Implementing age verification with Danish Digital Identity Wallet (DKTB)

Contents

Version	Description	Responsible	Date
0.9	First draft.	The Danish Agency for Digital Government	2025-10-28

Disclaimer: This is not a final document. The document will be updated as the project progresses.

1 Introduction

By spring 2026 the Danish Agency for Digital Government will launch the Danish Digital Identity Wallet (DKTB), an app that will, among other things, enable digital age verification. This document describes how a relying party i.e. a verifier may utilize the DKTB for verifying age of users/customers in online (remote) and proximity usage scenarios. The intention is to provide the technical information necessary, for verifiers to implement support for age attestation from the DKTB. The target audience for this document is therefore IT professionals that are interested in knowing how an age attestation from the DKTB can be presented to an IT system.

Because this document focuses on age verification there is not a general introduction to the Danish Digital Identity Wallet project. If you as a reader are looking for a general introduction, we recommend reading the following paper before engaging with this document (albeit in Danish): Notat om den nationale digitale identitetstegnebog[1]. If you have questions to the project team about the documentation you are welcome to send an email at tegnebog@digst.dk.

The development of the DKTB stems from the eIDAS2-regulation. The DKTB is therefore to a large degree built on standards and specifications agreed upon in the EU, enabling the use of the DKTB across the EU, among other things for age verification. If you are interested in knowing more about the related work in EU, see [ARF] and [AV].

The first section is the present introduction. The introduction also contains a terminology list that defines relevant terms. The second section presents a short description of the DKTB app itself as well as a description of the so-called Proof of Age (PoA) attestation and the verification mechanisms underpinning the issuance and validation of this attestation. Section 3 and 4 presents the two transmission protocols that can be implemented in order to receive an attestation from the DKTB and implementation guidelines for these. The appendix contains guidelines for validation of PoA attestations as well as examples of data structures and the two protocols in action. Figure 1 illustrates the contents in scope of this document:

[1] https://digst.dk/media/hb1lt01c/bilag-1-notat-om-den-digitale-identitetstegnebog.pdf

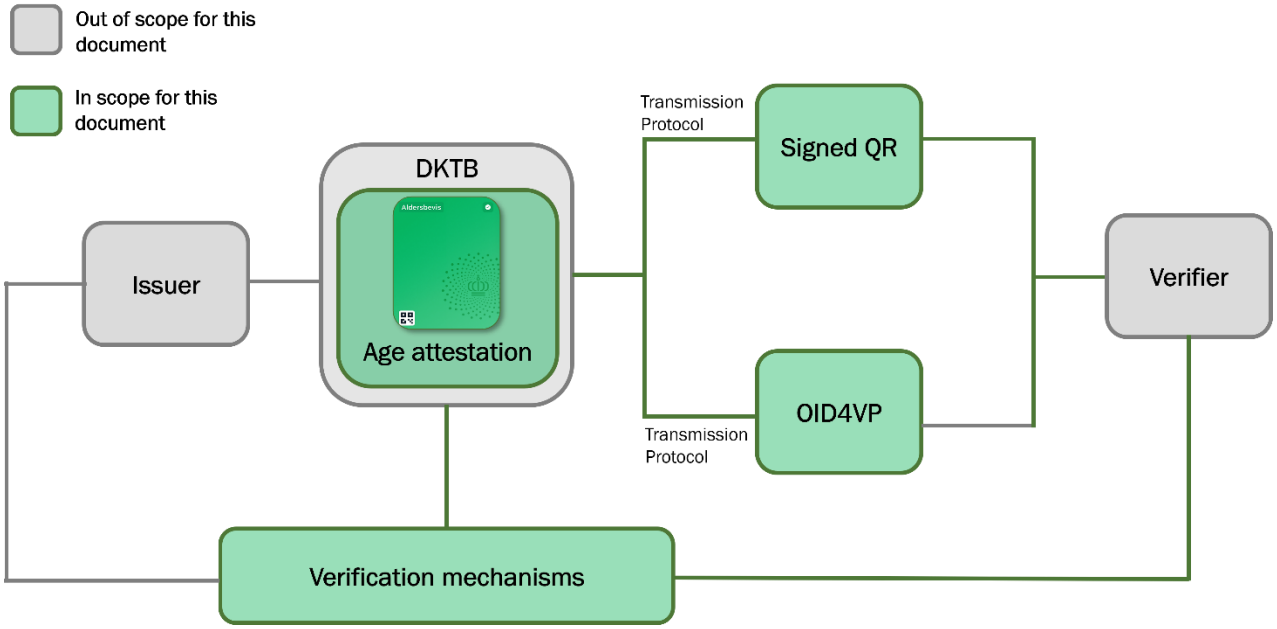

Figure 1: Conceptual Model of the age verification with DKTB.

1.1 Terminology

Attestation (can also interchangeably be called credential): A set of one or more claims about a subject made by an Attestation Issuer.

Attestation Issuer: A public entity that issues attestations to a User.

DKTB: Danish Digital Identity Wallet

Wallet: An entity used by the Holder to receive, store, present, and manage Attestations and key material.

User: A natural person to whom a digital wallet has been issued and who uses it to receive, store, manage and present attestations.

Verifier: An entity that requests and verifies information presented by a User through the DKTB system. In eIDAS2 terminology a verifier is referred to as a relying party.

Presentation: Data that is presented to a Verifier, derived from an Attestation. I.e. Verifiers receive presentations rather than actual attestations.

OID4VP: OpenID for Verifiable Presentations 1.0 [OID4VP] – a specification from OpenID Foundation defining a protocol for requesting and presenting credentials/attestations over the internet.

Signed QR: A bespoke format and protocol for transmitting an attestation with the DKTB where all information required for verification is contained within the signed payload of a multipart QR-code.

Online scenarios: Situations where a verifier interacts with the DKTB user through a digital service, such as a website or mobile application.

Proximity scenarios: Situations where the verifier and the DKTB user are physically present in the same location.

2 Functional description

2.1 The Danish Digital Identity Wallet

The DKTB will function as a personal container that citizens can use to store and share digital attestations in a secure and privacy-preserving manner. Users can initially acquire a DKTB on their smartphone or tablet via Google Play or the Apple App store.

The DKTB is a personal app. It is therefore assumed, that the User will not share it with other people, and that only the User can access and control their personal DKTB. Ultimately, this means that all attestations in a DKTB are expected to pertain to and only be presented by the same User. This is enforced by requiring the user to authenticate using biometry or PIN-code when using the app and only allowing the DKTB application to be installed on one device per user.

