

v1.0.1

Summary

This document implements Mobile Driving Licence use case and Qualified Electronic Attestations of Attributes Provider from [EUDI Architecture and Reference Framework](#). [EUDI-ARF](#) defines the architecture of the EUDI ecosystem and the set of technical standards planned to ensure interoperability among different parties.

The aim is to ensure that the Estonian (Q)EAA issuer implementation is interoperable within the EUDI ecosystem across all member states, while meeting the necessary security, privacy, and user experience standards.

Versions

Version	Date	Changes
1.0.0	30.11.2023	EUDI (Q)EAA Provider Technical Documentation
1.0.1	07.12.2023	Token request redirect_uri parameter description fix. Removed expires_in from token response. Removed the max value requirement for c_nonce_expires_in parameter.

Keywords

This document uses the capitalized keywords **MUST**, **MUST NOT**, **REQUIRED**, **SHALL**, **SHALL NOT**, **SHOULD**, **SHOULD NOT**, **RECOMMENDED**, **MAY** and **OPTIONAL** as specified in [RFC 2119](#) to indicate requirements, recommendations and options specified in this document.

1. Requirements and Scope

EUDI-ARF Requirements

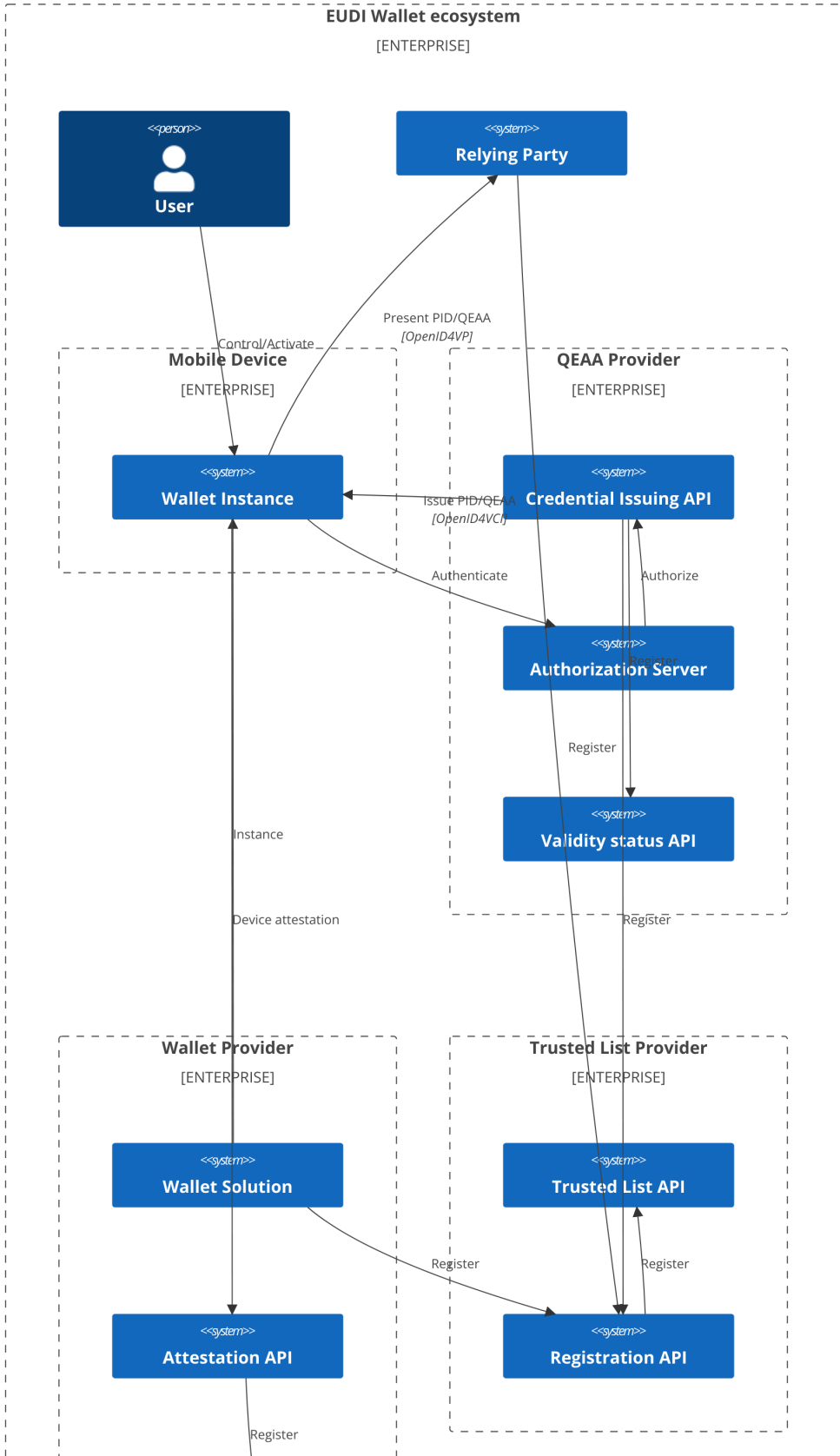
[EUDI Architecture and Reference Framework](#) Sections 5 and 6 specify the requirements for PID and (Q)EAA Providers and EUDI Wallet Solution implementers.

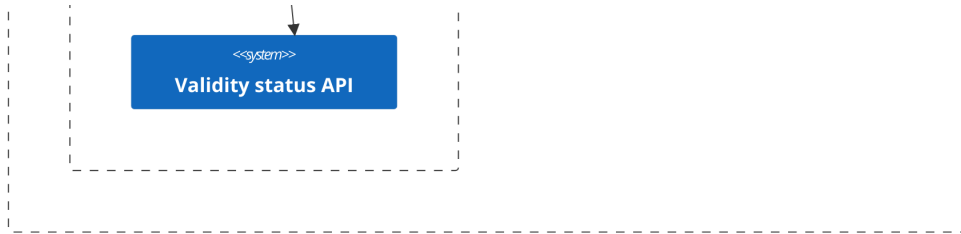
Implementation Scope

1. Only QEAA Provider related requirements **SHALL** be considered.
2. Only Mobile Driving Licence (mDL) use case **SHALL** be considered.
3. Only Type 1 configuration requirements **SHALL** be considered.
4. Only [ISO/IEC 18013-5:2021](#) data model **SHALL** be considered, due to EC Regulation 2023_127 (4th Driving License Regulation). This affects [EU DI-ARF](#) Section 5.2.1 requirements 6,7,9 and 10 and Section 6.5.3 Attestation exchange Protocol - 7, Data model -2, PID & (Q)EAA formats - 1 and Signature formats -1 requirements.
5. Only same-device issuing ([OPENID4VCI](#)) and presentation ([OPENID4VP](#)) flows **SHALL** be considered.
6. Pseudonymous authentication ([SIOPv2](#)) **SHALL NOT** be considered in [OpenID4VP](#) attestation exchange protocol. This affects [EUDI-ARF](#) Section 6.5.3 Attestation exchange Protocol - 1 requirement.
7. Trusted List mechanism to publish and obtain information about authoritative parties, e.g. Issuers of PID, (Q)EAA and Relying Parties as defined in [EUDI-ARF](#) Section 6.2 **SHALL NOT** be considered^[^1].
8. Relying Party **MUST** be authenticated to Wallet Instance in [OPENID4VP](#) presentation flow^[^1].
9. Wallet Instance **SHALL NOT** be required to authenticate to Relying Party in [OPENID4VP](#) presentation flow^[^1].
10. The User **SHALL** have a valid Wallet Instance Attestation stored in a Wallet Instance^[^1].
11. The User **SHALL** have a valid Person Identification Data Attestation stored in a Wallet Instance^[^1].
12. The Authorization Server and Credential Issuer **SHALL** be considered as separate entities in [OPENID4VCI](#) protocol implementation.
13. The authorization response in [OPENID4VP](#) flow **SHALL NOT** be encrypted.

[^1]: Implementation details are still under discussion within the EUDI Wallet ecosystem.

2. Components





3. Attestations

3.1. mDL

MDOC-CBOR Format

This section describes the structure, type, data element identifiers and logical organisation of the mandatory attributes of the CBOR encoded Mobile Driving License (mDL) attestation.

mDL data model is described in section 7 of [ISO/IEC 18013-5:2021](#), the standard for the mobile driving license use case.

mDL document type and namespace

Document type and namespace are used to encapsulate the document type and the space in which the data elements are defined.

1. The document type for an mDL document SHALL be `org.iso.18013.5.1.mDL`.
2. The namespace for mDL data elements defined in [Mandatory mDL data elements](#) SHALL be `org.iso.18013.5.1`.

Mandatory mDL data elements

1. Following data elements SHALL be considered by this document and are declared mandatory by [ISO/IEC 18013-5:2021](#), 7.2.1.

Identifier	Description	Encoding
family_name	Last name, surname, or primary identifier, of the mDL holder.	tstr
given_name	First name(s), other name(s), or secondary identifier, of the mDL holder.	tstr
birth_date	Day, month and year on which the mDL holder was born.	full-date
issue_date	Date when mDL was issued.	tdate or fulldate
expiry_date	Date when mDL expires.	tdate or fulldate
issuing_country	Alpha-2 country code, as defined in ISO 3166-1, of the issuing authority's country or territory.	tstr
issuing_authority	Issuing authority name.	tstr
document_number	The number assigned or calculated by the issuing authority.	tstr
portrait	A reproduction of the mDL holder's portrait.	bstr
driving_privileges	Driving privileges of the mDL holder.	See ISO/IEC 18013-5:2021 , 7.2.4
un_distinguishing_sign	Distinguishing sign of the issuing country according to ISO/IEC 18013-1:2018 , Annex F.	tstr

CBOR Data Types

Following data types are used by [ISO/IEC 18013-5:2021](#) and are defined in [RFC 8949](#):

Type	Major types	Description
------	-------------	-------------

any		A nonspecific value that permits all CBOR values to be placed here.
bool	#7.20, #7.21	A boolean value.
bstr	#2	A byte string.
int	#0, #1, #7.25, #7.26, #7.27	An unsigned integer or a negative integer.
uint	#0	An unsigned integer.
nint	#1	A negative integer.
null, nil	#7.22	A null value.
tstr	#3	A UTF-8 text string.
tdate	#6.0, #6.1	As defined in RFC 8610 .
full-date	#6.1004, #6.100	As defined in RFC 8943 .
array	#4	An array of data items.
map	#5	A map of pairs of data items.
encoded-cbor	#6.24	Encoded CBOR data item as defined in RFC 8949 , 3.4.5.1. Tag number 24 (CBOR data item) can be used to tag the embedded byte string as a single data item encoded in CBOR format.

Document Schema

This document will specify the elements required solely for the [OpenID4VP](#) presentation flow.

Document

A root element of mDL document, that contains following elements as CBOR encoded map:

Element	Description	Encoding
docType	The document type. It MUST be set to <code>org.iso.18013.5.1.mDL</code>	tstr
issuerSigned	Contains IssuerSigned structure for returned data elements signed by the issuer.	map
deviceSigned	Contains DeviceSigned structure for returned data elements signed by the device (Wallet). It is not returned in OpenID4VCI issuance flow and is used exclusively in OpenID4VP presentation flow.	map

IssuerSigned

It contains data elements that are signed by the issuing authority. This provides integrity and authenticity for these data elements through issuer data authentication discussed in [ISO/IEC 18013-5:2021](#), 9.1.2.

Element	Description	Encoding
nameSpaces	Contains IssuerNameSpaces structure for returned data elements signed by issuer.	map
issuerAuth	Contains IssuerAuth structure for issuer data authentication.	map

IssuerNameSpaces,

The `IssuerNameSpaces` is a map type that contains namespaces as keys and array of `IssuerSignedItem`'s encoded in CBOR Tag 24 ([cbor-tags](#)) as values, representing the disclosure information for each digest within the [Mobile Security Object](#). It is not signed by [Mobile Security Object](#) and can be used to selectively disclose attestation data.

IssuerSignedItem

Element	Description	Encoding
elementIdentifier	Data element identifier. It MUST be set to any identifier specified in Mandatory mDL data elements	tstr
elementValue	Data element value.	any
digestID	Digest ID for issuer data authentication.	uint

random	Random value for issuer data authentication to prevent the digests themselves from leaking information about the data element contents.	bstr
--------	---	------

IssuerAuth

The `IssuerAuth` is an untagged value containing a `COSE_Sign1` structure, as defined in [RFC 9052](#); 4.2, where `payload` is `Mobile Security Object` encoded in `CBOR Tag 24` ([cbor-tags](#)).

A non-normative example of the `issuerAuth` in `CBOR binary encoding`:

```
a16a697373756572417574688443a10126a11821825901533082014f3081f5a0030201020208a7276d952e234ed3300a06082a8648ce3d0403
0230183116301406035504030c0d4d444f4320497465726d204341301e170d3233313030363134303230375a170d3233313030373134303230
375a3021311f301d06035504030c164d444f4320497465726d2054657374204973737565723059301306072a8648ce3d020106082a8648ce3d
0301070342000415fbbd14d4c1e102db5ef7b774bc522b4907c5f2e022029e0d7a07dac2a2d16e830cdc50ac4869c53a6ac3026a7719b3174
6ecdf5b6214371ea0176bdb6549fa320301e300c0603551d130101ff04023000300e0603551d0f0101ff040403020780300a06082a8648ce3d
0403020349003046022100aca0267164e4a07ffee123e155a03c8add49f2b58290f5ab20401c275a1e5960022100bb93d74d49a21f30ea3301
10bda124fda68156f469ea43e502a29a0b8553491559014c308201483081eea0030201020208052d609340e6a580300a06082a8648ce3d0403
0230173115301306035504030c0c4d444f4320524f4f54204341301e170d3233313030363134303230375a170d323331303037313430323037
5a30183116301406035504030c0d4d444f4320497465726d2043413059301306072a8648ce3d020106082a8648ce3d030107034200040edb17
ccfcbe549944e1cfff1d65234b1e9fa2bdeeb6cfb78cd29f66580796481731535da4f4a0a03243f1cf12e60aad8c96901ef45892052574ea798
04f3463ca3233021300f0603551d130101ff040530030101ff300e0603551d0f0101ff040403020106300a06082a8648ce3d04030203490030
46022100bb3f6e73db3f9a4ae1d9b100fb3154705e499a068fc3c13454383fa4772acd03022100bf4f2faba60376ec2e5896e722580a5f16c6
0d6fe4812ea318f4467e56611e1f5902dbd8185902d6a66776657273696f6e63312e306f646967657374416c676f726974686d675348412d32
35366c76616c756544696765737473a1716f72672e69736f2e31383031332e352e31ab0058208e794e6ba8252f23edcbdd2666cb1073b3fd0f8
771e49f0a0d011276fb047ff27015820ed2c5471ab1ccd7a2d00db3e9ae3275d75904a9bcb92bcb95dddf27b71bfb30b02582041c988cc5d7f
caalddbac193140da6f7a3a8352b68e6d56ee553a0882628d27e03582031cb399f252e056a71c28c9ca28b249c785dae244bc0f08ffa4ca4d7
0dfe16820458208444b5052ca02399ee973e0b01f605e0fc6fdb728d69cb8d3583efd0b283eec0055820f99ec3a3ba6b5d1d4dbdeefe0d2f6c
38985937920e2b993429a8f152b61d540e0658208e4222d730bdc0b514353948be7d94cf715d525be2983b12b911893df86289f6075820cef5
fc62570abdafel5ae9995c0a1ef3dd52df20d04400e52da48bf58c90e7d808582030f0c29056c8882059a65155399d79f0b250340721efd991
0ca99bc6f67981ef095820b1ba0d0db16abbaa868d9f0b07534f2933f7f3c5b32838bc9d0a8435f9ae0ad40a58208f191c2093fc55cbf9229e
e375f4569a4128b55a69b654be65d90c3f5fea7d106d6465766963654b6579496e666fa1696465766963654b6579a401022001215820bc834b
5d33cc69a120410cf9ba7c0cd08e86c9750bd8a812c2629dabc5853a0f2258208bb8dc2fd6733f623d4c662ada00e481cb7a4a154b1dfe242d
54acfaedb8b87267646f6354797065756f72672e69736f2e31383031332e352e312e6d444c6c76616c6964697479496e666fa3667369676e65
64c0781e323032332d31302d30365431343a30323a30372e3932393436373630305a6a76616c69646556e74696cc0781e323032342d31302d30355431343a30323a30372e393239
5431343a30323a30372e3932393436373630305a6a76616c69646556e74696cc0781e323032342d31302d30355431343a30323a30372e393239
3436373630305a58406b5192db86e78a1606e235d4e2ee9cd12f42cc77283c83d94d2a585c7ec3c0e0224240cec14b2b5d7e3e9ab4b72b8036
c3ae71dacd36fd8458e065bbe75fcccl
```

