbnMucGRmIiwiY2hlY2tzdW0iOiJzaGEyNTYtSFpRelptTUFJV2VrZkdIMC9aS1c
xbnNkdDB4ZzNINmJaWXp0Z3NNVEx3MD0iLCJzaWduYXR1cmVfZm9ybWF0IjoiUCIsImNvbmZvcm1hb
mNlX2xldmVsIjoiQWRFUy1CLUIiLCJzaWduZWRfZW52ZWxvcGVfcHJvcGVydHkiOiJDZXJ0aWZpY2F
0aW9uIiwic2lnbmF0dXJlUXVhbGlmaWVyIjoiZXVfZWlkYXNfcWVzIiwicmVzcG9uc2VVUkkiOiJod
HRwczovL2V4YW1wbGUuY29tL3NpZ25hdHVyZVJlc3BvbnNlLzEyMy8ifV19.u2rHWH4PdaCi561Uvt
B_GJmI-8kVFA7ru480HhOHRPtTUr2xvDHUu4pYogL8-MMTO5MaEBnfgYTlcGA01escOJ2CihQXXYt5
3gI1ooUMmBSqHhYb7-lZJh3fgo_fSuPb
```

The relying party makes the JWT available at a URL, e.g. `https://example.com/signatureRequst/123/` and prepares a QR-code that encodes the URL. The relying party shows the QR-code to the signer who scans the QR-code with the driving application. The driving application downloads the JWT and validates that the issuer is authorized. If the issuer is authorized, the driving application invokes the signature creation application with parameters from the JWT.

6.1.3 Security considerations

- The JWT SHALL be signed.
- The signature algorithm used to sign the JWT SHOULD be an approved algorithm.
- The driving application SHALL validate the signature of the JWT.
- The driving application SHOULD validate that the issuer is authorized to request signatures.

6.2 Binding to OID4VC

6.2.1 *qesRequest*

The transaction data must be associated with a qualified certificate for creating a QES. This credential can be identified by a field *credential_ids* specified below. One way to identify the credential is an *x509MetadataQuery*.

The transaction data type identifier for requesting QES is: `"https://cloudsignatureconsortium.org/2025/qes"`.

6.2.1.1 Transaction data parameters

Parameter	Presence	Value	Description
type	REQUIRED	*String*	A data type identifier. MUST be `"https://cloudsignatureconsortium.org/2025/qes"`.
credential_ids	REQUIRED Conditional	*Array of String*	References to credentials to approve a transaction with. MUST be included if and only if the protocol for handling transaction data requires it.
signatureRequests	REQUIRED	Array of *signatureRequest*	An array of elements containing details about the requested signing operation. All elements SHALL be of type *signatureRequest* with *documentReference* as defined in [9]. The AS SHOULD use the *label* element in the user consent to designate the document. Signature applications that support this transaction data type MUST support `data:` URLs [3] with base64 encoding in the *href* parameter. They MAY restrict the supported media types and the acceptable body length.

Note 4: The *credential_ids* data component is included for compatibility with OpenID4VP [i.5]. Typically, it refers to dynamic local identifiers that are defined in a query. In contrast, *credentialID* is a static credential identifier defined at a remote signing service provider.

Note 5: Since *qesRequest* is used to request a QES, it is important to specify the applicable trust framework. Therefore, in contrast to the CSC API [i.3] which requires the *signatureQualifier* to be present conditionally, this binding always requires the *signatureQualifier* to be present.

Note 6: This transaction data type is likely to change in future versions of this document.

6.2.1.2 Transaction authorization request example

A non-normative example of a base64url-decoded OpenID4VP [i.5] *transaction_data* string:

```
{
    "type": "https://cloudsignatureconsortium.org/2025/qes",
    "credential_ids": ["xyz123"],
    "signatureQualifier": "eu_eidas_qes",
    "signatureRequests": [
        {
            "label": "Example Contract",
            "access": { "type": "OTP", "oneTimePassword": "51623" },
            "href": "https://protected.rp.example/contract-01.pdf?
        token=HS9naJKWwp901hBcK348IUHiuH8374",
            "checksum": "sha256-sTOgwOm+474gFj0q0x1iSNspKqbcse4IeiqlDg/HWuI=",
            "signature_format": "P",
            "conformance_level": "AdES-B-B",
            "signed_envelope_property": "Certification",
            "signAlgo": "1.2.840.113549.1.1.1"
        },
        {
            "label": "Example Terms of Service",
            "access": { "type": "public" },
            "href": "https://public.rp-cdn.example/terms-and-conditions.pdf",
            "checksum": "sha256-HZQzZmMAIWekfGH0/ZKW1nsdt0xg3H6bZYztgsMTLw0=",
            "signature_format": "P",
            "conformance_level": "AdES-B-B",
            "signed_envelope_property": "Certification",
            "signAlgo": "1.2.840.113549.1.1.1"
        },
        {
            "label": "Example Configuration",
            "href": "data:application/json;base64,eyJleGFtcGxlS2V5IjoiZXhhbXBsZVZhbHVlIn0K",
            "signature_format": "J",
            "conformance_level": "AdES-B-B",
            "signed_envelope_property": "Attached",
            "signAlgo": "1.2.840.113549.1.1.1"
        }
    ]
}
```

In this example, the credential identified by `"xyz123"` in the associated DCQL query will be used to sign the provided documents, after obtaining them at the Signature Application.

Note 7: The credential ID in OpenID4VP [i.5] is an opaque string defined by the creator of the transaction data and unrelated to the credential ID in the CSC API [i.3].

While *transaction_data* is defined by the OpenID4VP [i.5] specification and therefore base64url-encoding is applied on the entire *Object*, for data structures within the object, which are defined by this document, the encoding defined in this document SHALL be used (e.g. the included checksum data element follows the definition of Common documentInfo parameters, it uses the Subresource Integrity [i.8] data format with base64-encoded checksum and without padding).

For obtaining the first document, the Signature Application displays a one-time password to the user. When the user enters the same one-time password in the Driving Application, the Driving Application unlocks the resource at the URI for download. This mechanism addresses the risk of over-the-shoulder attacks in cross-device scenarios.

6.2.2 *qesResponse*

In a QES transaction using an X.509 certificate, the private key corresponding to the public key identified in the X.509 certificate is used to create the QES. The *qesResponse* object is composed of the following parameters:

Note 8: The below specification defines common response data that could be used in OpenID4VP responses (e.g., VP token response or DCQL response). There is no requirement of specific encoding in OpenID4VP. CSC is by intention, for consistency and compatibility with other data elements specified by CSC, defining the below data elements with base64 encoding, and not base64url-encoded as in other elements defined in OpenID4VP.

Parameter	Presence	Value	Description
documentWithSignature	REQUIRED Conditional	*Array of String*	One or more base64 encoded signatures enveloped within the documents. MUST be included only if the OpenID4VP [i.5] request's *transaction_data* did not specify a *responseURI* AND if the format of the signed data object implies the signature is embedded in the document (use e.g., for PAdES signatures, or CAdES/XAdES enveloped signatures).
signatureObject	REQUIRED Conditional	*Array of String*	One or more base64 encoded signatures that are detached from or contain the signed data. MUST be included only if the request's *transaction_data* did not specify a *responseURI* AND if the format of the signed data object implies the signature is not embedded in the document (use e.g., for CAdES/XAdES detached or enveloping signatures).

Note 9: In case of *responseURI* being specified in the transaction data, neither *documentWithSignature* nor *signatureObject* will be contained in the *qesResponse*.

6.2.3 Example application

In the Wallet-centric model, a Relying Party requests a QES transaction from a Wallet (Driving Application) using an X.509 credential:

```
{
    "dcql_query": {
        "credentials": [
            {
                "id": "abc234",
                "format": "https://cloudsignatureconsortium.org/2025/x509",
                "meta": {}
            }
        ]
    },
    "transaction_data": [
        "ewogICAgInR5cGUiOiAiaHR0cHM6Ly9jbG91ZHNpZ25hdHVyZWNvbnNvcnR..."
    ]
}
```

Note 10: As *transaction_data* relates to OpenID4VP [i.5] specification, it is base64url-encoded (i.e. without padding).

Note 11: The credential ID in DCQL is an opaque string defined by the creator of the query and unrelated to the credential ID in the CSC API [i.3].

Here, the *transaction_data* contains a base64url-encoded *Object* in UTF-8 encoding:

```
{
    "type": "https://cloudsignatureconsortium.org/2025/qes",
    "credential_ids": ["abc234"],
    "signatureQualifier": "eu_eidas_qes",
    "signatureRequests": [
        {
            "label": "Example Terms of Service",
            "access": { "type": "public" },
            "href": "https://public.rp-cdn.example/terms-and-conditions.pdf",
            "checksum": "sha256-HZQzZmMAIWekfGH0/ZKW1nsdt0xg3H6bZYztgsMTLw0=",
            "signature_format": "P",
            "conformance_level": "AdES-B-B",
            "signed_envelope_property": "Certification",
            "signAlgo": "1.2.840.113549.1.1.1"
        }
    ]
}
```

Note 12: While *transaction_data* is defined by OpenID4VP specification and therefore base64url-encoding is applied on the entire JSON, the data structure within the *Object* which is defined by this document use the encoding defined in this document (e.g. the included checksum data element SHALL follow the Subresource Integrity [i.8] data format with base64-encoded checksum and without padding).

After creating the QES, the Wallet responds with the QES:

```
{
    "abc234": {
        "qes": {
            "documentWithSignature": [
                "<base64-encoded document with signature>"
            ]
        }
    }
}
```

6.3 Transaction data processing

If the *qesRequest* is provided in the context of a credential that cannot be used to create a QES with the specified *signatureQualifier*, the Signature Application MUST abort the transaction.

If a *signatureRequest* object contains both a *href* and a *checksum* value, the Signature Application MUST verify resource integrity against the *checksum* and abort the transaction upon failure.

The Signature Application SHALL log the provided transaction data and the user's decision to approve or reject the transaction.

If the *responseURI* is provided and the QES is created, the Signature Application SHALL attempt to send the response to the identified endpoint.

When sending the response to a *responseURI* with the `https:` URI scheme, the Signature Application SHALL use HTTP/1.1 POST according to RFC 9112 [7] as follows.

HTTP POST request messages SHALL have the following structure:

```
POST <responseURI path> HTTP/1.1
Host <responseURI host>
Content-Length: <qesResponse length>
Content-Type: application/json

<qesResponse>
```

HTTP POST response messages, upon successful receipt, SHALL have the following structure:

```
HTTP/1.1 200 OK
```

Other request or response headers MAY be included.

The response endpoint SHOULD process only the first successfully received request.

6.4 Transaction data rendering

The Signature Application SHALL render the provided transaction data to the user upon authorization or upon reviewing logs. The rendering can be visual, audible, or through other means in such a way that the user can be expected to understand it. Some transaction data may only be fully rendered upon the user's request, for example, when requesting detailed inspection of the transaction data. The following rules apply to all fields included in QES transaction data.

Parameter	Rendering requirements
type	The Signature Application SHALL always render a clear indication that the transaction creates a QES, in such a way that the user can distinguish this from any other transactions performed using the application.
signatureQualifier	The Signature Application SHALL always render a clear indication of whether the QES is an electronic signature or an electronic seal, The Signature Application SHALL render a clear indication of the trust framework under which it is qualified.
signatureRequests.label	The Signature Application SHALL always render the full label if specified. If unspecified, the Signature Application SHALL clearly indicate that the document has no label.
signatureRequests.access	If the type is `"OTP"`, the Signature Application SHALL render the *oneTimePassword* value when the user may need to enter it into the Driving Application to release access to the document using the provided *signatureRequestInfos/documentInfo/href* value.
signatureRequests.href	The Signature Application SHALL enable the user to load the document in an application that the user has configured for the media type.
signatureRequests.checksum	If the document is specified with *href*, the Signature Application SHALL indicate whether resource integrity has been automatically verified or not.
signatureRequests.signed_props	If specified, the Signature Application SHALL render the full set of attribute names and values to the user. If the Signature Application recognises an attribute name and has a custom way of rendering it that helps the user understand the consequences of the transaction, it MAY render these attributes in a custom way. If the Signature Application does not recognise the attribute name, it SHALL render the attribute value as-is.
signatureRequests.signature_format	If specified, the Signature Application SHALL enable the user to distinguish which signature format is required in the transaction.
signatureRequests.conformance_level	If specified, the Signature Application SHALL enable the user to distinguish which conformance level is required in the transaction.
responseURI	The Signature Application SHALL enable the user to learn unambiguously which URI is used.

Note 13: If the transaction data contains *href* but the user chooses not to load the document, the Signature Application may need to retrieve the document anyway later to be able to compute the data to be signed. However, this retrieval can be done using streaming, while typically a user render requires storing the whole document.

7 Qualified electronic signature or seal approval

This transaction data type lets the credential express the user's approval for QES creation during an authorization flow run by a server trusted by the provider that remotely manages the qualified signature or seal creation device.

7.1 qesApprovalRequest

The transaction data type identifier for approving qualified electronic signatures or seals (QES) is: `"https://cloudsignatureconsortium.org/2025/qes-approval"`. The transaction data must be associated with a credential for approval of QES creation. This credential can be identified by a field *credential_ids* specified below. One way to identify the credential is an *x509MetadataQuery*.

7.1.1 Transaction data parameters

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

- *signatureCreationApproval* (see "Data model for remote signature applications" [9])
- The parameters in the table below

Parameter	Presence	Value	Description
type	REQUIRED	*String*	A data type identifier. MUST be `"https://cloudsignatureconsortium.org/2025/qes-approval"`.
credential_ids	REQUIRED Conditional	*Array of String*	References to credentials to approve a transaction with. MUST be included if and only if the protocol for handling transaction data requires it.
locations	OPTIONAL	*Array of String*	The locations of remote signing service providers as defined in RFC 9396 [i.4].

Note 14: The *credential_ids* data component is included for compatibility with OpenID4VP [i.5]. Typically, it refers to dynamic local identifiers that are defined in a query. In contrast, *credentialID* is a static credential identifier defined at a remote signing service provider.

Note 15: While *circumstantialData* may not be part of transaction data, the transaction may be approved as part of authorizing access to the `signatures/signDoc` CSC API [i.3] endpoint. The authorization details for that request may need to include the *circumstantialData*. In general, while there is overlap between transaction data and authorization details, these do not always need to contain the same values.

Note 16: This transaction data type is likely to change in future versions of this document.

7.1.2 Transaction authorization request example

A non-normative example of a base64url-decoded OpenID4VP [i.5] *transaction_data* string:

```
{
    "type": "https://cloudsignatureconsortium.org/2025/qes-approval",
    "credential_ids": ["xyz123"],
    "numSignatures": 2,
    "signatureQualifier": "eu_eidas_qes",
    "documentInfos": [
      {
        "label": "Example Contract",
        "hash": "sTOgwOm+474gFj0q0x1iSNspKqbcse4IeiqlDg/HWuI=",
        "hashType": "sodr",
        "access": { "type": "OTP", "oneTimePassword": "51623" },
        "href": "https://protected.rp.example/contract-01.pdf?token=HS9naJKWwp901hBcK348IUHiuH8374",
        "checksum": "sha256-sTOgwOm+474gFj0q0x1iSNspKqbcse4IeiqlDg/HWuI="
      },
      {
        "label": "Example Terms of Service",
        "hash": "HZQzZmMAIWekfGH0/ZKW1nsdt0xg3H6bZYztgsMTLw0=",
        "hashType": "sodr",
        "access": { "type": "public" },
        "href": "https://public.rp-cdn.example/terms-and-conditions.pdf",
        "checksum": "sha256-HZQzZmMAIWekfGH0/ZKW1nsdt0xg3H6bZYztgsMTLw0="
      },
      {
        "label": "Example Invoice",
        "hash": "nL7zQmAKfQ2jADrOxkEZh2UqV4Lx4WsmelSivP6LjoQ=",
        "hashType": "sodr",
        "access": { "type": "OTP", "oneTimePassword": "83920" },
        "href": "https://protected.rp.example/invoice-2025-07.pdf?token=jk47ns88sna9a",
        "checksum": "sha256-nL7zQmAKfQ2jADrOxkEZh2UqV4Lx4WsmelSivP6LjoQ="
      }
    ],
    "hashAlgorithmOID": "2.16.840.1.101.3.4.2.1"
}
```


In this example, the credential identified by `"xyz123"` in the associated DCQL query will be used to approve the signature of the provided documents.

Note 17: The credential ID in OpenID4VP [i.5] is an opaque string defined by the creator of the transaction data and unrelated to the credential ID in the CSC API [i.3].

While *transaction_data* is defined by the OpenID4VP [i.5] specification and therefore base64url-encoding is applied on the entire *Object*, for data structures within the object, which are defined by this document, the encoding defined in this document SHALL be used (e.g. the included checksum data element follows the definition of Common documentInfo parameters, it uses the Subresource Integrity [i.8] data format with base64-encoded checksum and without padding).

7.2 *qesApproval*

The result of approval is a single data component that binds the credential presentation to the transaction data. The *qesApproval* is a *String* containing a base64 encoded hash digest of an encoded *qesApprovalRequest*. The hash algorithm SHALL be identified using the *hashAlgorithmOID* parameter of the *qesApprovalRequest*. The encoding used for the hash input SHALL preserve the UTF-8 encoded *qesApprovalRequest* as originally provided. The encoding of the *qesApprovalRequest* in the *qesApproval* hash input SHALL depend on the format of the credential using which the *qesApproval* is authenticated.

7.2.1 Transaction data binding to credential formats

Note 18: This section on binding QES transactions to other credentials is still being validated with experts and can be changed in future versions of this document.

This section specifies the binding to two formats:

- ISO/IEC 18013-5-compliant encoding
- SD-JWT VC-based encoding

7.2.1.1 ISO/IEC 18013-5-compliant encoding

This section applies to QES transactions using credentials in ISO/IEC 18013-5 mdoc [i.11] format.

The *qesApproval* SHALL be protected in a *DeviceSigned* data structure.

Note 19: Since the *qesApproval* value is dynamic, it could not be signed by the issuer.

The *qesApproval* SHALL be identified using the *NameSpace* value `"org.cloudsignatureconsortium.dm.1"`.

The *qesApproval* SHALL be identified using the *DataElementIdentifier* value `"qesApproval"`.

The *qesApproval* SHALL be represented as a *DataElementValue* encoded as a CDDL [8] *bstr* value in CBOR, containing the SHA-256 hash digest without Base64 encoding this digest.

The hash input to create *qesApproval* SHALL be the UTF-8 encoded *qesApprovalRequest*.

Note 20: If the *qesApprovalRequest* is provided with base64url encoding, such as in OpenID4VP [i.5], this means that this value needs to be base64url decoded before hashing.

Note 21: An ISO/IEC 18013-5 mdoc needs to have explicit support to enable this application. In particular:

- The mdoc type, typically specified in a rulebook, needs to include the *qesApproval* attribute.
- The mdoc needs to be issued with a *KeyAuthorizations* object that includes:
  - `"org.cloudsignatureconsortium.dm.1"` in *AuthorizedNameSpaces*;
  - the mapping from `"org.cloudsignatureconsortium.dm.1"` to *qesApproval* in *AuthorizedDataElements*.

7.2.1.2 SD-JWT VC-based encoding

This section applies to QES transactions using credentials in SD-JWT VC [i.12] format.

The *qesApproval* SHALL be protected in a Key Binding JWT included in the presentation of the SD-JWT VC.

The *qesApproval* SHALL be identified using the top-level claim key `"org.cloudsignatureconsortium.dm.1.qesApproval"`.

The *qesApproval* SHALL be represented as a *String*.

The hash input to create *qesApproval* SHALL be the base64url encoded UTF-8 encoded *qesApprovalRequest*.

Note 22: The SD-JWT VC standard does not specify the inclusion of additional top-level claims in the Key Binding JWT. However, it is expected that wallets only return the `"org.cloudsignatureconsortium.dm.1.qesApproval"` claim for credentials that are issued with the intention to support the authorization of QES creation. This intention can, for example, be expressed in a rulebook that specifies the SD-JWT VC type.

7.3 Example application

In the Provider-centric model, the AS is related to a trust service provider managing a remote QSCD. When issuing a qualified certificate to a user, the trust service provider also issues a "service user attestation" as an attestation of attributes in mdoc or SD-JWT VC format with *qesApproval* support. In this example, the attestation has type `"com.example.service.1.user"` specified in a rulebook with the following attributes:

Namespace	Attribute identifier	Specification
`"com.example.service.1"`	*userName*	A CDDL [8] *tstr* identifying the user at the AS.
`"com.example.service.1"`	*credentialID*	A CDDL [8] *tstr* identifying the credential at the AS.
`"org.cloudsignatureconsortium.dm.1"`	*qesApproval*	Specified in *qesApproval*.

When authorizing the creation of a QES, the AS requests identification using this attestation, bound to QES transaction data:

```
{
    "dcql_query": {
        "credentials": [
            {
                "id": "qscd_service_attestation",
                "format": "mso_mdoc",
                "meta": {
                    "doctype_value": "com.example.service.1.attestation"
                },
                "claims": [
                    { "path": ["com.example.service.1", "userName"], "values": ["willeke"] },
                    { "path": ["com.example.service.1", "credentialID"], "values": ["GX0112348"] },
                    { "path": ["org.cloudsignatureconsortium.dm.1", "qesApproval"] }
                ]
            }
        ]
    },
    "transaction_data": [
        "ewogICAgInR5cGUiOiAiaHR0cHM6Ly9jbG91ZHNpZ25hdHVyZWNvbnNvcnR..."
    ]
}
```

Note that the AS specifically asks for an attestation of the attribute *credentialID* equal to `GX0112348`, which corresponds with an example credential managed in the remote QSCD for user `willeke`.

Here, the *transaction_data* contains a base64url-encoded *Object* in UTF-8 encoding:

```
{
    "type": "https://cloudsignatureconsortium.org/2025/qes-approval",
    "credential_ids": ["xyz123"],
    "numSignatures": 2,
    "signatureQualifier": "eu_eidas_qes",
    "documentInfos": [
      {
        "label": "Example Terms of Service",
        "hash": "sTOgwOm+474gFj0q0x1iSNspKqbcse4IeiqlDg/HWuI=",
        "hashType": "dtbsr",
        "access": { "type": "OTP", "oneTimePassword": "51623" },
        "href": "https://protected.rp.example/contract-01.pdf?token=HS9naJKWwp901hBcK348IUHiuH8374",
        "checksum": "sha256-sTOgwOm+474gFj0q0x1iSNspKqbcse4IeiqlDg/HWuI="
      }
    ],
    "hashAlgorithmOID": "2.16.840.1.101.3.4.2.1"
}
```

After user authorization of the transaction towards their identity wallet, the identity wallet returns to the AS the verifiable presentation of the service credential:

```
{
    "abc123": ["<base64url-encoded DeviceResponse>"]
}
```

This *DeviceResponse* object in this example is a CBOR object matching the following CDDL [8]:

```
{
    "version": "1.0",
    "documents": [
        {
            "docType": "com.example.service.1.attestation",
            "issuerSigned": {
                "nameSpaces": {
                    "com.example.service.1": {
                        "userName": "willeke",
                        "credentialID": "GX0112348"
                    }
                },
                "issuerAuth": IssuerAuth              ; from ISO/IEC 18013-5
            },
            "deviceSigned": {
                "nameSpaces": #6.24(bstr .cbor {
                    "org.cloudsignatureconsortium.dm.1": {
                        "qesApproval": bstr .size 32 ; SHA-256 hash
                    }
                }),
                "deviceAuth": DeviceAuth              ; from ISO/IEC 18013-5
            }
        }
    ],
    "status": 0
}
```

After validating the presented attestation using the *DeviceResponse*, the AS is assured of the identity of the user and the consent to authorize QES creation with the specified *credentialID* and data to be signed. Subsequently, the AS may proceed with any other needed steps to obtain an authorization grant from the resource owner.

Note 23: In this example, the *DeviceResponse* object identifies the user towards the AS, but not towards the remote QSCD. Also, its *DeviceAuth* property can be verified only by the AS acting as an mdoc reader, and not by other roles. Therefore, the *DeviceResponse* object is not signature activation data. The AS could rely on the validation of the electronic attestation of attributes to create an identity assertion for use in the signature activation data. Depending on the signature activation protocol, additional steps may include cryptographically binding the *credentialID* and the data to be signed to the signature activation data. The specification of such steps is out of scope for this document.

8 X.509 certificate

This section specifies the X.509 credential formats for use in remote signature specifications. Credential formats are applied in the OpenID for Verifiable Credentials (OID4VC) series of standards [i.6]. These formats are specified in addition to OpenID4VP [i.5] Appendix B.

The credential format identifier for X.509 certificates is: `"https://cloudsignatureconsortium.org/2025/x509"`.

A request for proof of possession of an X.509 certificate MAY include *transaction_data* for creating electronic signatures or seals using the queried certificate. A successful credential response includes the created electronic signatures or seals.

Note 24: The transaction data type specifies which parameters are included and how these are processed. These parameters represent signer's original documents, signed properties, and processing rules. One applicable transaction data type is specified under Qualified electronic signature or seal request. In OID4VC terms, the result of creating an electronic signature or seal is a presentation of the X.509 certificate, bound to the transaction data.

This section defines X.509 certificate-specific (DCQL) credential query and response types:

- *x509MetadataQuery*: Parameters in the *meta* parameter in the credential query.
- *x509PresentationResponse*: Attributes in the presentation response.

These types are specified according to the following sections. There are no parameters defined to query specific claims in an X.509 certificate.

8.1 *x509MetadataQuery*

Parameter	Presence	Value	Description
certificateFingerprints	OPTIONAL	Array of [*hash*]	Definition of acceptable certificate fingerprints. Only hashing algorithms as strong or stronger than SHA-256 SHALL be used. The hash algorithm SHOULD follow the recommendations of ETSI TS 119 312 [i.13].
certificatePolicies	OPTIONAL	*Array of String*	List of OIDs identifying applicable certificate policies. Default value: `[]`.
keys	OPTIONAL	*Array of keyInfo*	Allowed signing key configurations. If omitted, all signing key configurations are allowed.

If the *certificatePolicies* are specified, the response SHALL only include a certificate that includes at least one of the listed *certificatePolicies*.

If *certificateFingerprints* are specified, the response SHALL only include a certificate that matches one of the listed fingerprints.

8.1.1 *keyInfo*

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

- *algo*
- *len*
- *curve*

specified according to the following table:

Parameter	Presence	Value	Description
algo	REQUIRED	*String*	The list of OIDs of the supported key algorithms. For example: 1.2.840.113549.1.1.1 = RSA encryption, 1.2.840.10045.4.3.2 = ECDSA with SHA-256.
len	REQUIRED Conditional	*Integer*	The length of the cryptographic key in bits. The value SHALL NOT be used if *keyAlgo* is based on elliptic curve cryptography.
curve	REQUIRED Conditional	*String*	The OID of the elliptic curve. The value SHALL only be used if *keyAlgo* is based on elliptic curve cryptography.

8.2 *x509PresentationResponse*

The following are X.509 certificate-specific credential response parameters.

Parameter	Presence	Value	Description
qes	REQUIRED Conditional	*qesResponse*	The result of processing transaction data with a *qesRequest* using the resulting credential. It MUST be included if and only if the *qesResponse* is produced and if the *qesRequest* does not contain a *responseURI*. If it does contain a *responseURI*, the response is instead posted to this endpoint before returing the (empty) credential presentation. The result SHALL contain qualified electronic signatures or seals with the resulting credential as the signer certificate.

Note 25: In OpenID4VP terms [i.5], approval of a *qesRequest* transaction using an X.509 certificate leads to a Presentation. The Presentation is not a Verifiable Presentation, since no Cryptographic Holder Binding is defined that protects against replaying the QES. That is, a QES can be replayed if the same DCQL query with the same QES *transaction_data* is provided twice. In terms of electronic signatures and seals however, the QES is cryptographically bound to the subject of the X.509 certificate. Applications can implement various ways to protect against replay if needed, such as including an application-specific nonce (unrelated to the OpenID4VP nonce) in the signer's original document.

Note 26: When used with OpenID4VP [i.5], this binding specifies two options for receiving the *qesResponse*:

- Inline option as a *qes* value in *x509PresentationResponse*.
- Out-of-band option using *responseURI*. The *x509PresentationResponse* is returned without a *qes* value.

Wallets can support either or both options, and it is up to the relying party to decide which option to request. Each option has operational and information security considerations. With the inline option, the relying party could leverage existing OpenID4VP and possibly Digital Credentials [i.14] infrastructure, including its information security controls. However, in some cases such infrastructure is unable to handle large documents with QES. With the out-of-band option, the relying party is more flexible in how it processes the response. However, both the wallet and the relying party need to design and implement appropriate security controls to maintain confidentiality of the *qesResponse* object. For example, the wallet could limit its trust anchors to only accept a server connection with a qualified website authentication certificate from within the same trust framework as the qualified certificate for QES.

8.2.1 DCQL query and presentation response examples for X.509 credential

A non-normative example DCQL query using the X.509 credential format is:

```
{
    "credentials": [
        {
            "id": "xyz123",
            "format": "https://cloudsignatureconsortium.org/2025/x509",
            "meta": {
                "certificatePolicies": [
                    "0.4.0.2042.1",
                    "0.4.0.194112.1"
                ]
            }
        }
    ]
}
```

Note 27: The credential ID in DCQL is an opaque string defined by the creator of the query and unrelated to the credential ID in the CSC API [i.3].

A non-normative example for a VP token in the response is:

```
{
    "xyz123": {
        "qes": {
            "documentWithSignature": [
                "<base64-encoded document 1 with signature>",
                "<base64-encoded document 2 with signature>"
            ]
        }
    }
}
```

Note 28: Even if VP token is an OpenID4VP response, and in awareness that most data in OpenID4VP are using base64url encoding, data objects introduced by this document (e.g. the *documentWithSignature* representation) SHALL use base64 encoding (with padding), which is the common encoding in CSC context.

Another non-normative example for a VP token in the response, when the request includes a *qesRequest* with a *responseURI* is:

```
{
    "xyz123": {}
}
```

Another non-normative example DCQL query using the X.509 format is:

```
{
    "credentials": [
        {
            "id": "abc234",
            "format": "https://cloudsignatureconsortium.org/2025/x509",
            "meta": {
                "certificateFingerprints": [
                    {
                        "hashValue": "Jal8JexCNec2FBjKNGNM6WSiiE44NWjbvJBHav+vCXU=",
                        "hashAlgorithmOID": "2.16.840.1.101.3.4.2.1"
                    },
                    {
                        "hashValue": "ax8xsy0gdqyJtxmcw8xgZ85DbiC905oKdYBQspddxmk=",
                        "hashAlgorithmOID": "2.16.840.1.101.3.4.2.1"
                    }
                ]
            }
        },
        {
            "id": "def345",
            "format": "https://cloudsignatureconsortium.org/2025/x509",
            "meta": {
                "keys": [
                    {
                        "algo": "1.2.840.10045.4.3.2",
                        "curve": "1.2.840.10045.3.1.7"
                    }
                ]
            }
        }
    ]
}
```

Note 29: Even if the DCQL response in general is defined by OpenID4VP, and in awareness that most data in OpenID4VP are using base64url encoding, data objects added to the response by this document (e.g. *certificateFingerprint*) SHALL use base64 encoding (with padding), which is the common encoding in CSC context.