2.1.1 Age attestation in the DKTB

After installing the DKTB, The Danish Agency for Digital Government will enable the user to request the issuance of a Proof of Age (PoA) attestation.

The purpose of the **PoA attestation** is that a user can prove that they are over or under a specific age without sharing any personal information or metadata that can be linked to the specific user. The PoA attestation will contain “age over” attributes for specific values e.g. but not limited to 13, 15, 16, 18. The value of the attribute is a Boolean (true/false) indicating whether the User is indeed over the given age, e.g. in the case of a 15-year-old User:

- age_over_13: true
- age_over_15: true
- age_over_16: false
- age_over_18: false
- …

The attestation stored in the DKTB app will contain the entire set of attributes. It will be possible to specifically request a single attribute or multiple attributes at the same time e.g. “age_over_16” and “age_over_18” if required.

In order to ensure that no personal information or any metadata that can enable tracking of a specific individual is shared, multiple PoA attestations will be issued for single use by the user. This means, that if two verifiers compare data from PoA attestations from the same person they cannot be linked.

2.1.2 Verification of the PoA attestation from the Danish Digital Identity Wallet

The PoA attestation is in itself “verifiable” — meaning that the PoA attestation contains the necessary information for a verifier to confirm that it is genuine and valid.

Specifically, the integrity and authenticity of a PoA attestation is protected through Public Key Infrastructure (PKI) and the use of cryptographic mechanisms and digital certificates according to the ISO/IEC 18013-5 standard for mobile driving licenses (mDL). This means that all PoA attestations are issued, stored and presented in the ISO mdoc format and encoded using Concise Binary Object Representation (CBOR). In the following a general overview is provided, but for more detailed info on the ISO mdoc format we refer to [ISO18013-5] and Appendix A (section 5).

In the ISO mdoc format, attestations (mdocs) are identified by a DocumentType (doctype), for PoA this is "eu.europa.ec.av.1". A doctype contains one or more NameSpaces, each of which contain one or more data elements (attributes presented as key-value pairs). For PoA the namespace is also "eu.europa.ec.av.1". In an online scenario, a verifier needs to specify both the doctype, the namespace(s) and the specific attributes when making a request for a presentation. Finally, the mdoc contains a Mobile Security Object (MSO), signed by the Issuer using a private key to prevent tampering, which enables the verifier to verify the validity of the attestation and the included attributes. See Figure 2 for a graphical representation of a PoA attestation in the ISO mdoc format.

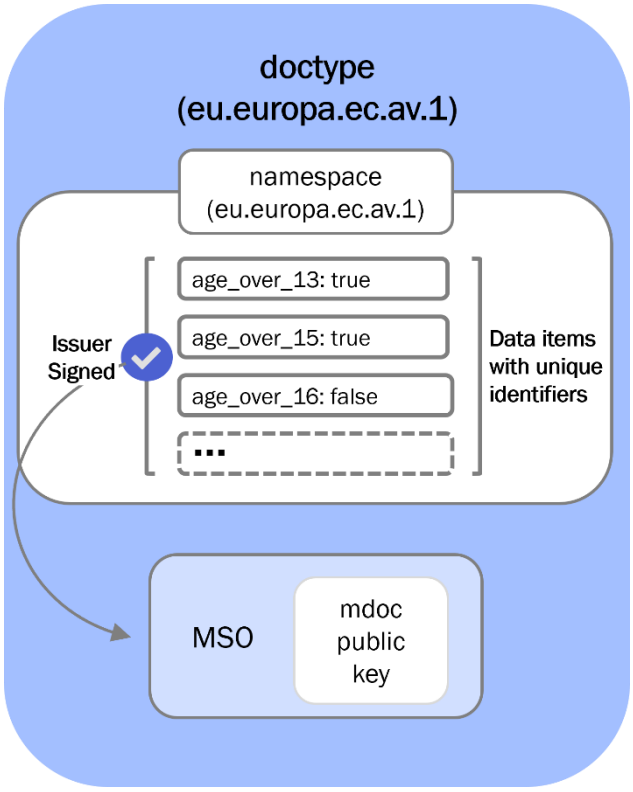

Figure 2: mdoc data model

For more detailed information on the content and data model of the PoA attestation see Appendix A (section 5).

3 Transmission protocol Signed QR

3.1 About Signed QR

Signed QR is a bespoke protocol and format developed for the DKTB. It allows presentation and verification of attestations to take place through a signed QR code exchange between the DKTB and the verifier.

Signed QR is designed for use in proximity scenarios, where a DKTB user presents a QR code containing a presentation of a PoA to a verifier. In this way, the Signed QR protocol allows both the User and the verifier to be offline during this process, as all information required for verification is contained within the signed payload of the QR code.

3.1.1 Sharing attestations with Signed QR

Sharing a PoA attestation via the signed QR protocol is a four-step process (Figure 3):

1. **Presentation** of the QR-code from the PoA attestation by the DKTB user
2. **Verifier scans** the displayed multipart QR-code (in its entirety)
3. **Verifier assembles** the parts into a single, signed payload
4. **Verifier validates** the PoA and continues business process – or rejects to do so, based on outcome.