A non-normative example of the `issuerAuth` in `CBOR diagnostic notation`:

```

{
  "issuerAuth": [
    h
    'al0126',
    {
      33_0: [
        h
        '3082014f3081f5a0030201020208a7276d952e234ed3300a06082a8648ce3d04030230183116301406035504030c0d4d444f4320497465726
d204341301e170d3233313030363134303230375a170d3233313030373134303230375a3021311f301d06035504030c164d444f43204974657
26d2054657374204973737565723059301306072a8648ce3d020106082a8648ce3d0301070342000415fbbd14d4c1e102db5ef7b774bc522b4
907c5f2e022029e0d7a07dac2a2d16e830cdcf50ac4869c53a6ac3026a7719b31746ecdf5b6214371ea0176bdb6549fa320301e300c0603551
d130101ff04023000300e0603551d0f0101ff040403020780300a06082a8648ce3d0403020349003046022100aca0267164e4a07ffee123e15
5a03c8add49f2b58290f5ab20401c275a1e5960022100bb93d74d49a21f30ea330110bda124fda68156f469ea43e502a29a0b85534915',
        h
        '308201483081eea0030201020208052d609340e6a580300a06082a8648ce3d04030230173115301306035504030c0c4d444f4320524f4f542
04341301e170d3233313030363134303230375a170d3233313030373134303230375a30183116301406035504030c0d4d444f4320497465726
d2043413059301306072a8648ce3d020106082a8648ce3d030107034200040edb17ccfcbe549944e1cfff1d65234b1e9fa2bdeeb6cfb78cd29f
66580796481731535da4f4a0a03243f1cf12e60aad8c96901ef45892052574ea79804f3463ca3233021300f0603551d130101ff04053003010
1ff300e0603551d0f0101ff040403020106300a06082a8648ce3d0403020349003046022100bb3f6e73db3f9a4ae1d9b100fb3154705e499a0
68fc3c13454383fa4772acd03022100bf4f2faba60376ec2e5896e722580a5f16c60d6fe4812ea318f4467e56611e1f'
      ]
    }
  ],
  h
  'd8185902d6a66776657273696f6e63312e306f646967657374416c676f726974686d675348412d3235366c76616c756544696765737473a17
16f72672e69736f2e31383031332e352e31ab0058208e794e6ba8252f23edcbdb2666cb1073b3fd0f8771e49f0a0d011276fb047ff27015820e
d2c5471lab1ccd7a2d00db3e9ae3275d75904a9bcb92bcb95dddf27b71bfb30b02582041c988cc5d7fcaa1ddbacc193140da6f7a3a8352b68e6d
56ee553a0882628d27e03f582031cb399f252e056a71c28c9ca28b249c785dae244bc0f08ffa4ca4d70dfe16820458208444b5052ca02399ee9
73e0b01f605e0fc6fdb728d69cb8d3583efd0b283eec0055820f99ec3a3ba6b5d1d4dbdeefe0d2f6c38985937920e2b993429a8f152b61d540
e0658208e4222d730bdc0b514353948be7d94cf715d525be2983b12b911893df86289f6075820cef5fc62570abdaf15ae9995c0a1ef3dd52d
f20d04400e52da48bf58c90e7d808582030f0c29056c8882059a65155399d79f0b250340721efd9910ca99bc6f67981ef095820b1ba0d0db16
abbaa868d9f0b07534f2933f7f3c5b32838bc9d0a8435f9ae0ad40a58208f191c2093fc55cbf9229ee375f4569a4128b55a69b654be65d90c3
f5fea7d106d6465766963654b6579496e666fa1696465766963654b6579a401022001215820bc834b5d33cc69a120410cf9ba70cd08e86c97
50bd8a812c2629dabc5853a0f2258208bb8dc2fd6733f623d4c662ada00e481cb7a4a154b1dfe242d54acfaed8b87267646f6354797065756
f72672e69736f2e31383031332e352e312e6d444c6c76616c6964697479496e666fa3667369676e6564c0781e323032332d31302d303654313
43a30323a30372e3932393436373630305a6976616c696446726f6dc0781e323032332d31302d30365431343a30323a30372e3932393436373630305a' ,
  h
  '6b5192db86e78a1606e235d4e2ee9cd12f42cc77283c83d94d2a585c7ec3c0e0224240cecc14b2b5d7e3e9ab4b72b8036c3ae71dacd36fd845
8e065bbe75fcccl'
}

```

COSE_Sign1 structure

CDDL ([RFC 8610](#)) fragment that represents the COSE_Sign1 as defined in [RFC 9052](#), 4.2:

```

COSE_Sign1 = [
  Headers,
  payload : bstr / nil,
  signature : bstr
]

Headers = (
  protected : empty_or_serialized_map,
  unprotected : header_map
)

header_map = {
  Generic_Headers,
  * label => values
}

empty_or_serialized_map = bstr .cbor header_map / bstr .size 0

```

Protected header

A CBOR bstr that encapsulates a CBOR map of protected headers. These headers are covered by the signature.

1. It MUST contain alg header as defined in [RFC 9052](#), 3.1 and [iana-cose](#)
2. Other elements should not be present in the protected header as required by [ISO/IEC 18013-5:2021](#), 9.1.2.4.

Non-normative example of protected header, that contains alg header parameter:

CBOR binary encoding:

43a10126

Corresponding CBOR diagnostic notation [RFC 8610](#):

h'a10126'

a10126 in CBOR diagnostic notation, where 1 denotes alg header as defined in [iana-cose](#), Headers section and -7 denotes algorithm ES256 as defined in [iana-cose](#), Algorithms section. The exact steps how to decode CBOR is discussed in [RFC 8949](#).

```
{
  1: -7
}
```

Unprotected header

A CBOR map of unprotected headers. These headers are not covered by the signature, so they can be changed without invalidating the signature.

1. It MUST contain x5chain header parameter as defined in [RFC 9360](#) and required by [ISO/IEC 18013-5:2021](#), 9.1.2.4.
2. If a single certificate is conveyed in x5chain, it SHALL be placed in a CBOR byte string.
3. If multiple certificates are conveyed in x5chain, a CBOR array of byte strings SHALL be used, with each certificate being in its own byte string.
4. It MUST be included as an unprotected header element as suggested in [ISO/IEC 18013-5:2021](#), 9.1.2.4. It is included in the unprotected header to enable the Holder to update the X.509 certificate chain, without breaking the signature.
5. It MUST include at least one DER-encoded certificate and MAY contain more.
6. First certificate in x5chain MUST be mDL issuer signing certificate.
7. Certificates contained in x5chain header parameter MUST use certificate profiles as described in [ISO/IEC 18013-5:2021](#), Annex B.
8. The Issuing Authority Certificate Authority (IACA) root certificate MUST NOT be included in the x5chain element.
9. The Issuing Authority infrastructure SHALL use one of the following signature algorithms for calculating the signature over the MSO: "ES256" (ECDSA with SHA-256), "ES384" (ECDSA with SHA-384), "ES512" (ECDSA with SHA-512) or "EdDSA" (EdDSA). "ES256" shall be used with curves P-256 and brainpoolP256r1. "ES384" shall be used with curves P-384, brainpoolP320r1 and brainpoolP384r1. "ES512" shall be used with curves P-521 and brainpoolP512r1. "EdDSA" shall be used with curves Ed25519 and Ed448. The Verifier MUST support all of these signature algorithms and curves.

Non-normative example of unprotected header, that contains x5chain header parameter in CBOR binary encoding:

```
a11821825901533082014f3081f5a0030201020208a7276d952e234ed3300a06082a8648ce3d04030230183116301406035504030c0d4d444f
4320497465726d204341301e170d3233313030363134303230375a170d3233313030373134303230375a3021311f301d06035504030c164d44
4f4320497465726d2054657374204973737565723059301306072a8648ce3d020106082a8648ce3d0301070342000415fbbd14d4c1e102db5e
f7b774bc522b4907c5f2e022029e0d7a07dac2a2d16e830cddf50ac4869c53a6ac3026a7719b31746ecdf5b6214371ea0176bdb6549fa32030
1e300c0603551d130101ff04023000300e0603551d0f0101ff040403020780300a06082a8648ce3d0403020349003046022100aca0267164e4
a07ffee123e155a03c8add49f2b58290f5ab20401c275a1e5960022100bb93d74d49a21f30ea330110bda124fda68156f469ea43e502a29a0b
8553491559014c308201483081eea0030201020208052d609340e6a580300a06082a8648ce3d04030230173115301306035504030c0c4d444f
4320524f4f54204341301e170d3233313030363134303230375a170d3233313030373134303230375a30183116301406035504030c0d4d444f
4320497465726d2043413059301306072a8648ce3d020106082a8648ce3d030107034200040edb17ccfcbce549944e1cfff1d65234b1e9fa2bde
eb6cbf78cd29f66580796481731535da4f4a0a03243f1cf12e60aad8c96901ef45892052574ea79804f3463ca3233021300f0603551d130101
ff040530030101ff300e0603551d0f0101ff040403020106300a06082a8648ce3d0403020349003046022100bb3f6e73db3f9a4ae1d9b100fb
3154705e499a068f3c13454383fa4772acd03022100bf4f2faba60376ec2e5896e722580a5f16c60d6fe4812ea318f4467e56611e1f
```

Corresponding example in CBOR diagnostic notation, containing two certificates in DER encoding:

```
{
  33_0: [
    h
    '3082014f3081f5a0030201020208a7276d952e234ed3300a06082a8648ce3d04030230183116301406035504030c0d4d444f4320497465726
d204341301e170d3233313030363134303230375a170d3233313030373134303230375a3021311f301d06035504030c164d444f43204974657
26d2054657374204973737565723059301306072a8648ce3d020106082a8648ce3d0301070342000415fbbd14d4c1e102db5ef7b774bc522b4
907c5f2e022029e0d7a07dac2a2d16e830cddf50ac4869c53a6ac3026a7719b31746ecdf5b6214371ea0176bdb6549fa320301e300c0603551
d130101ff04023000300e0603551d0f0101ff040403020780300a06082a8648ce3d0403020349003046022100aca0267164e4a07ffee123e15
5a03c8add49f2b58290f5ab20401c275a1e5960022100bb93d74d49a21f30ea330110bda124fda68156f469ea43e502a29a0b85534915' ,
    h
    '308201483081eea0030201020208052d609340e6a580300a06082a8648ce3d04030230173115301306035504030c0c4d444f4320524f4f542
04341301e170d3233313030363134303230375a170d3233313030373134303230375a30183116301406035504030c0d4d444f4320497465726
d2043413059301306072a8648ce3d020106082a8648ce3d030107034200040edb17ccfcbce549944e1cfff1d65234b1e9fa2bdeeb6cbf78cd29f
66580796481731535da4f4a0a03243f1cf12e60aad8c96901ef45892052574ea79804f3463ca3233021300f0603551d130101ff04053003010
1ff300e0603551d0f0101ff040403020106300a06082a8648ce3d0403020349003046022100bb3f6e73db3f9a4ae1d9b100fb3154705e499a0
68f3c13454383fa4772acd03022100bf4f2faba60376ec2e5896e722580a5f16c60d6fe4812ea318f4467e56611e1f'
  ]
}
```

Contained certificates in PEM encoding:

```

-----BEGIN CERTIFICATE-----
MIIBTzCB9aADAgECAgInJ22VLI00zAKBggqhkJOPQDAjAYMRyWfAYDVQDDA1NRE9DIE10ZXJt
IENBMB4XDTEzMTAwNjE0MDIwNl0XDTEzMTAwNzE0MDIwNl0wITFfMB0GALUEAwWTURPQyBjGvY
bSBuZXN0IElzc3VlcjBZMBMBGByqGSM49AgEGCCqGSM49AwEHA0IABBx7vRTUweEC2173t3S8UitJ
B8Xy4CICng16B9rCotFugwzc9QrEhpxTpqwwJqdxmzF0bs31tiFDceobDr22VJ+jIDAeMAwGALUd
EwEB/wQCMAADgYDVR0PAQH/BAQDAgeAMAoGCCqGSM49BAMCA0kAMEYCIQCsoCZxZOSgf/7hI+FV
oDyK3UnytYKQ9asgQBwnWh5ZYAIhALuT101Joh8w6jMBEL2hJP2mgVb0aepD5QKimguFU0kV
-----END CERTIFICATE-----

```

```

-----BEGIN CERTIFICATE-----
MIIBSDCB7qADAgECAggFLWCTQOalgDAKBggqhkJOPQDAjAXMRUwEwYDVQDDAxNRE9DIFJPTlQg
Q0EwHhcNMjMxMDA2MTQwMjA3WjcNMjMxMDA3MTQwMjA3WjAYMRyWfAYDVQDDA1NRE9DIE10ZXJt
IENBMBFkEwYHKOZIZj0CAQYIKoZIZj0DAQcDQgAEDtsXzPy+VJ1E4c/x11I0sen6K97rbPt4zSn2
ZYB5ZIFzFTXaT0oKAYQ/HPEuYKrYyWk8B7QWJIFJXTqeYBPNPKMjMCEwDwYDVR0TAQH/BAUwAwEB
/zAOBgNVHQ8BAf8EBAMCAQYwCgYIKoZIZj0EAWIDSQAwrGThALs/bnPbP5pK4dmxAPsxVHBeSzoG
j8PBNFQ4P6R3Ks0DAiEAv08vq6YDduwuWJbnIlgKXxbGDW/kgS6jGPRGf1ZhHh8=
-----END CERTIFICATE-----

```

Payload

Contains [Mobile Security Object](#) encoded in CBOR Tag 24 ([cbor-tags](#))

Signature

The actual cryptographic signature as `bstr`. The signature is computed over the protected headers and the `payload`.

Mobile Security Object

Element	Description	Encoding
version	MSO version. It MUST be set to 1.0.	tstr
docType	The document type. It MUST be set to <code>org.iso.18013.5.1.mDL</code>	tstr
digestAlgorithm	Message digest algorithm used. It MUST be set to SHA-256, SHA-384 or SHA-512.	tstr
valueDigests	Digests of all data elements per namespace. It contains <code>digestsID</code> 's as keys and corresponding data element digest as values. It MUST contain namespace <code>org.iso.18013.5.1</code> . It MUST contain all Mandatory mDL data elements digests.	map
deviceKeyInfo	Contains DeviceKeyInfo structure for the mdoc authentication public key and information related to this key.	map
validityInfo	Contains ValidityInfo structure for the validity of the MSO and its signature.	map

DeviceKeyInfo

1. It MUST contain public key defined in WIA `cnf` claim.

Element	Description	Encoding
deviceKey	It contains the public part of the key pair used for mdoc authentication as untagged <code>COSE_Key</code> structure. Further requirements are defined in ISO/IEC 18013-5:2021 , 9.1.5.2.	map

CDDL ([RFC 8610](#)) fragment that represents the `COSE_Key` as defined in [RFC 9052](#), 7, [iana-cose](#) and [ISO/IEC 18013-5:2021](#), 9.1.5.2:

```

COSE_Key = {
  2 => int,           ; kty: key type
  -1 => int,          ; crv: EC identifier - From the "COSE Elliptic Curves" registry [iana-cose]
  -2 => bstr,         ; x: value of x-coordinate [iana-cose]
  ? -3 => bstr/ bool ; y: value or sign bit of y-coordinate; only applicable for EC2 key types [iana-cose]
}

```

ValidityInfo

Element	Description	Encoding
signed	Signature timestamp.	tdate
validFrom	Validity start timestamp.	tdate
validUntil	Validity end timestamp.	tdate

A non-normative example of the `Mobile Security Object` in CBOR binary encoding:

```
d8185902d6a66776657273696f6e63312e306f646967657374416c676f726974686d675348412d3235366c76616c756544696765737473a171
6f72672e69736f2e31383031332e352e31ab0058208e794e6ba8252f23edcbd2666cb1073b3fd0f8771e49f0a0d011276fb047ff27015820ed
2c5471ablccd7a2d00db3e9ae3275d75904a9bcb92bcb95ddd27b71bfb30b02582041c988cc5d7fcaald1dbac193140da6f7a3a8352b68e6d5
6ee553a0882628d27e03582031cb399f252e056a71c28c9ca28b249c785dae244bc0f08ffa4ca4d70dfe16820458208444b5052ca02399ee97
3e0b01f605e0fc6fdb728d69cb8d3583efd0b283e0055820f99ec3a3ba6b5d1d4dbdeefe0d2f6c38985937920e2b993429a8f152b61d540e
0658208e4222d730bdc0b514353948be7d94cf715d525be2983b12b911893df86289f6075820cef5fc62570abdafa15ae9995c0a1ef3dd52df
20d04400e52da48bf58c90e7d808582030f0c29056c8882059a65155399d79f0b250340721efd9910ca99bc6f67981ef095820b1ba0d0db16a
bbaa868d9f0b07534f2933f7f3c5b32838bc9d0a8435f9ae0ad40a58208f191c2093fc55cbf9229ee375f4569a4128b55a69b654be65d90c3f
5fea7d106d6465766963654b6579496e666fal696465766963654b6579a401022001215820bc834b5d33cc69a120410cf9ba7c0cd08e86c975
0bd8a812c2629dabc5853a0f2258208bb8dc2fd6733f623d4c662ada00e481cb7a4a154b1dfe242d54acfaedb8b87267646f6354797065756f
72672e69736f2e31383031332e352e312e6d44c6c76616c6964697479496e666fa3667369676e6564c0781e323032332d31302d30365431343a30323a30372e39323934363736
3a30323a30372e3932393436373630305a6976616c69646726f6dc0781e323032332d31302d30365431343a30323a30372e39323934363736
30305a6a76616c6964556e74696cc0781e323032342d31302d30355431343a30323a30372e3932393436373630305a
```

A non-normative example of the Mobile Security Object in CBOR diagnostic notation:

```
24_0(<<{
  "version": "1.0",
  "digestAlgorithm": "SHA-256",
  "valueDigests": {
    "org.iso.18013.5.1": {
      0: h
      '8e794e6ba8252f23edcbd2666cb1073b3fd0f8771e49f0a0d011276fb047ff27',
      1: h
      'ed2c5471ablccd7a2d00db3e9ae3275d75904a9bcb92bcb95ddd27b71bfb30b',
      2: h
      '41c988cc5d7fcaald1dbac193140da6f7a3a8352b68e6d56ee553a0882628d27e',
      3: h
      '31cb399f252e056a71c28c9ca28b249c785dae244bc0f08ffa4ca4d70dfe1682',
      4: h
      '8444b5052ca02399ee973e0b01f605e0fc6fdb728d69cb8d3583efd0b283e0',
      5: h
      'f99ec3a3ba6b5d1d4dbdeefe0d2f6c38985937920e2b993429a8f152b61d540e',
      6: h
      '8e4222d730bdc0b514353948be7d94cf715d525be2983b12b911893df86289f6',
      7: h
      'cef5fc62570abdafa15ae9995c0a1ef3dd52df20d04400e52da48bf58c90e7d8',
      8: h
      '30f0c29056c8882059a65155399d79f0b250340721efd9910ca99bc6f67981ef',
      9: h
      'b1ba0d0db16abbaa868d9f0b07534f2933f7f3c5b32838bc9d0a8435f9ae0ad4',
      10: h
      '8f191c2093fc55cbf9229ee375f4569a4128b55a69b654be65d90c3f5fea7d10'
    }
  },
  "deviceKeyInfo": {
    "deviceKey": {
      1: 2,
      -1: 1,
      -2: h
      'bc834b5d33cc69a120410cf9ba7c0cd08e86c9750bd8a812c2629dabc5853a0f',
      -3: h
      '8bb8dc2fd6733f623d4c662ada00e481cb7a4a154b1dfe242d54acfaedb8b872'
    }
  },
  "docType": "org.iso.18013.5.1.mDL",
  "validityInfo": {
    "signed": 0(
      "2023-10-06T14:02:07.929467600Z"
    ),
    "validFrom": 0(
      "2023-10-06T14:02:07.929467600Z"
    ),
    "validUntil": 0(
      "2024-10-05T14:02:07.929467600Z"
    )
  }
}>>)
```

Validation

1. mDL MUST be validated as directed in [ISO/IEC 18013-5:2021](#), 9.3.

Example mDL

A non-normative example of the mDL document in CBOR binary encoding:


```
7b8b9bac2c3c4c5c6c7c8c9cad2d3d4d5d6d7d8d9dae2e3e4e5e6e7e8e9eaf2f3f4f5f6f7f8f9affda000c03010002110311003f00fb4bc69
e365d726f136adad78b2ebc0ff0d7c31789a65d5d589f26e751bb2c88e4ce033c502492c718f2c2bb48ae4b040034576bff00086ea5e256f87
be2cd5b5ed57c2620975df08eb5a94fa8acb0c91f9aa8935c334914cd16e6460e549003a9eab5afaee7f03dd6bbe07bd8f4463aa6b371ad693
178914ae9facdbdc4ad3cf6866daca970933b900abl1dbe5b0561b82bf5cf885e21d1f58d614e93e1cd33c5dadd90b2d23c37a6ce2fb52bebb0
760b9bd9163023b68411c90e02331660c563ad083dc345d5ed7c41a3d8ea961289ec6fa08ee60947478dd432b7e208a2b3fc0be178bc11e09f
0f7872090cb0e91a75be9e921ce5962896307924f217b9345202f6b3a1e9de22d3e5b0d5b4fb5d4ec65c7996b790acd13e0e465581079aa1e1
5f01f867c0b6f241elbf0ee93elf864c6f8f4ab18ad95b1d32114668a29f403768a28a407ffd9'
```

```
}
>>),
24_0(<<
{
  "digestID": 9,
  "random": h
  'e9bfd3a7a6e15cla74ee61de44b03e99',
  "elementIdentifier": "driving_privileges",
  "elementValue": [
    "A",
    "B"
  ]
}
>>),
24_0(<<
{
  "digestID": 10,
  "random": h
  'dde8b53a73c496f8017cafec8fbfd852',
  "elementIdentifier": "un_distinguishing_sign",
  "elementValue": "EST"
}
>>)
]
},
"issuerAuth": [
  h
  'a10126',
  {
    33_0: [
      h
```

```
'3082014f3081f5a0030201020208a7276d952e234ed3300a06082a8648ce3d04030230183116301406035504030c0d4d444f4320497465726
d204341301e170d3233313030363134303230375a170d3233313030373134303230375a3021311f301d06035504030c164d444f43204974657
26d2054657374204973737565723059301306072a8648ce3d020106082a8648ce3d0301070342000415fbbd14d4c1e102db5ef7b774bc522b4
907c5f2e022029e0d7a07dac2a2d16e830cddf50ac4869c53a6ac3026a7719b31746ecdf5b6214371ea0176bdb6549fa320301e300c0603551
d130101fff04023000300e0603551d0f0101fff040403020780300a06082a8648ce3d0403020349003046022100aca0267164e4a07ffee123e15
5a03c8add49f2b58290f5ab20401c275a1e5960022100bb93d74d49a21f30ea330110bda124fda68156f469ea43e502a29a0b85534915',
h
```

```
'308201483081eea0030201020208052d609340e6a580300a06082a8648ce3d04030230173115301306035504030c0c4d444f4320524f4f542
04341301e170d3233313030363134303230375a170d3233313030373134303230375a30183116301406035504030c0d4d444f4320497465726
d2043413059301306072a8648ce3d020106082a8648ce3d030107034200040edb17ccfcbe549944e1cfff1d65234b1e9fa2bdeeb6cfb78cd29f
66580796481731535da4f4a0a03243f1cf12e60aad8c96901ef45892052574ea79804f3463ca3233021300f0603551d130101fff04053003010
1ff300e0603551d0f0101fff040403020106300a06082a8648ce3d0403020349003046022100bb3f6e73db3f9a4ae1d9b100fb3154705e499a0
68fc3c13454383fa4772acd03022100b4f2faba60376ec2e5896e722580a5f16c60d6fe4812ea318f4467e56611e1f'
]
},
h
```

```
'd8185902d6a66776657273696f6e63312e306f646967657374416c676f726974686d675348412d3235366c76616c756544696765737473a17
16f72672e69736f2e31383031332e352e31ab0058208e794e6ba8252f23edcbdb2666cb1073b3fd0f8771e49f0a0d011276fb047ff27015820e
d2c5471lablccd7a2d00db3e9ae3275d75904a9bcb92bcb95dddff27b71bfb30b02582041c988cc5d7fcaa1ddbacl93140da6f7a3a8352b68e6d
56ee553a0882628d27e03582031cb399f252e056a71c28c9ca28b249c785dae244bc0f08ffa4ca4d70dfe16820458208444b5052ca02399ee9
73e0b01f605e0fc6fdb728d69cb8d3583efd0b283eec0055820f99ec3a3ba6b5d1d4dbdeefe0d2f6c38985937920e2b993429a8f152b61d540
e0658208e4222d730bdc0b514353948be7d94cf715d525be2983b12b911893df86289f6075820cef5fc62570abdafe15ae9995c0alef3dd52d
f20d04400e52da48bf58c90e7d808582030f0c29056c8882059a65155399d79f0b250340721ef910ca99bc6f67981ef095820blba0d0db16
abbaa868d9f0b07534f2933f7f3c5b32838bc9d0a8435f9ae0ad40a58208f191c2093fc55cbf9229ee375f4569a4128b55a69b654be65d90c3
f5fea7d106d6465766963654b6579496e666fa1696465766963654b6579a401022001215820bc834b5d33cc69a120410cf9ba7c0cd08e86c97
50bd8a812c2629dabc5853a0f2258208bb8dc2fd6733f623d4c662ada0e481cb7a4a154b1dfe242d54acfaedb8b87267646f6354797065756
f72672e69736f2e31383031332e352e312e6d444c6c76616c6964697479496e666fa3667369676e6564c0781e323032332d31302d30365431343a30323a30372e3932393436373
43a30323a30372e3932393436373630305a6976616c696446726f6dc0781e323032332d31302d30365431343a30323a30372e3932393436373
630305a6a76616c6964565e74696cc0781e323032342d31302d30355431343a30323a30372e3932393436373630305a',
h
```

```
'6b5192db86e78a1606e235d4e2ee9cd12f42cc77283c83d94d2a585c7ec3c0e0224240cec14b2b5d7e3e9ab4b72b8036c3ae71dacd36fd845
8e065bbe75fccc1'
]
}
}
```

mDL Presentation

When mDL is presented during [OpenID4VP](#) presentation flow, the Wallet MUST bind authorization request `client_id` and `nonce` values to the presented mDL. This is accomplished by adding `deviceSigned` element to mDL document that SHALL contain the required `client_id` and `nonce` as device signed elements. Current [ISO/IEC 18013-5:2021](#) specification has not considered the requirements of [OpenID4VP](#) and therefore current specification extends [SessionTranscript](#) structure with new Handover type [OpenID4VPHandover](#) to support `client_id` and `nonce` as required by [OpenID4VP](#).

DeviceSigned

It contains data elements that are signed by the holder with the key defined in [Mobile Security Object](#). This provides integrity and authenticity for these data elements through `mdoc authentication` discussed in [ISO/IEC 18013-5:2021](#), 9.1.3.

Element	Description	Encoding
<code>nameSpaces</code>	It MUST be set to empty <code>map</code> encoded in CBOR Tag 24 (cbor-tags).	<code>encoded-cbor</code>
<code>deviceAuth</code>	Contains DeviceAuth structure for <code>mdoc authentication</code> .	<code>map</code>

DeviceAuth

Element	Description	Encoding
<code>deviceSignature</code>	Contains DeviceSignature structure to authenticate the <code>mdoc</code> with <code>mdoc ECDSA/EdDSA authentication</code> .	<code>array</code>

DeviceSignature

The `DeviceSignature` is a `COSE_Sign1` structure with [DeviceAuthentication](#) structure as detached payload - meaning only signature SHALL be conveyed and the validator MUST reconstruct the payload to validate the signature.

DeviceAuthentication

The `DeviceAuthentication` is a `array` type encoded in CBOR Tag 24 ([cbor-tags](#)) containing elements in following order:

Element	Description	Encoding
<code>DeviceAuthentication</code>	It MUST be set to <code>DeviceAuthentication</code> .	<code>tstr</code>
<code>SessionTranscript</code>		<code>array</code>
<code>DocType</code>	It MUST be set to <code>org.iso.18013.5.1.mDL</code>	<code>tstr</code>
<code>DeviceNameSpacesBytes</code>	It MUST be set to empty <code>map</code> encoded in CBOR Tag 24 (cbor-tags).	<code>encoded-cbor</code>

SessionTranscript

The `SessionTranscript` as described in [ISO/IEC 18013-5:2021](#) Section 9.1.5.1 is a `array` type containing following elements:

Element	Description	Encoding
<code>DeviceEngagementBytes</code>	It MUST be set to <code>null</code> .	<code>array</code>
<code>EReaderKeyBytes</code>	It MUST be set to <code>null</code> .	<code>null</code>
<code>Handover</code>	It MUST be set to OpenID4VPHandover structure.	<code>array</code>

OpenID4VPHandover

The proposed `OpenID4VPHandover` type requires, that `DeviceEngagementBytes` and `EReaderKeyBytes` are set to `null` in `SessionTranscript`.