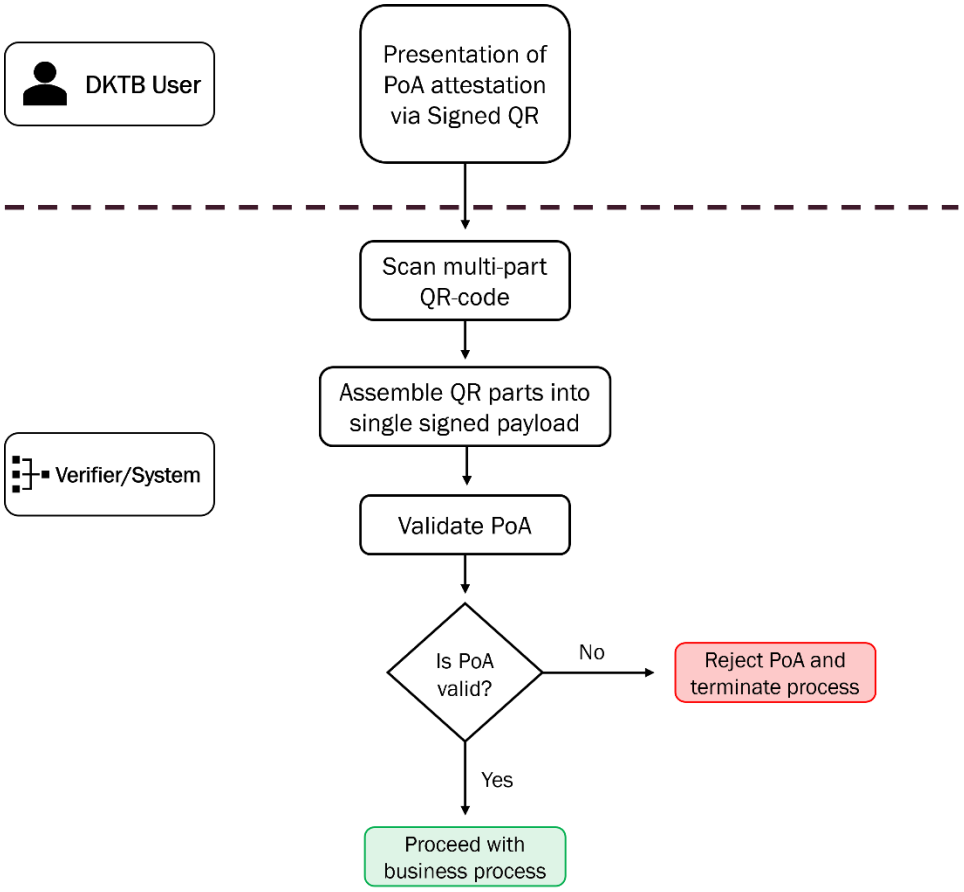

Figure 3: Signed QR age verification flow

Step 1: Presentation of QR-code

A verifier must request the User to present a PoA from the DKTB. Upon this request, the User opens DKTB and selects the PoA ("Aldersbevis"). The content of the QR-code is defined by the user who has the option to configure which age_over_NN attribute to disclose by selecting the button "Vælg hvad du deler" ("Choose what you share"). By default, "age_over_18" will be presented, if the user is indeed above the age of 18. Otherwise, the highest age limit for which the user qualifies will be shown. Once selected, the signed multipart QR code is immediately displayed on the Users device (Figure 4).

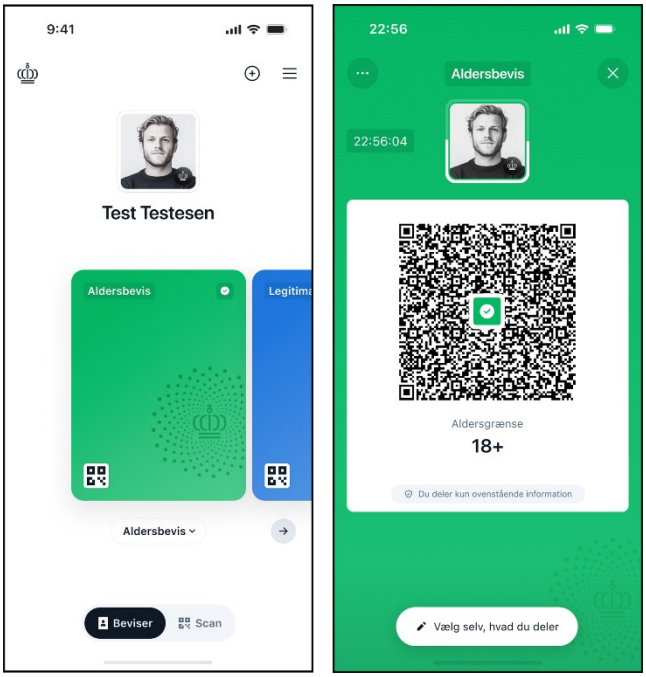

Figure 4: Mock-up of user interface for DKTB PoA Attestation and display of Signed QR

Steps 2 and 3: Scanning and assembling multipart QR codes

Due to the size limitations of QR codes, the presentation is split into multiple parts represented by a QR code each. These individual QR codes are shown in very quick succession and looped. By making use of multipart QR codes, the size of a single QR code can be kept constant, while the only variable is the total amount of QR codes – parts – that make up the presentation. Figure 5 shows a normative example of a QR-code split into four different parts.

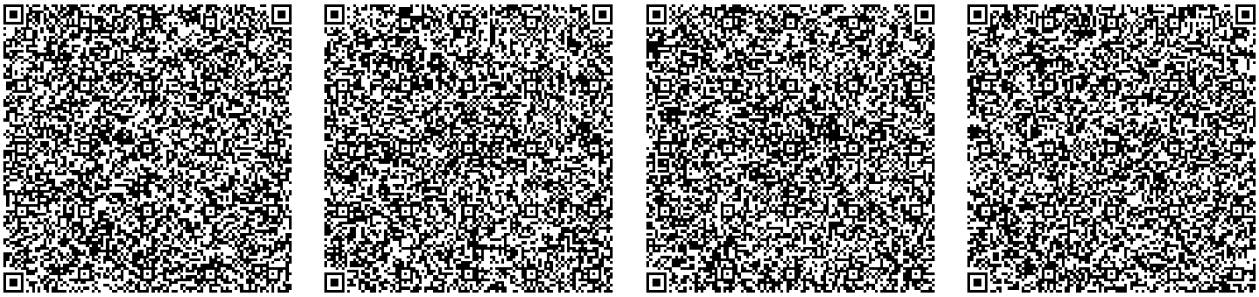

Figure 5: Multipart QR-code

Each part contains a binary CBOR payload. The payload of each QR-code part is structured as an ordered CBOR map as shown below where *n* is equal to 4:

```
PartialQRPayload = {
  "i": uint,          ; Number of this part
  "n": uint,          ; Number of parts in total
  "p": bstr           ; Payload of part i, i.e. pᵢ
}
```

A camera or other QR-code reader is used to scan all *n* parts. When all parts have been scanned, the full binary payload can be assembled.

Step 4: Validate the Proof of Age

In order to validate the contents of the PoA, the steps outlined in Appendix B (section 6) must be followed.

4 Transmission protocol OID4VP

4.1 About OID4VP

OID4VP is a specification developed by the OpenID Foundation that defines a protocol for the secure and privacy-preserving requesting and delivering of so-called “Presentations of Credentials”. Credentials and Presentations can be of any format, including, but not limited to ISO mdoc [ISO.18013-5] – as such OID4VP enables the requesting and presentation of attestations between the DKTB and a verifier. The full OID4VP specification is available at https://openid.net/specs/openid-4-verifiable-presentations-1_0.html.

For verifiers to rely on the OID4VP protocol to receive an age attestation from a DKTB user they must implement OID4VP verifier functionality. In order to simplify this verifier implementation, and to ensure alignment with ongoing EU-initiatives related to age-verification online, the DKTB solution aligns with the EU interoperable age-verification OID4VP profile [AVP]. The main simplification of [AVP] vis-à-vis [OID4VP] is that verifiers are not required to register: Verifiers identify themselves to DKTB simply by the URL, that DKTB use to provide the PoA to the verifier.

4.1.1 Sharing attestations with OID4VP

In order to enable the sharing of a PoA attestation via the OID4VP protocol, the verifier must send an Authorization Request to the DKTB User, including requirements for the attestation(s) that are requested. An Authorization Request is sent to the user via a URL with encoded parameters. The URL containing the authorization request can be shown as a QR code that the user must scan to initiate the presentation flow, or if the verifier interaction takes place on the same device (e.g. in an online setting), the user can instead follow a direct link with the Authorization Request. The URL opens the DKTB and initiates the presentation flow.

In Figure 6 the user scans a QR code whereafter DKTB retrieves the presentation request from the verifier and prompts the user to share the requested PoA attestation (In this example of a +16 age request).

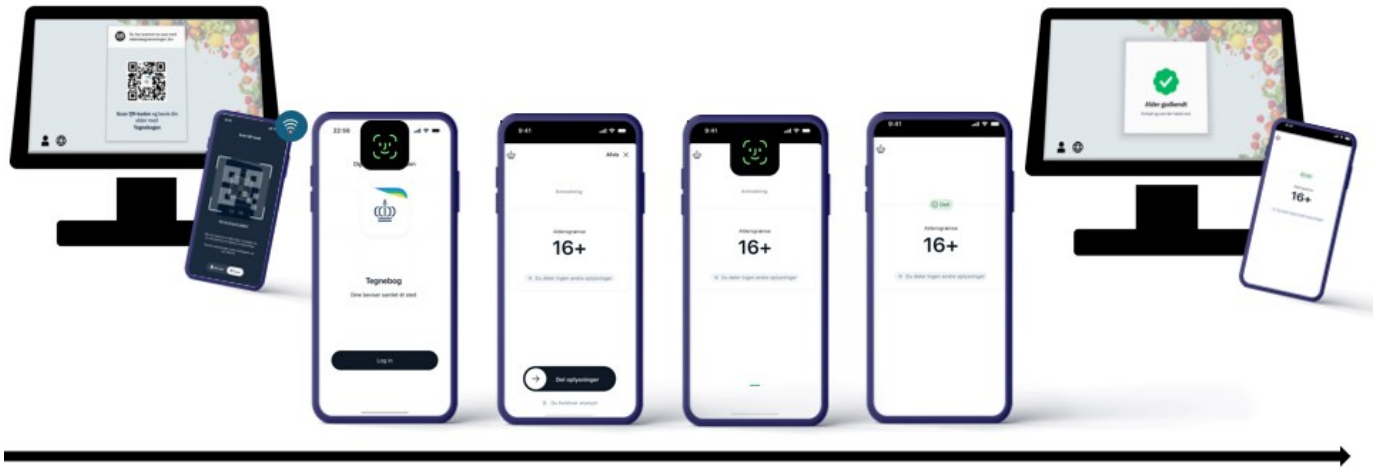

Figure 6: User flow for sharing a PoA via OID4VP

4.1.2 Age verification flow with OID4VP following [AVP]

An example of an interaction between a verifier and DKTB in an OID4VP exchange is shown below in Figure 7. The figure assumes that the verifier application consists of a frontend part (Client) and REST-oriented backend, however other design patterns can be applied based on the business needs and existing infrastructure of the Verifier.

OID4VP specifies exchanges of Authorization Request and Authorization Response – roughly steps 3, 4, 8, 9, 10, 11, and 13 in, which are thus mandatory. The interaction between Verifier Client and Verifier Backend (steps 1, 2, 5, 6, 7, 12, 14, and 15) is not part of the OID4VP protocol, and the description in Table 1 merely suggests an implementation strategy.