Element	Description	Encoding
<code>name</code>	It MUST be set to <code>openID4VPHandover</code> .	<code>tstr</code>
<code>aud</code>	The intended audience. It MUST be set to <code>client_id</code> value from Verifier authorization request.	<code>tstr</code>
<code>nonce</code>	It MUST be set to <code>nonce</code> value from Verifier authorization request.	<code>tstr</code>

A non-normative example of the `deviceSigned` with detached payload in CBOR diagnostic notation:

A non-normative example of the mDL document with selective disclosure and device authentication in CBOR diagnostic notation:

```
{
  "docType": "org.iso.18013.5.1.mDL",
  "issuerSigned": {
    "nameSpaces": {
      "org.iso.18013.5.1": [
        24_0(<<{
          "digestID": 7,
          "random": h'6f1c39f27d502b240aec0ef8c57facb7',
          "elementIdentifier": "document_number",
          "elementValue": "ET000000",
        }>>),
      ],
    },
  },
  "issuerAuth": [
    h'a10126',
    {
      33_0: [
        h'3082014f3081f5a0030201020208c752e53f9d66c8cc300a06082a8648ce3d04030230183116301406035504030c0d4d444f432049746572
        6d204341301e170d3233313032363132353033345a170d3233313032373132353033345a3021311f301d06035504030c164d444f4320497465
        726d2054657374204973737565723059301306072a8648ce3d020106082a8648ce3d030107034200047dc64bab03c0b0396671728f720022f6
        a79bae782c4017b02e88a1b52bbeb2399e96c3df934d88763b7ab92ce5fc2b14cf812bdcd429b7fb396b7eff9a29245ea320301e300c060355
        1d130101ff04023000300e0603551d0f0101ff040403020780300a06082a8648ce3d0403020349003046022100ed88ffe14216147a79c10d72
        9602b02f57610d365dfba7032c23f65b4ef72070022100f279053afda5e124956bdc07ffaa903188fe911d69f03e0657bbb5aaa5e6c16a',
        ],
      ],
    },
  ],
  "deviceSigned": {
    "nameSpaces": 24_0(<<{}>>),
    "deviceAuth": {
      "deviceSignature": [
        h'a10126',
        {33_0: []},
        null,
      ],
    },
  },
  "deviceAuth": [
    h'a40589563d7ed6836eb4a066be0fa0b7fcb60c41ecc44cc04e3cb2b651db6681e6ae89224fa615c825b81b021cc03009af4465aae8e8235
    3fafd154bb70432f',
    ],
  ],
}
```

3.2. PID

Requirements

1. The user SHALL have a valid Person Identification Data (PID) Attestation stored in a Wallet Instance.
2. The PID Attestation MUST conform to the issuing requirements defined in [EUDI-ARF](#).

To support PID authentication in QEAA issuing flow the following non-normative example of the PID Attestation is used:


```

{
  "docType": "eu.europa.ec.eudiw.pid.1",
  "issuerSigned": {
    "nameSpaces": {
      "eu.europa.ec.eudiw.pid.1": [
        24_0(<<{
          "digestID": 0,
          "random": h'cc019b417be2b900042a41a2ac4e8c2a',
          "elementIdentifier": "family_name",
          "elementValue": "Mary",
        }>>),
        24_0(<<{
          "digestID": 1,
          "random": h'bb0cd140583a6da24dd0865fa69acb38',
          "elementIdentifier": "given_name",
          "elementValue": "Ann",
        }>>),
        24_0(<<{
          "digestID": 2,
          "random": h'ef18bf0a12c947aac2ff267712f1d1dd',
          "elementIdentifier": "birth_date",
          "elementValue": 1004_1("1980-01-01"),
        }>>),
        24_0(<<{
          "digestID": 3,
          "random": h'c5b14dbc1c38a086b999b8d6de3a2a29',
          "elementIdentifier": "age_over_18",
          "elementValue": "1",
        }>>),
        24_0(<<{
          "digestID": 4,
          "random": h'6144769b053ab558801e70d2df199645',
          "elementIdentifier": "unique_id",
          "elementValue": "60001019906",
        }>>),
        24_0(<<{
          "digestID": 5,
          "random": h'e58e73e12e41964cdcf23bfdb76cfa3f',
          "elementIdentifier": "issuance_date",
          "elementValue": 1004_1("2015-01-01"),
        }>>),
        24_0(<<{
          "digestID": 6,
          "random": h'f277b610f9f5e9a29816eb30b2bd41d5',
          "elementIdentifier": "expiry_date",
          "elementValue": 1004_1("2025-01-01"),
        }>>),
        24_0(<<{
          "digestID": 7,
          "random": h'050c1901b19442d665887e6a567f1014',
          "elementIdentifier": "issuing_authority",
          "elementValue": "PPA",
        }>>),
        24_0(<<{
          "digestID": 8,
          "random": h'befb9d4003e90eef163b306f4b0a6e9e',
          "elementIdentifier": "issuing_country",
        }>>),
      ],
    },
  },
  h'd81859029ca66776657273696f6e63312e306f646967657374416c676f726974686d675348412d3235366c76616c756544696765737473a1781865752e6575726f70612e65632e65756469772e7069642e31a90058205aa48ef4f8e22e56462adc4a5917a5916bad6b260d338e64c97932ef1857610701582097e20a3b42f8f153dabb6f1c84f9d952a06f466616cd3f69f34ce8cac1f5c5fa02582098d8b1232297ab1cfb460fcbf1bdf90b43cfc5947bfd23396946e85ealae8f035820ebf344ad82eabc8c8d7b3622a711e9c01d2ced89a7ae86254c381f1071b8186004582088a017c48765235a8f6fd685d79e7cbc35b507d26bf59eaa7494a0e98f165f160558206de4fc380ba090850d9778713f578192945529ce4ea597b33f8a41a6aea046a7065820d90f5251ecdb6ad7814b28f0c85e98c5c7823423526e56e4c08a795db5104fac07582005c7769bd4355a32e6f15c677a1b7098892aa5a4e0fd6814e52ad68c5fdc943b085820165eafe617b9bfd297005e23d474518f9487d5a5d13ce2fad9d3ac4165d0cb506d6465766963654b6579496e666fa1696465766963654b6579a4010220012158204be4a1397ca8646eee0e13edc77cb0d79b23566c2e09c8c3e7e0da8eff968b6522582056a57d0c5020a7abd8eb3cda705d7edf070521961dea993eccbb0d3d9f09c6c267646f6354797065781865752e6575726f70612e65632e65756469772e7069642e316c76616c6964697479496e666fa3667369676e6564c0781e323032332d31312d32395430393a32335a32382e3735303630373230305a6976616c696446726f6dc0781e323032332d31312d32395430393a32335a32382e3735303630373230305a6a76616c6964556e746966cc0781e323032342d31312d32385430393a32353a32382e3735303630373230305a',
  h'4cfec240b978e6cb103a3fa539620ad584bb38c2fa3c57bad657a62429a2068e59f56ec3028a7665a82938015114aa0e615992ce4a09dcdf0ec65a189b87a27b',
  ],
},
}

```

3.3. WIA

The Wallet Instance Attestation (WIA) attests the authenticity and trustworthiness of a specific Wallet Instance. It may contain details about the Wallet Provider, the Wallet Solution, the Wallet Instance, and the device's security level where the Wallet Instance is installed. WIA implementation details are still under discussion within the EUDI Wallet ecosystem.

Requirements

1. The User **SHALL** have a valid WIA stored in a Wallet Instance.

To support [OAuth 2.0 Attestation-Based Client Authentication](#) in QEAA issuing and PID presentation flows the following non-normative example of the Wallet Instance Attestation JWT is used:

Wallet Attestation JWT

Header

```
{
  "typ": "wallet-attestation+jwt",
  "alg": "ES256",
  "kid": "wallet-provider-kid"
}
```

Payload

```
{
  "iss": "https://wallet-provider.example.com",
  "sub": "https://wallet-provider.example.com",
  "iat": 1541493724,
  "exp": 1516247022,
  "cnf": {
    "jwk": {
      "kty": "EC",
      "crv": "P-256",
      "x": "TCAER19Zvu3OHF4j4W4vfSVoHIP1ILlDls7vCeGemc",
      "y": "ZxjiWWbZMQGHVWVKVQ4hbSIrsVfuecCE6t4jT9F2HZQ"
    }
  }
}
```

4. Flows

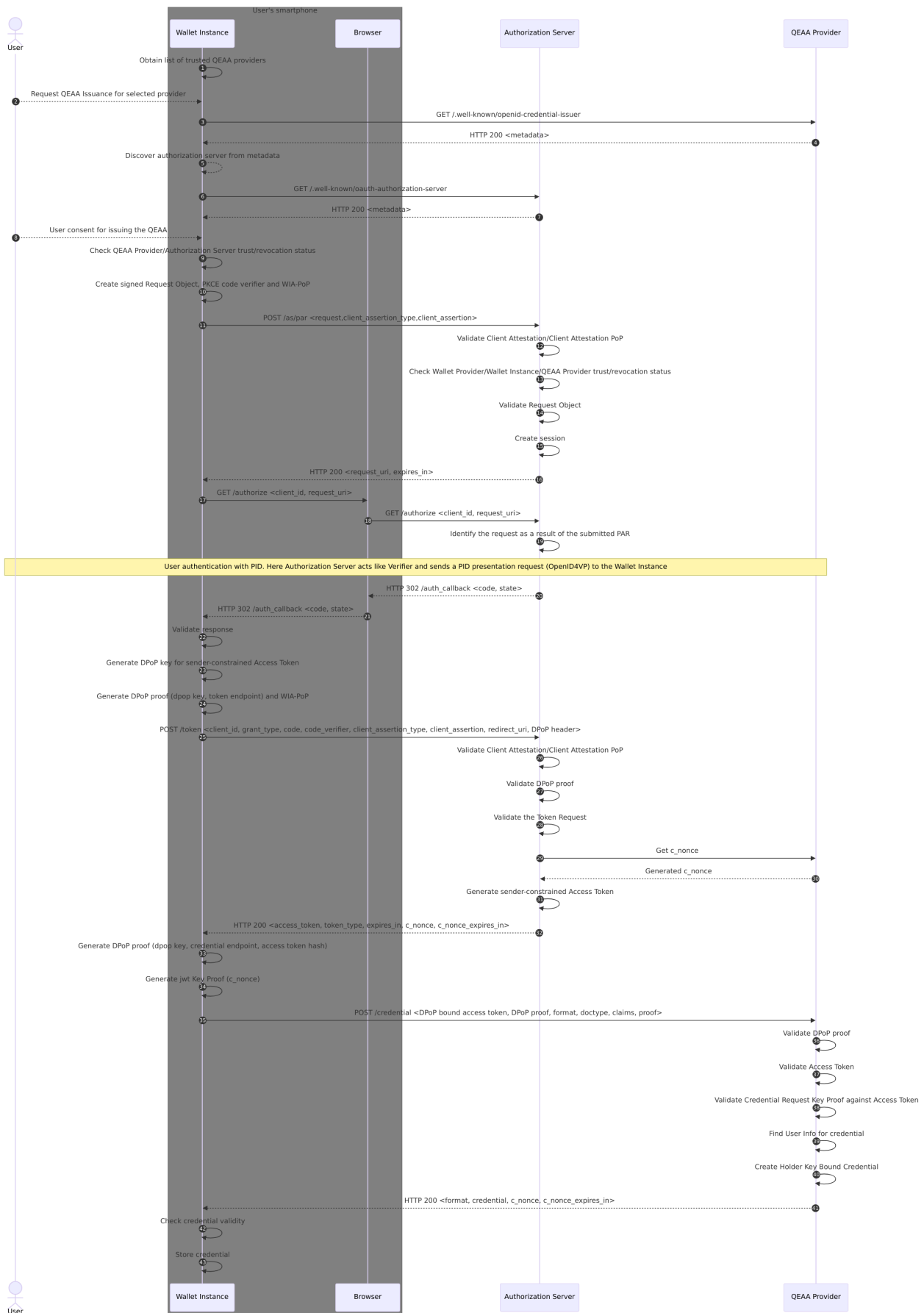
4.1. (Q)EAA Issuing

[OpenID for Verifiable Credential Issuance](#) specification defines an API that is used to issue verifiable credentials. Verifiable Credentials are very similar to identity assertions, like ID Tokens in [OpenID Connect](#), in that they allow a Credential Issuer to assert End-User claims. However, in contrast to the identity assertions, a verifiable credential follows a pre-defined schema (the Credential type) and is bound to a certain holder, e.g through cryptographic holder binding. This allows secure direct presentation of the Credential from the End-User to the Relying Party (RP), **without involvement of the Credential Issuer**.

1. The QEAA's are issued according to [OpenID4VCI](#) protocol.
2. It is based on the [Authorization Code Flow with Pushed Authorization Requests](#) and [Proof Key for Code Exchange \(PKCE\)](#) as recommended in [OpenID4VCI](#), Section 3.4.
3. It uses sender-constrained access tokens by [Demonstrating Proof of Possession \(DPoP\)](#).
4. It uses `attest_jwt_client_auth` Client Authentication method ([OAuth 2.0 Attestation-Based Client Authentication](#)) in PAR and Token Endpoint.

Issuance Flow

The flow is **Wallet Initiated** and the User receives the QEAA directly in response to the Credential Request (Immediate Flow).



Step 1 (Trusted list of QEAA providers).

- Obtain list of trusted QEAA providers. The exact methods to obtain list of trusted parties are still under discussion in [EUDI-ARF](#).

Step 2 (Issuance Request).

- User selects which credential to request.

Steps 3-7 (Discovery).

- Wallet Instance *SHALL* request QEAA Provider metadata and use `authorization_server` claim to request Authorization Server metadata. The metadata contains technical information about the Issuer and translations/display data for the offered credentials.
- It *MUST* check if the QEAA Provider/Authorization Server are trusted and not revoked. The exact methods to attest trust and validity are still under discussion in [EUDI-ARF](#).

Step 8 (Consent).

- The Wallet shows information about the QEAA Provider and the offered QEAA to the user and asks for consent.

Step 9 (Provider trust).

- The Wallet checks QEAA Provider/Authorization Server trust/revocation status.

Steps 10-11 (PAR Request).