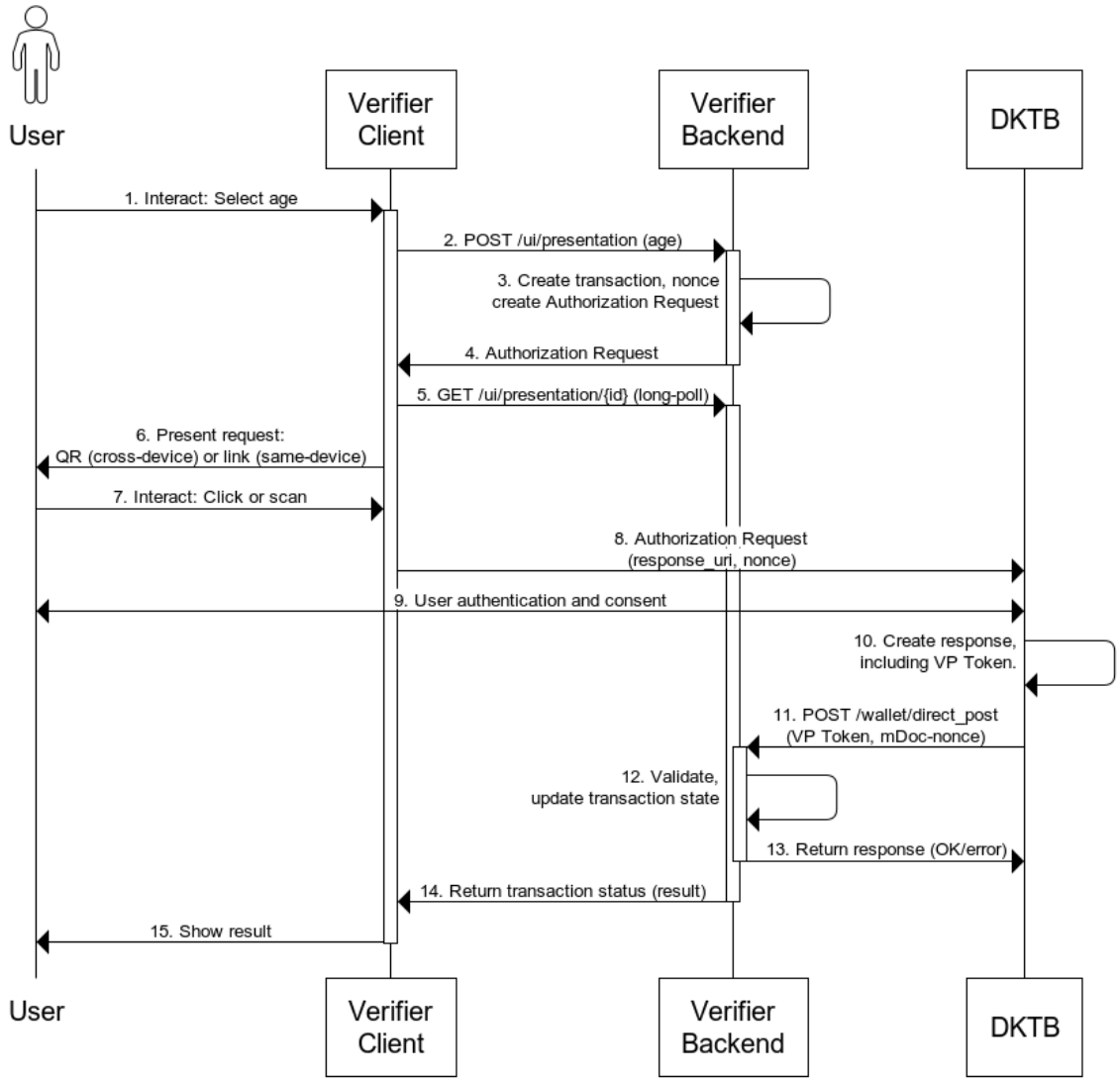

Figure 7: OID4VP age verification flow following [AVP].

Table 1 is a description of each step in the OID4VP age verification flow following [AVP].

Step	Description
1	User interacts with verifier website (client), in some way, that requires their age to be verified. The diagram suggests that the user selects the age to be verified, usually this is a choice made by the verifier application.
2	Client sends the age(s) to be verified to the backend.

3 Backend creates a transaction record with a unique identifier (id). A random nonce is generated and associated to the transaction, and the transaction record is stored. The backend then creates a corresponding OID4VP Authorization Request (as URL encoded values), including a specification of the desired age(s) to verify in the form of a DCQL query (see [OID4VP] section 5 and 6).

4 The Authorization Request is returned to client as response to 2.

5 The client starts a (long)-polling request, awaiting the DKTB response. Long-polling is suggested for optimization reasons.

6 The client now decides how to present the Authorization Request to the user: If the client detects that the user is using a mobile device, it is likely that DKTB is installed on that same device (same-device scenario). In that case, the client should present the Authorization Request as a button/link ("Verify age") in the UI.

If not, the client should encode the Authorization Request in a QR-code[2] and display this in the UI. This allows the user to scan the QR-code using their mobile device with DKTB installed (cross-device scenario).

In the same-device scenario, it is suggested to display another button ("Verify age using other device"), that causes a QR-code to be displayed.

7 User interacts with UI – clicks button/link (same-device) or scans QR-code (cross-device).

8 Authorization Request is sent to DKTB which validates the request and ensures that it conforms to [AVP].

9 User interacts with DKTB and accepts to present the requested attributes of the PoA to the verifier.

10 DKTB creates the presentation (VP Token), which is an encoded DeviceResponse. This involves creating a random mdocGeneratedNonce, constructing SessionTranscript and DeviceAuthenticationBytes5.4, and creating the device signature, using the key that was bound to the PoA during issuance.

11 DKTB POSTs the response with presentation to the response_uri provided by the verifier in the Authorization Request.

12 Verifier backend receives the response, performs validation and updates the transaction record state correspondingly. In case of validation errors, transaction is marked as failed, and an error description (user friendly) should be added to the transaction record.

For details, see 'Appendix D: Guidelines for validating Authorization Response in OID4VP' (section 8).

13 Verifier backend responds to DKTB POST.

[2] This can be done by utilising one of the many available open-source libraries for QR-code generation, such as https://github.com/nayuki/QR-Code-generator/

14	With the transaction now being completed the (long)-polling request can be replied to, in accordance with the status determined by backend validation in step 12. The client response should include ok/not ok and a suitable error message in case of errors.
15	Client shares the result with the user, usually by simply continuing the business operation for successful age verification and by displaying an error message in case of error.

Table 1: Description of each step in the OID4VP age verification flow following [AVP].

Please refer to ‘Appendix E: OID4VP example’ (section 9) for a normative example highlighting the exchanged data elements.

5 Appendix A: Data structures

This appendix will present and describe the data structure(s) related to the exchange of PoA attestations from DKTB.

Attestations in DKTB adhere to the ISO 18013-5 mobile document format (mdoc), which encodes data using CBOR (Concise Binary Object Representation), a binary alternative to JSON optimized for size and speed. CBOR preserves data types precisely and is suitable for signing and verification, but it’s not human-readable — every field and structure must be decoded to inspect its content. When working with mdoc data, you can use decoding tools or libraries such as [CBORZ] to convert it to diagnostic notation for inspection.

To make the specification of mdoc easier to understand, it can be viewed as hierarchical binary tree, where each node represents a CBOR map or array, and each leaf contains a concrete data value (string, number, or byte string). At the root, you have the overall document container, which branches into key sections such as docType, issuerSigned, deviceSigned, and validityInfo. Each of these nodes can in turn contain nested maps — for instance, issuerSigned → nameSpaces → eu.europa.ec.av.1 → age_over_18 → ["true"].

The structure of a mdoc document is provided below, followed by a stepwise walkthrough of the elements based on PoA:

```
- Document
├── docType : DocType
├── issuerSigned
│   ├── ? nameSpaces : NameSpace => [IssuerSignedItemBytes]+
│   │   └── IssuerSignedItemBytes = #6.24(bstr .cbor IssuerSignedItem)
│   │       └── IssuerSignedItem
│   │           ├── digestID : uint
│   │           ├── random : bstr
│   │           ├── elementIdentifier : DataElementIdentifier
│   │           └── elementValue : DataElementValue
│   └── issuerAuth : IssuerAuth
├── deviceSigned
│   ├── nameSpaces : DeviceNameSpacesBytes
│   │   └── DeviceNameSpacesBytes = #6.24(bstr .cbor DeviceNameSpaces)
│   │       └── DeviceNameSpaces
│   │           └── NameSpace => DeviceSignedItems
│   │               └── DeviceSignedItems
│   │                   └── DataElementIdentifier => DataElementValue
│   └── deviceAuth
│       ├── deviceSignature : DeviceSignature
```

5.1 Document

Document is the PoA attestation itself and has the following CBOR syntax:

```
Document = {
   "docType" : DocType,           ; Document type returned, always “eu.europa.ec.av.1” for a PoA
   "issuerSigned" : IssuerSigned, ; Returned data elements signed by the issuer
   "deviceSigned" : DeviceSigned, ; The Wallet signature
}
```

5.2 IssuerSigned

Here *IssuerSigned* contains the PoA attestation including the disclosed age_over_NN attribute(s) as selected by the user:

```
IssuerSigned = {
  "nameSpaces" : IssuerNameSpaces,    ; Returned data elements, i.e. age_over_NN attribute(s)
  "issuerAuth"   : IssuerAuth         ; Contains the Mobile Security Object (MSO)
}
```

Here *IssuerNameSpaces* contains the age_over_NN attributes included in the presentation:

```
IssuerNameSpaces = {                            ; Returned data elements for each namespace
   + tstr => [ + IssuerSignedItemBytes ]
}

IssuerSignedItemBytes = #6.24(bstr .cbor IssuerSignedItem) ; Tagged bstr [3]

IssuerSignedItem = {
```

[3] Syntax #6.24 means a CBOR tag (major type 6), tag number 24 (binary data), see https://www.rfc-editor.org/rfc/rfc8949.html#name-encoded-cbor-data-item

```
   "digestID" : uint,                           ; Digest ID for issuer data authentication
   "random" : bstr,                             ; Random value for issuer data authentication
   "elementIdentifier" : DataElementIdentifier, ; Data element identifier, e.g. “age_over_18”
   "elementValue" : DataElementValue            ; Data element value, e.g. “true”
}
```

5.3 Issuer signature

IssuerAuth contains the MSO which is a [COSE] (COSE_Sign1) signature:

```
IssuerAuth = COSE_Sign1               ; As defined in section 4.2 of [COSE]

COSE_Sign1 = [                        ; Contents of the COSE_Sign1 structure of IssuerAuth
   ProtectedHeaderBytes : bstr,       ; Encoded map with header elements included in signature
   UnprotectedHeader,                 ; Map with header elements not included in signature
   Payload : MobileSecurityObjectBytes,; CBOR encoded MSO bytes
   Signature: bstr                    ; The signature
}

ProtectedHeader = {
  1 : -7                              ; alg_id = ES256 (ECDSA w/ SHA-256)
}

UnprotectedHeader = {
   33 : [bstr, bstr, ... ] | bstr
            ; Array with X.509 certificate chain (33) or single bstr with issuer certificate
            ; If given as chain (array), chain begins with issuer certificate
            ; and ends with root-certificate.
            ; If given as single certificate, only issuer certificate with public key
            ; needed to validate MSO is provided.
}

MobileSecurityObjectBytes = #6.24(bstr .cbor MobileSecurityObject)
```

Note: For PoA, the protected header always contains a single element, stating that the signature algorithm used is ECDSA with SHA-256, and the unprotected header always contains a single element, with the issuer certificate and corresponding certificate chain.

The *Payload* is the CBOR encoded Mobile Security Object:

```
MobileSecurityObject = {
   "version" : tstr,                ; Version of the MSO, always “1.0” for PoA
   "digestAlgorithm" : tstr,        ; Message digest algorithm used, always ”SHA-256” for PoA
   "valueDigests" : ValueDigests,   ; Digests of all data elements per namespace
   "deviceKeyInfo" : DeviceKeyInfo, ; Device key to use for presentation
   "docType" : tstr,                ; Same as Document DocType, i.e. “eu.europa.ec.av.1”
   "validityInfo" : ValidityInfo    ; Temporal validity information
}

ValueDigests = {
  + NameSpace => DigestIDs   ; DigestIDs for a given NameSpace, PoA always contains only a
}                            ; single NameSpace: “eu.europa.ec.av.1”.

DigestIDs = {
  + DigestID => Digest       ; Map from DigestID to Digest
}

DigestID = uint              ; DigestID corresponding to the one(s) in IssuerSignedItem
Digest = bstr                ; Digest of the IssuerSignedItems used for data authentication

DeviceKeyInfo = {
   "deviceKey" : DeviceKey
}

DeviceKey = COSE_Key         ; Public key of the device key-pair used to sign the presentation

ValidityInfo = {
   "signed" : tdate,         ; Timestamp at which the MSO signature was created
   "validFrom" : tdate,      ; Timestamp before which the MSO is not yet valid
   "validUntil" : tdate,     ; Timestamp after which the MSO is no longer valid
}
```

The MSO signature can be verified using the public key from the issuer certificate. This is obtained by the verifier as an unprotected header element (X.509 certificate chain, x5chain, label 33)

5.4 Device signature

DeviceSigned holds the signature created by DKTB using the device key, for which the public key is included in the Mobile Security Object.

```
DeviceSigned = {
   "nameSpaces" : tagged empty bstr,    ; Returned data elements, always empty for PoA
   "deviceAuth" : DeviceAuth            ; Contains the device signature
}

DeviceAuth = { ;
   "deviceSignature" : DeviceSignature  ; Signature used for mdoc (device) authentication
}
```

Where *DeviceSignature* is a COSE_Sign1 signature over *DeviceAuthentication* data:

```
DeviceSignature = COSE_Sign1

COSE_Sign1 = [                       ; Contents of the COSE_Sign1 structure of DeviceSignature
   ProtectedHeaderBytes : bstr,      ; Encoded map with header elements included in signature
   UnprotectedHeader,                  ; Map with header elements not included in signature.
```

```
                                        ; This map is empty for DeviceSignature
   Payload : DeviceAuthenticationBytes, ; Tagged and encoded DeviceAuthentication
                                        ; NB: Null when transmitted
   Signature : bstr                     ; The signature
}

DeviceAuthenticationBytes = #6.24(bstr .cbor DeviceAuthentication)

DeviceAuthentication = [
   "DeviceAuthentication",
   SessionTranscript,                ; See section 7.2
   DocType,                          ; Same as Document DocType, i.e. “eu.europa.ec.av.1”
   DeviceNameSpacesBytes             ; Always tagged bstr of empty map: #6.24(bstr .cbor {})
]
```

Note that DeviceSignature is *detached*, meaning that the COSE_Sign1 data contains a null value for the Payload element when transmitted. As a result, the payload bytes (DeviceAuthenticationBytes) must be **reconstructed** prior to validating the signature. Also note, that the only variable data in DeviceAuthentication is SessionTranscript.