- The Wallet Instance *MUST* create a fresh [PKCE](#) `code_verifier` and derive `code_challenge` from it to exchange the authorization code for the access token later on with `code_verifier`.
- It *MUST* use the `request` parameter (Request Object) as defined in [RFC 9126](#) Section 3.
- Request Object is a [JWT-Secured Authorization Request \(JAR\)](#) and it *MUST* be signed by the private key defined in Wallet Instance Attestation `cnf` claim.
- It *MUST* include all request parameters as claims of the Request Object except `client_assertion` and `client_assertion_type` as specified in PAR Request Object.
- PAR Endpoint is a protected endpoint. The Client Authentication is based on the model defined in [OAuth 2.0 Attestation-Based Client Authentication](#) using the WIA as Client Attestation JWT and Client Attestation PoP JWT (WIA-PoP), signed by a key defined in a WIA `cnf` claim, inside the `client_assertion` parameter and `urn:ietf:params:oauth:client-assertion-type:jwt-client-attestation` as `client_assertion_type`.
- The `authorization_details`, as defined in [RFC 9396](#), parameter *MUST* be used (as required by [OpenID4VCI](#)) and is extended to allow Wallet Instance to specify the types of the credentials required when requesting authorization for the QEAA issuance.

Steps 12-16 (PAR Response).

- The Authorization Server performs *REQUIRED* validation checks as described in PAR Validation Steps section. In summary:
 - It *MUST* authenticate the Wallet Instance using [OAuth 2.0 Attestation-Based Client Authentication](#).
 - It *MUST* request Wallet Provider/Wallet Instance/QEAA Provider revocation states.
 - It *MUST* validate the Request Object.
- The Authorization Server *MUST* create a new session related to `client_id`.
- Authorization Server *MUST* generate the `request_uri` representing a new authorization request and bind it to the `client_id` for one-time use.

Steps 17-19 (Authorization Request).

- The Wallet Instance sends an authorization request to the Authorization Endpoint.
- The Authorization Server performs *REQUIRED* validation checks as described in Authorization Request Validation Steps. In short:
 - It *MUST* identify the request as a result of the submitted PAR.
 - It *MUST* prevent replay of the authorization request.

User Authentication.

- Authorization Server *MUST* authenticate user by performing Dynamic Credential Request ([OpenID4VCI](#)), to obtain PID Verifiable Credential from Wallet Instance using [OpenID4VP](#) protocol.
- The Wallet Instance *MUST* have a valid PID obtained prior to starting a transaction with the Authorization Server.

Steps 20-22 (Authorization Response).

- The Authorization Server sends `authorization_code` and `state` parameters to the Wallet Instance `redirect_uri`.
- The Wallet Instance performs *REQUIRED* validation checks as described in Authorization Response Validation Steps.

Steps 23-24 (DPoP Proof for Token Endpoint).

- The Wallet Instance *MUST* create a new key pair for the DPoP and a fresh DPoP Proof JWT following the instruction provided in Section 4 of [RFC 9449](#) for the Token Endpoint.

- The DPoP Proof JWT **MUST** be signed by Wallet Instance using the newly created private key for DPoP, this provides a way to bind the Access Token to a certain sender (Wallet Instance) and mitigates the misuse of leaked or stolen Access Tokens at the QEAA Provider Credential Endpoint as the attacker needs to present a valid DPoP Proof JWT.
-

Step 25 (Token Request with DPoP proof).

- The Wallet Instance sends a token request to the Token Endpoint using the authorization `code`, `code_verifier`, DPoP Proof JWT and `attest_jwt_client_auth` parameters (`client_assertion_type` and `client_assertion`).
 - The Client Authentication is based on the model defined in [OAuth 2.0 Attestation-Based Client Authentication](#) using the WIA as Client Attestation JWT and Client Attestation PoP JWT (WIA-PoP), signed by a key defined in a WIA `cnf` claim, inside the `client_assertion` parameter and `urn:ietf:params:oauth:client-assertion-type:jwt-client-attestation` as `client_assertion_type`.
-

Steps 26-28 (Token Request Validation).

- The Authorization Server performs **REQUIRED** validation checks as described in Token Request Validation Steps. In summary:
 - It **MUST** authenticate the Wallet Instance using [OAuth 2.0 Attestation-Based Client Authentication](#).
 - It **MUST** validate the DPoP proof for Token Endpoint.
 - It **MUST** validate the Token Request.
-

Step 29-32 (Token Response).

- Due to the Requirement 11 the `c_nonce` has to be requested from QEAA Issuer Nonce Endpoint or alternately not be returned by Token Endpoint and instead acquired from Credential endpoint error response as described in [OpenID4VCI](#).
 - If the validation is successful, it **MUST** issue an Access Token bound to the DPoP key and a `c_nonce`, that is used to create a proof of possession of key material when requesting a Credential Endpoint.
-

Step 33-34 (Proofs for Credential Endpoint).

- The Wallet Instance **MUST** create a new DPoP proof for the Credential Endpoint and Access Token Hash.
 - It **MUST** create `jwt Key Proof` for key, that will be bound to issued credential, as described in Credential Request.
-

Step 35 (Credential request).

- The Wallet Instance requests a QEAA issuance.
 - The request **MUST** contain the sender-constrained Access Token in the Authorization header as described in [RFC 9449](#).
 - The request **MUST** contain DPoP Proof JWT with `ath` claim in the DPoP header as described in [RFC 9449](#)
 - The request **MUST** contain `jwt Key Proof` with `c_nonce` value in `nonce` claim as `proof` parameter and the `format`, `doctype`, `claims` parameters as described in Credential Request parameters.
-

Steps 36-38 (Credential request validation).

- The QEAA Provider performs **REQUIRED** validation checks as described in Credential Request Validation Steps. In summary:
 - It **MUST** validate the DPoP proof for Credential Endpoint/Access Token Hash.
 - It **MUST** validate the sender-constrained Access Token.
 - It **MUST** validate the `jwt Key Proof`.
-

Steps 39-40 (Credential issuance).

- The QEAA Provider **SHALL** find required data for credential using Access Token sub claim.
 - It **MUST** issue credential in the format requested by client in the `doctype` and `format` parameters.
 - It **MUST** use `jwt Key Proof` type to bind the Issued Credential to the Identifier (Public Key) of the End-User Possessing that Credential (Wallet Instance).
-

Step 41-42 (Credential Response).

- The Wallet Instance performs **REQUIRED** validation checks as described in Credential Response Validation Steps.
-

Step 43 (Credential Storage).

- Wallet Instance stores the credential.
-

Updating Issued Credentials

To update issued credential Wallet Instance **MUST** use the `c_nonce` returned by Credential Endpoint in step 37 and perform steps 29-39. It **MUST** do this before `c_nonce_expires_in` and Access Token `exp` claim or perform the credential issuance from beginning.

4.2. (Q)EAA Presentation

This section describes how a Relying Party (RP), acting as a Verifier, can request a Wallet Instance for presentation of the QEAA verifiable credentials.

Verifier is an entity that requests, receives, and validates Verifiable Presentations. During presentation of Credentials, Verifier acts as an OAuth 2.0 Client towards the Wallet that is acting as an OAuth 2.0 Authorization Server. The Verifier is a specific case of OAuth 2.0 Client, just like Relying Party in [OpenID Core](#).

[OpenID4VP](#) specification defines a mechanism on top of OAuth 2.0 to request and present Verifiable Credentials as Verifiable Presentations.

As the primary extension, [OpenID4VP](#) introduces the VP Token as a container to enable End-Users to present Verifiable Presentations to Verifiers using the Wallet. A VP Token contains one or more Verifiable Presentations in the same or different Credential formats.

Current implementation scope does not require Wallet Instance to be authenticated to Relying Party (Verifier).

Considered use cases

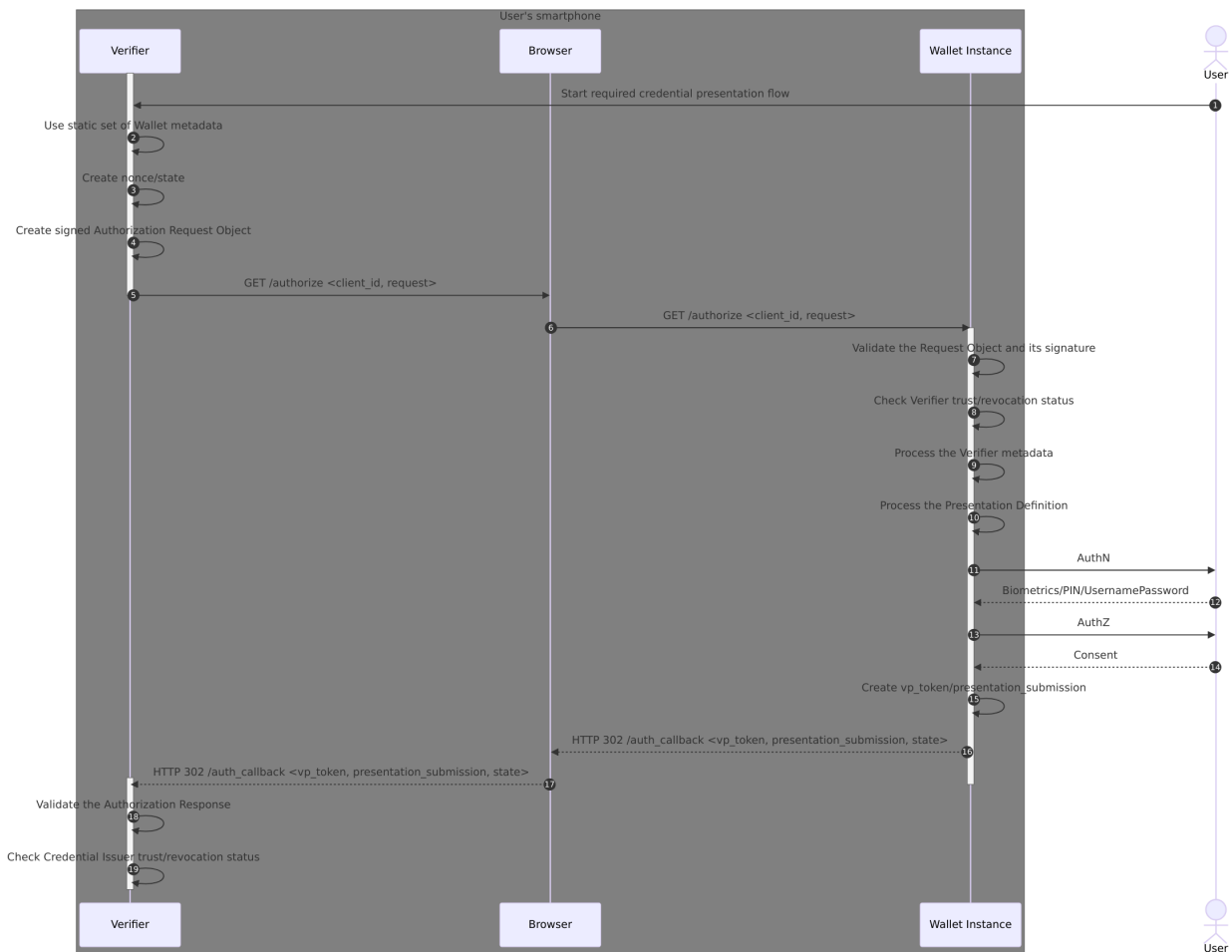
1. Native iOS/Android mDL consumer application requesting mDL presentation in same-device flow.

Wallet Metadata

1. Wallet Provider SHALL provide domain-bound iOS Universal Link/Android App link for Wallet Invocation in `https://<wallet-provider-domain>/.well-known/apple-app-site-association` and `https://<wallet-provider-domain>/.well-known/assetlinks.json` accordingly.
2. The Verifier SHALL use the following static set of Wallet metadata:

```
{
  "authorization_endpoint": "https://<wallet-provider-domain>/authorize",
  "client_id_schemes_supported": [
    "x509_san_dns"
  ],
  "presentation_definition_uri_supported": false,
  "request_object_signing_alg_values_supported": [
    "ES256"
  ],
  "response_types_supported": [
    "vp_token"
  ],
  "vp_formats_supported": {
    "mso_mdoc": {
      "alg_values_supported": [
        "ES256"
      ]
    }
  }
}
```

Presentation Flow



Step 1 (Presentation flow start).

- User (or the Verifier application) starts the required credential presentation flow.

Steps 2-6 (Authorization Request).

- The Verifier **MUST** use the static set of [Wallet metadata](#) to determine Credential formats, proof types and algorithms supported by the Wallet to be used in a protocol exchange.
- It **MUST** create a fresh `nonce` with sufficient entropy to securely bind the Verifiable Presentation(s) provided by the Wallet to the particular transaction.
- It **MUST** create a `state` to link Authorization Request to Authorization Response.
- It **MUST** create a [JWT-Secured Authorization Request \(JAR\)](#) with claims defined in [Authorization Request Object](#).

Steps 7-8 (Authorization Request validation).

- The Wallet performs the **REQUIRED** [Authorization Request validation steps](#).
- It **MUST** check if the Verifier is trusted and not revoked. The exact methods to attest trust and validity are still under discussion in [EUDI-ARF](#).

Step 9 (Verifier Metadata validation).

- The Wallet performs the **REQUIRED** [Verifier Metadata validation steps](#).

Step 10 (Presentation Definition validation).

- The Wallet performs the **REQUIRED** [Presentation Definition validation steps](#), that describes the requirements of the Credential(s) that the Verifier is requesting to be presented.

Steps 11-12 (User authentication).

- The Wallet **MUST** authenticate the User with Biometrics/PIN/UsernamePassword.

Steps 13-14 (User consent).

- The Wallet **MUST** ask the User consent for requested credentials and claims.

Step 15 (vp_token/presentation_submission creation).

- The Wallet **MUST** bind Verifiable Presentation to the authorization request `client_id` and `nonce` values by appending [DeviceSigned](#) structure with [OpenID4VPHandover](#) to mDL returned in [QEAA Issuance flow](#) and set the resulting CBOR encoded mDL directly as `vp_token` value.
- It **MUST** create [Presentation Submission](#) object in response to [Presentation Definition](#).

Steps 16-17 (Authorization Response).

- The Wallet **MUST** redirect the Authorization Response to the URI provided in Authorization Request `redirect_uri` parameter.

Steps 18-19 (Authorization Response validation).

- The Verifier performs the **REQUIRED** [Authorization Response validation steps](#)
- It **MUST** check if the Credential Issuer is trusted and not revoked. The exact methods to attest trust and validity are still under discussion in [EUDI-ARF](#).