5.5 SessionTranscript for signed QR-codes

The SessionTranscript takes a special form for Documents presented as QR-codes:

```
SessionTranscript = [
  DeviceEngagementBytes : bstr,             ; Always null for signed QR presentations
  EReaderKeyBytes       : bstr,             ; Always null for signed QR presentations
  Handover              : SignedQRHandover
]

SignedQRHandover = [
  mdocGeneratedNonce    : tstr,             ; 16 random bytes, Base64URL-encoded
  validFrom             : uint,             ; Unix timestamp the QR-code was generated at
  validTo               : uint              ; Unix timestamp the QR-code is no longer valid at
]
```

Note that the SessionTranscript purposely binds the presentation Document to the temporal validity of the PoA (validFrom and validTo, i.e. *f* and t in section 7.1) and random data generated by DKTB (mdocGeneratedNonce, i.e. m in section 7.1).

5.6 SessionTranscript for OID4VP

For OID4VP, the SessionTranscript takes a different form:

```
SessionTranscript = [
  DeviceEngagementBytes : bstr,             ; Always null for OID4VP
  EReaderKeyBytes       : bstr,             ; Always null for OID4VP
  Handover              : OID4VPHandover
]

OID4VPHandover = [
  clientIdHash          : bstr,             ; SHA-256 hash of clientIdToHash
  responseUriHash       : bstr,             ; SHA-256 hash of responseUriToHash
  nonce                 : tstr
]

clientIdToHash = [clientId, mdocGeneratedNonce]
responseUriToHash = [responseUri, mdocGeneratedNonce]
```

```
mdocGeneratedNonce : tstr,      ; Available from apu header
clientId           : tstr,      ; “client_id” from Authorization Request (see section 9.1)
responseUri        : tstr,      ; “response_uri” from Authorization Request
nonce              : tstr       ; “nonce” from Authorization Request
```

Note, that the *DeviceAuthentication* is not included in the response. This means that verifier must reconstruct the payload bytes (DeviceAuthenticationBytes) prior to validating the device signature. This process is illustrated in section 7.2.

5.7 mdocGeneratedNonce

The mdocGeneratedNonce is a random value created by the Wallet during an interaction with a verifier to ensure that each presentation is unique and cannot be reused in later sessions. By including this nonce in the signed response, the mdoc provides proof that the data was generated specifically for the current transaction, thereby protecting against replay attacks.

[version 1.0 of this document will describe in detail how the handover and verification of the mdocGeneratedNonce is performed as part of the communication flow between the mDoc and the verifier.]

6 Appendix B: Validation of PoA with Signed QR

6.1 QRPayload

The byte array *Q* is a binary CBOR encoding of a CBOR map with the following syntax:

```
QRPayload = {
  "m": tstr           ; mdocGeneratedNonce – random data generated by the Wallet
  "f": uint,          ; validFrom - Unix timestamp
  "t": uint,          ; validTo - Unix timestamp
  "d": bstr           ; CBOR encoding of Document
}
```

6.2 Steps for validation of PoA with Signed QR

With *Q* in hand, parse it into the QRPayload CBOR-map, and validate the PoA according to the steps shown in Table 2:

Step	Description
1	Validate that the CBOR map contains four parts named *m*, *f, t,* and *d*.
2	Validate that *f* and *t* are integers and construct the corresponding Unix UTC timestamps *validFrom* and *validTo*.
3	Validate that the current timestamp is equal to or later than *validFrom* (cf. 2), allowing for a clock skew.
4	Validate that *validTo* (cf. 2) is equal to or later than the current timestamp, allowing for a clock skew.
5	Validate that *m (mdocGeneratedNonce)* is Base64URL encoding of 16 bytes.

6	Extract the x5c header in MSO unprotected header, validate the certificate chain, and that the chain terminates in a root certificate registered in your issuer trust list.
7	Validate the issuer signature *(IssuerAuth)* using the public key from the issuer certificate from step 6.
8	Calculate the digest value for every IssuerSignedItem in the *Document/IssuerNameSpaces* structure and verify that these calculated digests equal the corresponding digest values in the signed MSO (*DigestIDs*). (see also section 7.3 for more info and an example)
9	Verify that the DocType in the MSO and the DocType in the Document structure both equal "eu.europa.ec.av.1".
10	Validate the elements in the *ValidityInfo* structure, i.e. verify: a. that the 'signed' date is within the validity period of the certificate in the MSO header and b. that the '*validFrom'* element shall be earlier than or equal to the current timestamp, and c. that the '*validUntil*' element shall be equal to or later than the current timestamp
11	Validate the device signature (*DeviceAuth)* as described in section 7.2: a. Construct the *SessionTranscript*. b. Use *SessionTranscript* to construct *DeviceAuthentication.* c. Validate the *DeviceAuth* COSE_Sign1 signature using CBOR encoding of *DeviceAuthentication* as external payload and the public key from the MSO (*DeviceKeyInfo/DeviceKey*).
12	Validate, that *mdocGeneratedNonce* is not replayed: a. Maintain a set of all presented *mdocGeneratedNonce* values, that are valid according to *validTo* cf. 2. b. Verify that *mdocGeneratedNonce* is not in this set. c. Add *mdocGeneratedNonce* to the set.
13	Validate that the disclosed age_over_NN attribute(s) satisfy the age requirement of the application.

Table 2: Validation of PoA with Signed QR

7 Appendix C: Signed QR-code example

The included examples only disclose the mandatory attribute *age_over_18* for brevity. In practice, PoA attestations can disclose multiple age_over_NN claims in addition to age_over_18, e.g. “age_over_13”, “age_over_15”. If multiple are be disclosed, they will be included as additional IssuerSignedItems.

All examples use the issuer certificate and corresponding certificate chain described in section 9.3.

When working with the examples, it is very convenient to use online tools such as [CBORZ] to parse binary data and examine the corresponding CBOR data structures.