Authorization Request

Authorization Request Object

Header

Claim	Description	Reference
<code>alg</code>	A digital signature algorithm identifier such as per IANA JSON Web Signature and Encryption Algorithms . It MUST NOT be set to <code>none</code> or any symmetric algorithm (MAC) identifier.	RFC 7515 Section 4.1.1
<code>typ</code>	It MUST be set to <code>oauth-authz-req+jwt</code> .	RFC 9101
<code>x5c</code>	Contains the X.509 public key certificate or certificate chain corresponding to the key used to digitally sign the JWS. The certificate or certificate chain is represented as a JSON array of certificate value strings. Each string in the array is a base64-encoded DER PKIX certificate value. The certificate containing the public key corresponding to the key used to digitally sign the JWS MUST be the first certificate. This MAY be followed by additional certificates, with each subsequent certificate being the one used to certify the previous one.	RFC 7515

Payload

Claim	Description	Reference
<code>response_type</code>	It MUST be set to <code>vp_token</code> . Indicates that a successful response MUST include the <code>vp_token</code> parameter. The default Response Mode for this Response Type is <code>fragment</code> , i.e., the Authorization Response parameters are encoded in the fragment added to the <code>redirect_uri</code> when redirecting back to the Verifier.	OpenID4VP, RFC 6749
<code>client_id</code>	It MUST be set to DNS name and match a <code>dNSName Subject Alternative Name (SAN)</code> [RFC5280] entry in the leaf certificate passed with the request header <code>x5c</code> claim. It is also used to bind the Verifiable Presentation to the intended audience.	OpenID4VP
<code>redirect_uri</code>	It MUST be set to callback URI the Authorization Response should be redirected.	OpenID4VP
<code>client_id_scheme</code>	It MUST be set to <code>x509_san_dns</code> .	OpenID4VP
<code>client_metadata</code>	A JSON object containing the Verifier metadata values. It MUST be UTF-8 encoded.	OpenID4VP
<code>presentation_definition</code>	A JSON object containing a Presentation Definition for requested credential.	OpenID4VP, DIF, PresentationExchange

nonce	It is used to bind the Verifiable Presentation to the particular transaction.	OpenID4VP, OpenID.Core
state	It is used to uniquely identify Authorization Request.	OpenID4VP, RFC 6749

Client Metadata

- The Verifier SHALL add a `vp_formats` element to its metadata to let the Wallet know what Verifiable Presentation formats and algorithms it supports, as described in [OpenID4VP Section 9](#). Additional requirements for Verifier Metadata MAY be added in future versions of this specification.
- The Wallet SHALL display `client_name`, `client_uri` and `logo_uri` in consent view.

A non-normative example of the Verifier Metadata:

```
{
  "client_name": "Verifier name",
  "client_uri": "http://verifier.example.com/info",
  "logo_uri": "http://verifier.example.com/logo.png",
  "vp_formats": {
    "mso_mdoc": {
      "alg": [
        "ES256"
      ]
    }
  }
}
```

Presentation Definition Object

Presentation Definition is an object, that defines what proofs Verifier requires to decide how or whether to interact with a Holder. Presentation Definitions are composed of inputs, which describe the forms and details of the proofs they require, and optional sets of selection rules, to allow Holders flexibility in cases where many different types of proofs may satisfy an input requirement.

Property	Description	Reference
id	It MUST be set to a UUIDv4 value to uniquely identify presentation request.	OpenID4VP, DIF. PresentationExchange
name	If present, its value MUST be a string that describes distinctive designation of the Presentation Definition.	OpenID4VP, DIF. PresentationExchange
purpose	It MUST be a string that describes the purpose for which the Presentation Definition's inputs are being used for.	OpenID4VP, DIF. PresentationExchange
input_descriptors	A JSON array containing a list of Input Descriptor Objects describing what type of input data/claim, or sub-fields thereof, are required for submission to the Verifier.	OpenID4VP, DIF. PresentationExchange

Input Descriptor Object

Property	Description	Reference
id	It MUST be set to <code>org.iso.18013.5.1.mDL</code> .	OpenID4VP, DIF. PresentationExchange
format	It MUST be set to a JSON object with single property <code>mso_mdoc</code> , where <code>mso_mdoc</code> is a JSON object with single property <code>alg</code> , that contains requested array of supported JSON Web Algorithms .	OpenID4VP, DIF. PresentationExchange
constraints	A Constraints JSON object that describes	OpenID4VP, DIF. PresentationExchange

Constraints Object

Property	Description	Reference
fields	A JSON array containing a list of Fields Objects	OpenID4VP, DIF. PresentationExchange
limit_disclosure	If present, enables selective disclosure by instructing the Wallet to submit only the data specified in the fields array. It MUST be set to <code>required</code> .	OpenID4VP, DIF. PresentationExchange

Fields Object

Property	Description	Reference
----------	-------------	-----------

path	A JSON array containing a one or more JSONPath expressions that defines static path to a certain claim in requested credential. Only mandatory MDL data elements from <code>org.iso.18013.5.1</code> namespace can be requested.	OpenID4VP, DIF. PresentationExchange, I-D. ietf-jsonpath-base
intent_to_retain	A boolean property introduced to meet requirements of ISO/IEC 18013-1:2018 to allow a Verifier to indicate it will retain the submitted value for the specific field.	OpenID4VP, DIF. PresentationExchange

A non-normative example of the Presentation Definition Object:

```
{
  "id": "80200d58-7198-11ee-b962-0242ac120002",
  "input_descriptors": [
    {
      "id": "org.iso.18013.5.1.mDL",
      "format": {
        "mso_mdoc": {
          "alg": [
            "ES256"
          ]
        }
      },
      "constraints": {
        "limit_disclosure": "required",
        "fields": [
          {
            "path": [
              "${'org.iso.18013.5.1'}['family_name']"
            ],
            "intent_to_retain": false
          },
          {
            "path": [
              "${'org.iso.18013.5.1'}['given_name']"
            ],
            "intent_to_retain": false
          },
          {
            "path": [
              "${'org.iso.18013.5.1'}['birth_date']"
            ],
            "intent_to_retain": false
          },
          {
            "path": [
              "${'org.iso.18013.5.1'}['document_number']"
            ],
            "intent_to_retain": true
          }
        ]
      }
    }
  ]
}
```

Validation Steps

- The Wallet **MUST** validate the [Authorization Request Object](#).
- It **MUST** validate the [Presentation Definition](#).

Verifier Metadata validation steps

- The Wallet **MUST** validate that it supports the requested VP format and algorithms.
- It **MUST** validate that the `client_name`, `client_uri` and `logo_uri` are present and in display them in consent view.

Presentation Definition validation steps

- The Wallet **MUST** validate the [Presentation Definition Object](#).
- It **MUST** ignore any format property inside a Presentation Definition object if that format was not included in the `vp_formats` property of the metadata.
- It **MUST** select candidate Verifiable Credential(s) using the evaluation process described in Section 8 of [DIF.PresentationExchange](#).
- It **MUST** display `purpose` claim and requested [Fields Object](#) with corresponding translations acquired from [Credentials Supported Display Object](#) in consent view.

Authorization Response

Parameter	Description	Reference
vp_token	JSON String that MUST contain a single CBOR encoded mDL.	OpenID4VP
presentation_submission	JSON object that contains mappings between the requested Verifiable Credentials and where to find them within the returned VP Token.	OpenID4VP , DIF. PresentationExchange
state	It MUST be set to the authorization request state value.	OpenID4VP

Presentation Submission Object

Property	Description	Reference
definition_id	It MUST be set to Presentation Definition id this Presentation Submission is intended to.	OpenID4VP , DIF. PresentationExchange
id	It MUST be set to a UUIDv4 value to uniquely identify presentation submission.	OpenID4VP , DIF. PresentationExchange
descriptor_map	JSON array that contains Input Descriptor Mapping Objects . As single CBOR encoded mDL is returned in vp_token, it SHALL contain single mapping object with path mapping \$ to denote that CBOR encoded mDL is set directly in the vp_token element.	OpenID4VP , DIF. PresentationExchange

Input Descriptor Mapping Object

Property	Description	Reference
id	It MUST be set to Input Descriptor id this Descriptor Object is intended to. It MUST be set to org.iso.18013.5.1.mDL.	OpenID4VP , DIF. PresentationExchange
format	It MUST be set to mso_mdoc.	OpenID4VP , DIF. PresentationExchange
path	It MUST be set to \$.	OpenID4VP , DIF. PresentationExchange
path_nested	It MUST NOT be used. When mDL is expressed in CBOR encoding the path_nested parameter cannot be used to point to the location of the requested claims. The user claims will always be included in the issuerSigned element of the mDL document.	OpenID4VP , DIF. PresentationExchange

A non-normative example of the Presentation Submission Object:

```
{
  "definition_id": "80200d58-7198-11ee-b962-0242ac120002",
  "id": "af53b57c-71a0-11ee-b962-0242ac120002",
  "descriptor_map": [
    {
      "id": "org.iso.18013.5.1.mDL",
      "format": "mso_mdoc",
      "path": "$"
    }
  ]
}
```

Error Response

- The Wallet **MUST** follow the error response rules as defined in [OpenID4VP](#) Section 6.4

Validation Steps

- Verifier **MUST** validate vp_token as described in [OpenID4VP](#) Section 6.5.
- Verifier **MUST** validate the signature mDL as described in [ISO/IEC 18013-1:2018](#) Section 9.3.
- Verifier **MUST** validate that client_id and nonce values are bound to the returned credential by reconstructing the [DeviceAuthentication](#) structure with [OpenID4VPHandover](#) and validating the signature provided in [DeviceSignature](#) structure.

5. Endpoints

5.1. PAR Endpoint

1. The Pushed Authorization Request Endpoint (PAR) Endpoint is an HTTP API at the Authorization Server and MUST accept HTTP POST request with parameters in the HTTP request message body using the `application/x-www-form-urlencoded` format.
2. It MUST use the `https` scheme.
3. Use of Pushed Authorization Requests (PAR) is RECOMMENDED by [OPENID4VCI](#).
4. It MUST use `attest_jwt_client_auth` Client Authentication method as defined in [OAuth 2.0 Attestation-Based Client Authentication](#).
5. It MUST use the `request` parameter (Request Object) as defined in [RFC 9126](#) Section 3.
6. It MUST use the `authorization_details` parameter, as defined in [RFC 9396](#) and as REQUIRED by [OpenID4VCI](#).
7. The Authorization Server MUST be able to uniquely identify the Credential Issuer based on the `locations` claim value in `authorization_details` object as suggested by [OpenID4VCI](#).

PAR Request

PAR Request Parameters

Parameter	Description	Reference
<code>request</code>	It MUST be a signed JWT. It MUST be signed by the private key defined in WIA <code>cnf</code> claim. All request parameters MUST appear as claims of the JWT representing the authorization request except <code>client_assertion</code> and <code>client_assertion_type</code> .	RFC 9126 Section 3
<code>client_assertion_type</code>	It MUST be set to <code>urn:ietf:params:oauth:client-assertion-type:jwt-client-attestation</code> .	OAuth 2.0 Attestation-Based Client Authentication
<code>client_assertion</code>	It MUST contain two JWTs separated by a <code>~</code> character. It MUST NOT contain more or less than precisely two JWTs separated by the <code>~</code> character. The first JWT MUST be the WIA as Client Attestation JWT the second JWT MUST be the Client Attestation PoP JWT (WIA-PoP) that MUST be signed by the private key defined in WIA <code>cnf</code> claim.	OAuth 2.0 Attestation-Based Client Authentication , Sections 4.1.1, 4.1.2

*Client Attestation PoP JWT (WIA-PoP)

Header

Claim	Description	Reference
<code>alg</code>	A digital signature algorithm identifier such as per IANA JSON Web Signature and Encryption Algorithms. It MUST NOT be set to <code>none</code> or any symmetric algorithm (MAC) identifier.	RFC 7515 Section 4.1.1
<code>kid</code>	It MUST reference the thumbprint value of the <code>cnf</code> claim in WIA .	RFC 7638 Section 3, attestation-based-client-auth
<code>typ</code>	It MUST be set to <code>wallet-attestation-pop+jwt</code> . The <code>typ</code> claims in Client Attestation and Client Attestation PoP are still under discussion.	attestation-based-client-auth

Payload

Claim	Description	Reference
<code>iss</code>	It MUST be set to <code>sub</code> claim value of the WIA .	attestation-based-client-auth
<code>aud</code>	It MUST be set to the URL of Authorization Server PAR Endpoint.	attestation-based-client-auth
<code>exp</code>	It MUST be UNIX Timestamp with the expiry time of the JWT.	attestation-based-client-auth
<code>jti</code>	Claim that provides a unique identifier for the token. The Authorization Server MAY ensure that JWTs are not replayed by maintaining the set of used <code>jti</code> values for the length of time for which the JWT would be considered valid based on the applicable <code>exp</code> instant.	attestation-based-client-auth

PAR Request Object

Header

Claim	Description	Reference
alg	A digital signature algorithm identifier such as per IANA JSON Web Signature and Encryption Algorithms . It MUST NOT be set to <code>none</code> or any symmetric algorithm (MAC) identifier.	RFC 7515 Section 4.1.1
kid	It MUST reference the thumbprint value of the <code>cnf</code> claim in WIA .	RFC 7638 Section 3

Payload

Claim	Description	Reference
iss	It MUST be set to <code>sub</code> claim value of the WIA .	RFC 9126 , RFC 9101 , RFC 7519
aud	It MUST be set to the URL of Authorization Server PAR Endpoint.	RFC 9101 , RFC 7519
exp	It MUST be UNIX Timestamp with the expiry time of the JWT.	RFC 7519
iat	It MUST be UNIX Timestamp with the time of JWT issuance.	RFC 7519
jti	Claim that provides a unique identifier for the token. The Authorization Server MAY ensure that JWTs are not replayed by maintaining the set of used <code>jti</code> values for the length of time for which the JWT would be considered valid based on the applicable <code>exp</code> instant.	RFC 9126 , RFC 7519
state	Unique session identifier at the client side. This value will be returned to the client in the response, at the end of the authentication. It MUST be a random string composed by alphanumeric characters and with a minimum length of 32 digits.	OpenID.Core Section 3.1.2.1, [RFC6749]
code_challenge	A challenge derived from the <code>code_verifier</code> that is sent in the authorization request.	RFC 7636 Section 4.2
code_challenge_method	A method that was used to derive <code>code_challenge</code> . It MUST be set as <code>S256</code> .	RFC 7636 Section 4.3
client_id	It MUST be set to <code>sub</code> claim value of the WIA .	RFC 9126 , RFC 6749 , RFC 7638
authorization_details	A JSON array containing a list of Authorization Details Object used to convey details about the credentials the wallet wants to obtain.	RFC 9396 , OpenID4VCI
response_type	It MUST be set to <code>code</code> .	RFC 6749
redirect_uri	Redirection URI to which the response is intended to be sent. It MUST be an Universal (iOS) or App Link (Android) registered with the local operating system.	RFC 6749

Authorization Details Object

Claim	Description	Reference
type	It MUST be set to <code>openid_credential</code>	RFC 9396 , OpenID4VCI
format	It MUST be set to <code>mso_doc</code>	RFC 9396 , OpenID4VCI
locations	If the Credential Issuer metadata contains an <code>authorization_server</code> parameter the <code>locations</code> field MUST be set to the Credential Issuer Identifier value. The value MUST be used as <code>aud</code> claim in Access Token returned by Token Endpoint.	RFC 9396 , OpenID4VCI
doctype	JSON string identifying the credential type. It MUST be set to <code>org.iso.18013.5.1.mDL</code> as defined in ISO/IEC 18013-5:2021	RFC 9396 , OpenID4VCI , ISO/IEC 18013-5:2021
claims	A JSON object containing a list of name/value pairs, where the name is a certain namespace as defined in ISO/IEC 18013-5:2021 (or any profile of it) and the value is a JSON object. It MUST defined as in Authorization Details Claims Object .	RFC 9396 , OpenID4VCI , ISO/IEC 18013-5:2021

Authorization Details Claims Object

Claim	Description	Reference
-------	-------------	-----------

org.iso.18013.5.1	A JSON object containing a list of name/value pairs, where the name is a claim name value that is defined in the respective namespace and is offered in the Credential.	OpenID4VCI
-------------------	---	------------

Validation Steps

1. Authorization Server **MUST** authenticate the Wallet Instance based on the `attest_jwt_client_auth` Client Authentication method ([OAuth 2.0 Attestation-Based Client Authentication](#)).
2. All request parameters **MUST** appear as claims of the JWT representing the authorization request except `client_assertion` and `client_assertion_type` as required in [RFC 9126](#) Section 3.
3. It **MUST** validate the signature of the Request Object using the algorithm specified in the `alg` header parameter ([RFC 9126](#), [RFC 9101](#)) and the public key that can be retrieved from the `WIA_cnf` claim using the `kid` header claim of the Request Object.
4. It **MUST** check that the used algorithm for signing the request in the `alg` header claim is among the appropriate cryptographic algorithms defined in [RFC 7515](#).
5. It **MUST** check that the `iss` claim in the Request Object matches the `client_id` claim in the Request Object ([RFC 9126](#), [RFC 9101](#)).
6. It **MUST** check that the `iss` and `client_id` claims are equal to the `sub` claim value in the Client Attestation JWT ([WIA](#)).
7. It **MUST** check that the `aud` claim in the Request Object is equal to the Authorization Server PAR Endpoint URI.
8. It **MUST** reject the PAR request, if Request Object contains the `request_uri` claim ([RFC 9126](#)).
9. It **MUST** check that the Request Object is not expired by checking the `exp` claim ([RFC 9126](#)).
10. It **MUST** check that the Request Object was issued at a time acceptable by the QEAA Provider by checking the `iat` claim ([RFC 9126](#)).
11. It **MUST** validate the `authorization_details` object.
12. It **MUST** check that the `jti` claim in the Request Object has not been used before by the Wallet Instance identified by the `client_id`. This allows the QEAA Provider to mitigate replay attacks ([RFC 7519](#)).
13. It **MUST** check the revocation status of the Wallet Provider, Wallet Instance and QEAA Provider.
14. It **MUST** validate that QEAA Provider is allowed to issue the credential type specified in `claims.doctype` and `claims`.

PAR Response

1. QEAA Provider **MUST** issue the `request_uri` for one-time use and bind it to the client identifier `client_id`.
2. It **MUST** send 201 HTTP status code on successful PAR Response.
3. PAR Response **MUST** be sent using `application/json` content type and contain following claims:

Claim	Description	Reference
<code>request_uri</code>	The request URI corresponding to the authorization request posted. This URI MUST be a single-use reference to the respective authorization request. It MUST contain some part generated using a cryptographically strong pseudorandom algorithm such that it is computationally infeasible to predict or guess a valid value. The value format MUST be <code>urn:ietf:params:oauth:request_uri:<reference-value></code> with <code><reference-value></code> as the random part of the URI that references the respective authorization request data. The <code>request_uri</code> value MUST be bound to the client that posted the authorization request.	RFC 9126
<code>expires_in</code>	JSON number that represents the lifetime of the request URI in seconds as a positive integer. It SHOULD NOT exceed 60 seconds.	RFC 9126

5.2. Authorization Endpoint

1. The Authorization Endpoint is an HTTP API at the Authorization Server and used in the same manner as defined in [RFC 6749](#) to interact with the QEAA Provider and obtain an authorization grant.
2. Client **MUST** use Pushed Authorization Requests (PAR) [RFC 9126](#) to send the Authorization Request.

Authorization Request

Parameter	Description	Reference
<code>client_id</code>	It MUST be set to <code>sub</code> claim value of the Wallet Instance Attestation .	RFC 9126
<code>request_uri</code>	It MUST be set to the same value as obtained by PAR Response.	RFC 9126

Validation Steps

1. It **MUST** treat `request_uri` values as one-time use and **MUST** reject an expired request.
2. It **MUST** identify the request as a result of the submitted PAR.
3. It **MUST** reject all the Authorization Requests that do not contain the `request_uri` parameter as the PAR is the only way to pass the Authorization Request from the Wallet Instance.
4. The Authorization Server **MUST** verify the identity of the User that owns the credential. It **MUST** initiate user authentication as described in [PID Authentication Flow](#).

Authorization response

Parameter	Description	Reference
code	Unique Authorization Code that the Wallet Instance submits to the Token Endpoint.	RFC 6749
state	It MUST be set to state value, that was used in Request Object	RFC 6749

Validation Steps

1. It MUST check the returned state value is equal to the value sent by Wallet Instance in the Request Object.

5.3. Token Endpoint

1. The Token Endpoint is an HTTP API at the Authorization Server and is used by the Wallet Instance to obtain an Access Token by presenting its authorization grant, as defined in [RFC 6749](#)
2. It MUST accept HTTP POST request with parameters in the HTTP request message body using the application/x-www-form-urlencoded format.
3. It MUST use the https scheme.
4. It MUST issue sender-constrained Access Tokens.
5. It MUST use attest_jwt_client_auth Client Authentication method as defined in [RFC 7523](#),[RFC 7521](#).

Token Request

Parameter	Description	Reference
client_id	It MUST be set to sub claim value of the WIA.	RFC 6749
grant_type	It MUST be set to authorization_code.	RFC 6749 , RFC 7521
code	It MUST be set to Authorization code returned in the Authentication Response.	RFC 6749 , RFC 7521
code_verifier	Verification code of the code_challenge sent in PAR Request.	RFC 9449
client_assertion_type	It MUST be set to urn:ietf:params:oauth:client-assertion-type:jwt-client-attestation.	OAuth 2.0 Attestation-Based Client Authentication
client_assertion	It MUST contain two JWTs separated by a ~ character. It MUST NOT contain more or less than precisely two JWTs separated by the ~ character. The first JWT MUST be the WIA as Client Attestation JWT the second JWT MUST be the Client Attestation PoP JWT (WIA-PoP) that MUST be signed by the private key defined in WIA cnf claim.	OAuth 2.0 Attestation-Based Client Authentication , Sections 4.1.1, 4.1.2
redirect_uri	It MUST be set as in the PAR Request Object.	RFC 6749 , RFC 7521

*Client Attestation PoP JWT (WIA-PoP)

Header

Claim	Description	Reference
alg	A digital signature algorithm identifier such as per IANA JSON Web Signature and Encryption Algorithms. It MUST NOT be set to none or any symmetric algorithm (MAC) identifier.	RFC 7515 Section 4.1.1
kid	It MUST reference the thumbprint value of the cnf claim in WIA.	RFC 7638 Section 3, attestation-based-client-auth
typ	It MUST be set to wallet-attestation-pop+jwt. The typ claims in Client Attestation and Client Attestation PoP are still under discussion.	attestation-based-client-auth

Payload

Claim	Description	Reference
iss	It MUST be set to sub claim value of the WIA.	attestation-based-client-auth

aud	It MUST be set to the URL of Authorization Server Token Endpoint.	attestation-based-client-auth
exp	It MUST be UNIX Timestamp with the expiry time of the JWT.	attestation-based-client-auth
jti	Claim that provides a unique identifier for the token. The Authorization Server MAY ensure that JWTs are not replayed by maintaining the set of used jti values for the length of time for which the JWT would be considered valid based on the applicable exp instant.	attestation-based-client-auth

DPoP Proof JWT

1. A DPoP Proof JWT MUST be included in an HTTP request using the DPoP header parameter containing a DPoP JWS.

Header

Claim	Description	Reference
typ	It MUST be set to dpop+jwt.	RFC 7515
alg	A digital signature algorithm identifier such as per IANA "JSON Web Signature and Encryption Algorithms" registry. It MUST NOT be set to none or with a symmetric algorithm (MAC) identifier.	RFC 7515
jwk	Public key generated by the Wallet Instance, in JSON Web Key (JWK) RFC 7517 format that the Access Token shall be bound to, as defined in Section 4.1.3 of RFC 7515. It MUST NOT contain a private key.	RFC 7515, RFC 7517

Payload

Claim	Description	Reference
jti	It MUST be set to a UUIDv4 value to uniquely identify the DPoP proof JWT.	RFC 4122, RFC 9449
htm	The value of the HTTP method of the request to which the JWT is attached. It MUST be set to POST.	RFC 9449
htu	The HTTP target URI, without query and fragment parts, of the request to which the JWT is attached. It MUST be set to Token Endpoint URI.	RFC 9449
iat	It MUST be set to the time of the JWT issuance as a UNIX Timestamp.	RFC 9449

Validation Steps

1. It MUST use attest_jwt_client_auth Client Authentication method as defined in OAuth 2.0 Attestation-Based Client Authentication.
2. It MUST ensure that the Authorization code is issued to the authenticated Wallet Instance (RFC 6749).
3. It MUST ensure the Authorization code is valid and has not been previously used (RFC 6749).
4. It MUST ensure the redirect_uri is equals to the value that was initially included in the Request Object (OpenID.Core, Section 3.1.3.1).
5. It MUST validate the DPoP Proof JWT following the steps in Section 4.3 of (RFC 9449). If the DPoP proof is invalid, the Token endpoint returns an error response, according to Section 5.2 of RFC 6749 with invalid_dpop_proof as the value of the error parameter.
6. It MUST validate the Token Request as specified in Token Request table.

Token response

1. It MUST send 200 HTTP status code on successful Token Response.
2. Token Response MUST be sent using application/json content type and contain following claims:

Claim	Description	Reference
access_token	The sender-constrained (DPoP) Access Token. Allows accessing the QEAA Provider Credential Endpoint to obtain the credential. It MUST be a signed JWT and contain claims as defined in Access Token table.	RFC 6749
token_type	Type of Access Token returned. It MUST be set to DPoP.	RFC 6749, RFC 9449
c_nonce	A nonce value to be used for proof of possession of key material in a subsequent request to the Credential Endpoint. When received, the Wallet MUST use this nonce value for its subsequent credential requests until the Credential Issuer provides a fresh nonce.	OpenID4VCI
c_nonce_expires_in	Expiry time of the c_nonce in seconds.	OpenID4VCI

Access Token

1. A sender-constrained (DPoP) Access Token **MUST** be generated by the Token Endpoint as a result of a successful token request.
2. It **MUST** be encoded in JWT format, according to [RFC 7519](#).
3. It **MUST** be bound to the public key, that is provided by the DPoP `proof` as defined in Section 6 of [RFC 9449](#).
4. It **MUST** have at least the following mandatory claims:

Claim	Description	Reference
<code>iss</code>	It MUST be an HTTPS URL that uniquely identifies the Authorization Server. The QEAA Provider MUST verify that this value matches the trusted Authorization Server.	RFC 9068
<code>sub</code>	It identifies the subject of the JWT. It MUST be set to the <code>unique_id</code> claim value of the <code>PID</code> .	RFC 9068
<code>client_id</code>	It MUST be set to <code>sub</code> claim value of the <code>WIA</code> .	RFC 9068
<code>aud</code>	It MUST be set to the Credential Issuer Identifier.	RFC 9068
<code>iat</code>	It MUST be set to the time of the JWT issuance as a UNIX Timestamp	RFC 9068
<code>exp</code>	It MUST be set to the expiry time of the JWT as a UNIX Timestamp	RFC 9068
<code>cnf</code>	JSON object. It MUST contain single claim <code>jkt</code> . It uses JWK SHA-256 Thumbprint Confirmation Method. The value of the <code>jkt</code> member MUST be the base64url encoding of the JWK SHA-256 Thumbprint of the DPoP public key (in JWK format) to which the Access Token is bound.	RFC 9449

Validation Steps

1. The Wallet Instance **MUST** encrypt long-lived sender-constrained Access Token before storing it.

5.4. Credential Endpoint

1. The Credential Endpoint is an HTTP API at the QEAA Provider and **MUST** accept HTTP `POST` using the `application/json` media type.
2. It **MUST** issue a Credential as approved by the End-User upon presentation of a valid Access Token, representing this approval, as defined in [OPENID4VCI](#).
3. It **MUST** accept only sender-constrained Access Tokens.
4. It **MUST** issue Credentials that are cryptographically bound to the identifier of the End-User who possesses the Credential (Wallet Instance).
5. It **MUST** use the `immediate` Credential Response type.
6. It **MUST** use `c_nonce` and `c_nonce_expires_in` to support Credential updates.
7. It **MUST** use DPoP Authentication Scheme ([RFC 9449](#)) by accepting the sender-constrained access token in the `Authorization` header and the DPoP `proof` JWT in the DPoP header.
8. It **MUST** support `jwt Key Proof` as described in [OPENID4VCI](#).

Credential Request

1. The Wallet Instance **MUST** create a new DPoP `proof` for the Credential Endpoint and bind it to access token using `ath` claim. It **MUST** be included in an HTTP request using the DPoP header parameter as described in [RFC 9449](#).
2. It **MUST** include sender-constrained Access Token in `Authorization` header as described in [RFC 9449](#).
3. It **MUST** use the following parameters in the entity-body of the HTTP `POST` request, using the `application/json` media type:

Claim	Description	Reference
<code>format</code>	Format of the Credential to be issued. It MUST be set to <code>mso_mdoc</code> as published in Credential Issuer Metadata.	OPENID4VCI
<code>doctype</code>	JSON string identifying the credential type as defined in ISO/IEC 18013-5:2021 . It MUST be set to <code>org.iso.18013.5.1.mDL</code> .	OPENID4VCI , Section E.2.5
<code>claims</code>	A JSON object containing a list of name/value pairs, where the name is a certain namespace as defined in ISO/IEC 18013-5:2021 (or any profile of it) and the value is a JSON object. It MUST be defined as in Credential Request Claims Object .	OPENID4VCI , Section E.2.2, E.2.5
<code>proof</code>	JSON object containing proof of possession of the key material the issued credential shall be bound to. The proof object MUST contain the mandatory claims as defined in Credential Request Proof Object table.	OPENID4VCI

Credential Request Claims Object

Claim	Description	Reference
-------	-------------	-----------

org.iso.18013.5.1	A JSON object containing a list of name/value pairs, where the name is a claim name value that is defined in the respective namespace and is offered in the Credential.	OpenID4VCI
-------------------	---	------------

Credential Request Proof Object

Claim	Description	Reference
proof_type	JSON string denoting the proof type. It MUST be set to <code>jwt</code> .	OPENID4VCI
jwt	The JWT used as proof of possession. It MUST be signed by the private key defined in <code>WIA_cnf</code> claim. It MUST contain JWT as defined in jwt Key Proof	OPENID4VCI

jwt Key Proof

The `jwt Key Proof` type MUST contain following header/payload claims:

Header

Claim	Description	Reference
Claim	Description	Reference
alg	A digital signature algorithm identifier such as per IANA "JSON Web Signature and Encryption Algorithms" registry. It MUST NOT be set to <code>none</code> or with a symmetric algorithm (MAC) identifier.	OPENID4VCI, RFC 7515, RFC 7517
typ	It MUST be set to <code>openid4vci-proof+jwt</code>	OPENID4VCI, RFC 7515, RFC 7517
jwk	It MUST contain the key material the new Credential shall be bound to.	OPENID4VCI, RFC 7515, RFC 7517

Payload

Claim	Description	Reference
iss	It MUST be set to <code>sub</code> claim value of the WIA .	OPENID4VCI, RFC 7517
aud	It MUST be set to the identifier of the QEAA Provider.	OPENID4VCI
iat	It MUST be set to the time of the JWT issuance as a UNIX Timestamp	OPENID4VCI, RFC 7519
nonce	It MUST be set to the <code>c_nonce</code> value returned by the Token Endpoint.	OPENID4VCI

DPoP proof JWT

In addition to the values that are defined in the Token Endpoint, the proof MUST contain following claim:

Claim	Description	Reference
ath	Hash of the <code>Access Token</code> . The value MUST be the result of a <code>base64url</code> encoding (as defined in Section 2 of RFC 7515) the SHA-256 hash of the ASCII encoding of the associated <code>Access Token</code> value.	RFC 9449, RFC 7515

Validation Steps

1. The Credential Endpoint MUST validate the `DPoP proof` sent in the `DPoP Header` as defined in [\[RFC 9449\]](#) Section 4.3. If the `DPoP proof` is invalid, the Credential Endpoint MUST return an error response with `invalid_dpop_proof` as the value of the `error` parameter.
2. If request to this endpoint is made without `DPoP proof JWT` or the `Access Token` is not sender-constrained the server MUST return 401 HTTP response status with `WWW-Authenticate` header as defined in [RFC 9449](#), Section 7.1
3. It must MUST validate the sender-constrained access token from `Authorization` header.
4. It must MUST validate the [Credential Request](#) parameters.
5. It must MUST validate the `jwt Key Proof` as described in [OPENID4VCI](#), Section 7.2.2.

Credential Response

1. Credential Response can be `immediate` or `deferred`. This document SHALL implement `immediate` response.
2. It MUST send 201 HTTP status code on successful Credential Response.
3. On failed Credential Response it MUST use error codes as described in [OPENID4VCI](#), Section 7.3.1.
4. Credential Response MUST be sent using `application/json` content type and contain following claims:

Claim	Description	Reference
format	It MUST be set to <code>mso_mdoc</code>	OPENID4VCI
credential	Contains the issued QEAA. It MUST be base64url-encoded JSON string in ISO/IEC 18013-5:2021 format. It MUST contain CBOR encoded mDL as described in MDOC-CBOR Format section.	OPENID4VCI , Appendix E
c_nonce	JSON string containing a nonce value to be used to create a proof of possession of the key material when requesting a further credential or for the renewal of a credential.	OPENID4VCI
c_nonce_expires_in	JSON integer corresponding to the <code>c_nonce</code> lifetime in seconds.	OPENID4VCI

Validation Steps

1. Wallet Instance MUST check that the response contains all the mandatory parameters and values are validated according to [Credential Response](#)
2. It MUST validate that the QEAA Provider is not revoked before the issued credential issuing time.
3. It MUST validate that the Issued Credential is signed by corresponding QEAA Provider.
4. It MUST store `c_nonce`, `c_nonce_expires_in` claims to perform credential update without authorization code flow.
5. It MUST perform credential update before `c_nonce_expires_in`.

5.5. Metadata Endpoints

5.5.1. Authorization Server

Authorization Servers publishing metadata MUST make a JSON document available at the path formed by concatenating the string `/.well-known/oauth-authorization-server` to the `Authorization Server Issuer Identifier`. If the `Authorization Servers Issuer` value contains a path component, any terminating `/` MUST be removed before appending `/.well-known/oauth-authorization-server`.

Authorization Server Metadata parameters are discussed in [RFC 8414](#) Section 2

Claim	Description	Reference
issuer	The Authorization Server Issuer Identifier, which is a URL that uses the "https" scheme and has no query or fragment components.	RFC 8414
authorization_endpoint	URL of the authorization server's authorization endpoint.	RFC 8414 , RFC 6749
token_endpoint	URL of the authorization server's token endpoint.	RFC 8414 , RFC 6749
jwtks_uri	URL of the authorization server's JWK Set. The referenced document contains the signing key(s) the client uses to validate signatures from the authorization server. This URL MUST use the <code>https</code> scheme. The JWK Set MAY also contain the server's encryption key or keys, which are used by clients to encrypt requests to the server. When both signing and encryption keys are made available, a <code>use</code> (public key use) parameter value is REQUIRED for all keys in the referenced JWK Set to indicate each key's intended usage.	RFC 8414 , RFC 7517
grant_types_supported	JSON array containing a list of the OAuth 2.0 grant type values that this authorization server supports. It MUST be set to <code>authorization_code</code>	RFC 8414
response_types_supported	JSON array containing a list of the OAuth 2.0 <code>response_type</code> values that this authorization server supports. It MUST be set to <code>code</code>	RFC 8414
token_endpoint_auth_methods_supported	JSON array containing a list of client authentication methods supported by this PAR and Token endpoint. It MUST be set to contain value <code>attest_jwt_client_auth</code> .	RFC 8414 , RFC 9126 , RFC 7521 , OAuth 2.0 Attestation-Based Client Authentication
pushed_authorization_request_endpoint	The URL of the pushed authorization request endpoint at which a client can post an authorization request to exchange for a <code>request_uri</code> value usable at the authorization server.	RFC 9126

require_pushed_authorization_requests	Boolean parameter indicating whether the authorization server accepts authorization request data only via PAR. It MUST be set to true	RFC 9126
dpop_signing_algorithm_values_supported	A JSON array containing a list of the JWS alg values supported by the authorization server for DPoP proof JWTs. It MUST be set to values RS256, RS384, RS512, ES256, ES384, ES512	RFC 7518 , RFC 9449
code_challenge_methods_supported	JSON array containing a list of Proof Key for Code Exchange (PKCE) code challenge methods supported by this authorization server. It MUST be set to S256	RFC 7636

The following is a non-normative example of Authorization Server Metadata, that requires authorization request to be sent with PAR, defines authentication methods for Token/PAR endpoints, signals support for sender constrained access tokens using DPoP and PKCE.

```
{
  "issuer": "https://as.example.com",
  "authorization_endpoint": "https://as.example.com/authorization",
  "token_endpoint": "https://as.example.com/token",
  "jwks_uri": "https://as.example.com/jwks",
  "response_types_supported": [
    "code"
  ],
  "token_endpoint_auth_methods_supported": [
    "attest_jwt_client_auth"
  ],
  "pushed_authorization_request_endpoint": "https://as.example.com/par",
  "require_pushed_authorization_requests": true,
  "dpop_signing_alg_values_supported": [
    "RS256",
    "RS384",
    "RS512",
    "ES256",
    "ES384",
    "ES512"
  ],
  "code_challenge_methods_supported": [
    "S256"
  ]
}
```

5.5.2. Credential Issuer

1. The Credential Issuer's configuration **SHALL** be retrieved using the Credential Issuer Identifier.
2. Credential Issuers publishing metadata **MUST** make a JSON document available at the path formed by concatenating the string /.well-known/openid-credential-issuer to the Credential Issuer Identifier. If the Credential Issuer value contains a path component, any terminating / **MUST** be removed before appending /.well-known/openid-credential-issuer.
3. The path formed following the steps above **MUST** point to a JSON document compliant with [OPENID4VCI](#) specification. The document **MUST** be returned using the application/json media type.

Credential Issuer Metadata Parameters are discussed in [OPENID4VCI](#) Sections 10.2 and E.2.2 for credentials complying with [ISO/IEC 18013-5:2021](#)

Claim	Description	Reference
credential_issuer	The Credential Issuer Identifier.	OpenID4VCI
authorization_server	Identifier of the OAuth 2.0 Authorization Server the Credential Issuer relies on for authorization. If this element is omitted, the entity providing the Credential Issuer is also acting as the AS, i.e., the Credential Issuer's identifier is used as the OAuth 2.0 Issuer value to obtain the Authorization Server metadata	OpenID4VCI , RFC 8414
credential_endpoint	URL of the Credential Issuer's Credential Endpoint. This URL MUST use the https scheme and MAY contain port, path, and query parameter components.	OpenID4VCI
display	An array of objects, where each object contains display properties of a Credential Issuer for a certain language. It MUST be defined as in Display Object .	OpenID4VCI

credentials_supported	A JSON array containing a list of JSON objects, each of them representing metadata about a separate credential type that the Credential Issuer can issue. The JSON objects in the array MUST conform to the structure defined in OpenID4VCI, Section 10.2.3.1. and as defined in Credentials Supported Object	OpenID4VCI
-----------------------	--	------------

Display Object

Claim	Description	Reference
name	String value of a display name for the claim.	OpenID4VCI
locale	String value that identifies language of this object represented as language tag values defined in BCP47 RFC 5646. There MUST be only one object for each language identifier.	OpenID4VCI, RFC 5646

Credentials Supported Object

Claim	Description	Reference
format	A JSON string identifying the format of this credential. It MUST be set to <code>mso_mdoc</code> .	OpenID4VCI
doctype	JSON string identifying the credential type as defined in ISO/IEC 18013-5:2021. It MUST be set to <code>org.iso.18013.5.1.mDL</code> .	OpenID4VCI, ISO/IEC 18013-5:2021
cryptographic_binding_methods_supported	A JSON array containing a list of supported cryptographic binding methods. It MUST be set to <code>cose_key</code> .	OpenID4VCI
proof_types_supported	A JSON array of case sensitive strings, each representing <code>proof_type</code> that the Credential Issuer supports. It MUST be set to <code>jwk</code> .	OpenID4VCI
display	A JSON array containing a list of JSON objects, where each object contains the display properties of the supported credential for a certain language. It MUST defined as in Credentials Supported Display Object .	OpenID4VCI
claims	A JSON object containing a list of name/value pairs, where the name is a certain namespace as defined in ISO/IEC 18013-5:2021 (or any profile of it) and the value is a JSON object. It MUST defined as in Credentials Supported Claims Object .	OpenID4VCI

Credentials Supported Display Object

Claim	Description	Reference
name	String value of a display name for the claim.	OpenID4VCI
locale	String value that identifies language of this object represented as language tag values defined in BCP47 RFC 5646. There MUST be only one object for each language identifier.	OpenID4VCI, RFC 5646
logo	A JSON object with information about the logo of the Credential.	OpenID4VCI
description	String value of a description of the Credential.	OpenID4VCI
background_color	String value of a background color of the Credential represented as numerical color values defined in CSS Color Module Level 37.	OpenID4VCI, CSS-Color
text_color	String value of a text color of the Credential represented as numerical color values defined in CSS Color Module Level 37	OpenID4VCI, CSS-Color

Credentials Supported Claims Object

Claim	Description	Reference
mandatory	Boolean which when set to true indicates the claim MUST be present in the issued Credential. It MUST be set to <code>true</code> for all mandatory claims defined in ISO/IEC 18013-5:2021	OpenID4VCI, ISO/IEC 18013-5:2021
display	A JSON array containing a list of JSON objects, where each object contains display properties of a certain claim in the Credential for a certain language. It MUST defined as in Display Object .	OpenID4VCI

The following is a non-normative example of Credential Issuer Metadata with `org.iso.18013.5.1.mDL` as single supported credential type and minimal set of mandatory claims as defined in [ISO/IEC 18013-5:2021](#).

```
{
  "credential_issuer": "https://credential-issuer.example.com",
  "authorization_server": "https://as.example.com",
  "credential_endpoint": "https://credential-issuer.example.com/credential",
  "display": [
    {
```

```

    "name": "Transpordiamet",
    "locale": "et-EE"
  },
  {
    "name": "Transport Administration",
    "locale": "en-US"
  }
],
"credentials_supported": [
  {
    "format": "mso_mdoc",
    "doctype": "org.iso.18013.5.1.mDL",
    "cryptographic_binding_methods_supported": [
      "cose_key"
    ],
    "proof_types_supported": [
      "jwt"
    ],
    "display": [
      {
        "name": "Mobiilne juhiluba",
        "locale": "et-EE",
        "logo": {
          "url": "https://examplestate.com/public/mdl.png",
          "alt_text": "mobiilse juhiloa kandiline kujund"
        },
        "background_color": "#12107c",
        "text_color": "#FFFFFF"
      },
      {
        "name": "Mobile Driving License",
        "locale": "en-US",
        "logo": {
          "url": "https://examplestate.com/public/mdl.png",
          "alt_text": "a square figure of a mobile driving license"
        },
        "background_color": "#12107c",
        "text_color": "#FFFFFF"
      }
    ]
  },
  {
    "name": "Mobile Driving License",
    "locale": "en-US",
    "logo": {
      "url": "https://examplestate.com/public/mdl.png",
      "alt_text": "a square figure of a mobile driving license"
    },
    "background_color": "#12107c",
    "text_color": "#FFFFFF"
  }
],
"claims": {
  "org.iso.18013.5.1": {
    "given_name": {
      "mandatory": true,
      "display": [
        {
          "name": "Eesnimi",
          "locale": "et-EE"
        },
        {
          "name": "Given Name",
          "locale": "en-US"
        }
      ]
    },
    "family_name": {
      "mandatory": true,
      "display": [
        {
          "name": "Perekonnanimi",
          "locale": "et-EE"
        },
        {
          "name": "Surname",
          "locale": "en-US"
        }
      ]
    },
    "birth_date": {
      "mandatory": true
    },
    "issue_date": {
      "mandatory": true
    },
    "expiry_date": {
      "mandatory": true
    },
    "issuing_country": {
      "mandatory": true
    },
    "issuing_authority": {
      "mandatory": true
    },
    "document_number": {

```

```
        "mandatory": true
      },
      "portrait": {
        "mandatory": true
      },
      "driving_privileges": {
        "mandatory": true
      }
    }
  }
}
]
```

6. Security Considerations

[OpenID4VCI](#) Section 11 outlines the various aspects of security issues in credential issuance, including trust establishment between Wallet and Issuer, credential offer endpoint issues, pre-authorized code flow security, credential lifecycle management recommendations, key proof replay prevention, and TLS requirements.

[OpenID4VP](#) Section 12 discusses various security considerations for the verifiable presentations protocol. It discusses preventing replay of the VP Token, various security issues when `direct_post` is used, issues with user authentication using verifiable credentials, issues related to [DIF.PresentationExchange](#) and following TLS best practices.

[OpenID4VC High Assurance Interoperability Profile with SD-JWT VC](#) aims to select features and define a set of requirements for the existing specifications to enable interoperability among Issuers, Wallets and Verifiers of Credentials where a high level of security and privacy is required.

[Security and Trust in OpenID for Verifiable Credentials](#) describes the trust architecture in OpenID for Verifiable Credentials (VCs), outlines security considerations and requirements for the components in an ecosystem, and provides an informal security analysis of the OpenID 4 VC protocols.

[OAuth 2.0 Threat Model and Security Considerations](#) gives additional security considerations for OAuth, beyond those in the OAuth 2.0 specification, based on a comprehensive threat model for the OAuth 2.0 protocol.

[OAuth 2.0 Security Best Current Practice](#) describes the best current security practice for OAuth 2.0. It updates and extends the [OAuth 2.0 Security Threat Model](#) to incorporate practical experiences gathered since OAuth 2.0 was published and covers new threats relevant due to the broader application of OAuth 2.0.

[OAuth 2.0 for Native Apps](#) suggests that OAuth 2.0 authorization requests from native apps should only be made through external user-agents, primarily the user's browser. It discusses the details the security and usability reasons why this is the case and how native apps and authorization servers can implement this best practice. It also recommends using domain-bound iOS Universal Link/Android App links for invoking native applications and are recommended by this specification also.