7.1 Multiple QR-codes

The four QR-codes shown in Figure 5 contain the following binary data:

i	p_i as Base64Url
0	o2FpAGFuBGFwWQKfpGFtdlF1M011a3Q0d3doN3ZwOGs3LUtxUUFhZhpo-KevYXQaaPioY2FkWQpQo2dkb2NUeXBlcWV1LmV1cm9wYS5lYy5hdi4xbGlzc3VlclNpZ25lZKJqbmFtZVNwYWNlc6FxZXUuZXVyb3BhLmVjLmF2LjGB2BhYYKRoZGlnZXN0SUQDZnJhbmRvbVgg3b5VcHhIubG2JM0qiLEtS4tvTrcYP7ja6SO3_GV_XOpxZWxlbWVudElkZW50aWZpZXJrYWdlX292ZXJfMThsZWxlbWVudFZhbHVl9Wppc3N1ZXJBdXRohEOhASahGCGDWQJOMIICSjCCAfCgAwIBAgIUQkfIkGJcupTqa_SxzW0Bl4Nsvc8wCgYIKoZIzj0EAwIwbTELMAkGA1UEBhMCREsxEzARBgNVBAcMCkvDuGJlbmhhdm4xITAfBgNVBAoMGERpZ2l0YWxpc2VyaW5nc3N0eXJlbHNlbjEMMAoGA1UECwwDS0VBMRgwFgYDVQQDDA9ES1RCIElzc3VpbmcgQ0EwHhcNMjUwNjE4MTQyMzUxWhcNMjYwNjE4MTQyMzUxWjB0MQswCQYDVQQGEwJESzETMBEGA1UEBwwKS8O4YmVuaGF2bjEhMB8GA1UECgwYRGlnaXRhbGlzZXJpbmdzc3R5cmVsc2VuMQwwCgYDVQQLDANLRUExHzAdBgNVBAMMFkRLVEIgQ3JlZGVudGlhbCBJc3N1ZXIwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAASkcFWfqRC7lkqfNIWs8st8dcRfpnFaWF8RLkxRuw4zE94UTPTM85Au8lTbvwz6YEdwkXVlTLb3ipizsPY9pwgfo2cwZTAOBgNVHQ8BAf8EBAMCB4AwHwYDVR0jBBgwFoA
1	o2FpAWFuBGFwWQKfFKIpVen1SlQ2mm7eWxDXsBDglv3iMB0GA1UdDgQWBBQwUoAEIiXsQfN2bcL3c_uKFUnytDATBgNVHSUEDDAKBggrBgEFBQcDAzAKBggqhkjOPQQDAgNIADBFAiEAz6cGyitgDHQYBfn87kW2Ww4sJyJagTP2iElqYc7QyawCIBUqQWHPNFRx_pqFYJXkVQxeozlVWvTlOP_MnD3nU5KwWQJGMIICQjCCAemgAwIBAgIUUyHgT02vW2tghii4g2oBl2LVaEowCgYIKoZIzj0EAwIwbzELMAkGA1UEBhMCREsxEzARBgNVBAcMCkvDuGJlbmhhdm4xITAfBgNVBAoMGERpZ2l0YWxpc2VyaW5nc3N0eXJlbHNlbjEMMAoGA1UECwwDS0VBMRowGAYDVQQDDBFES1RCIFRlc3QgUm9vdCBDQTAeFw0yNTA2MTIwNjMwNDJaFw0yNjA2MTIwNjMwNDJaMG8xCzAJBgNVBAYTAkRLMRMwEQYDVQQHDApLw7hiZW5oYXZuMSEwHwYDVQQKDBhEaWdpdGFsaXNlcmluZ3NzdHlyZWxzZW4xDDAKBgNVBAsMA0tFQTEaMBgGA1UEAwwRREtUQiBUZXN0IFJvb3QgQ0EwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAAT1fhK521uTMp38D2RfAjyp2N8u5NPcPu74TSsyxUjWxm39hXuC15n4-vjUD0kbSncgc3Sht5ZuryRuwy2B3uBHo2MwYTAPBgNVHRMBAf8EBTADAQH_MA4GA1UdDwEB_wQEAwIBBjAfBgNVHSMEGDAWgBRAjjvYDiIZXrQLco-s-vzZ4XH6MjAdBgNVHQ4EFgQUQI472A4iGV60C3KPrPr82eFx-jIwCgYIKoZIzj0EAwI
2	o2FpAmFuBGFwWQKfA0cAMEQCIGA_y1DbxQzZ_P37Ud9TLbPSefSAMVt3Mv4hE9c7B_l2AiAinsjwFrnmAoKbs_iT-ydSVhtLcvYlqmxIhpRpWKzB6VkCRTCCAkEwggHnoAMCAQICFFMbo_iBd2GGe78m7zaRAZd5ITTuMAoGCCqGSM49BAMCMG8xCzAJBgNVBAYTAkRLMRMwEQYDVQQHDApLw7hiZW5oYXZuMSEwHwYDVQQKDBhEaWdpdGFsaXNlcmluZ3NzdHlyZWxzZW4xDDAKBgNVBAsMA0tFQTEaMBgGA1UEAwwRREtUQiBUZXN0IFJvb3QgQ0EwHhcNMjUwNjE2MTQyNTA4WhcNMjYwNjE2MTQyNTA4WjBtMQswCQYDVQQGEwJESzETMBEGA1UEBwwKS8O4YmVuaGF2bjEhMB8GA1UECgwYRGlnaXRhbGlzZXJpbmdzc3R5cmVsc2VuMQwwCgYDVQQLDANLRUExGDAWBgNVBAMMD0RLVEIgSXNzdWluZyBDQTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABDqBXV2b3hm9ch2DfHjnQ1a1vpKKjWN8zEaeFXHSwWvCpc0mxO0Hm0_VsgxKXwRPD6IW7looQsi0ceqKYMvDCfCjYzBhMA8GA1UdEwEB_wQFMAMBAf8wDgYDVR0PAQH_BAQDAgEGMB8GA1UdIwQYMBaAFECOO9gOIhletAtyj6z6_NnhcfoyMB0GA1UdDgQWBBSiKVXp9UpUNppu3lsQ17AQ4Jb94jAKBggqhkjOPQQDAgNIADBFAiBBpe__iIaS1chce2aiexZIHVIBHrp9OCqop1r5inq9EgIhAMILC-JXgjQleQlQoSsmfR6IgE9KATQypubmxfXLW_dtWQHm2BhZAeGmZ3ZlcnM
3	o2FpA2FuBGFwWQKgaW9uYzEuMG9kaWdlc3RBbGdvcml0aG1nU0hBLTI1Nmx2YWx1ZURpZ2VzdHOhcWV1LmV1cm9wYS5lYy5hdi4xpQFYIL_XSePBUuKzOO6bZ5j4B9-h_ynFl036rVZfgQjNfc9MAlggUoPbCffX2A534_s-_JwodhUnkKlM1UiWPb7ACxb3mz0DWCCDW21WP7yU9zcirh4XjF7f3gevk4mH6w4EbrM3dNq49gRYICTNO9kx3r8dLQ_vnF_LrzoBNGLndCeZlBtPHE-I7w_VBVggpiFm7rY3aN7iTyhoQHit-ui-dlftq0h8zhfbcY5m_QJtZGV2aWNlS2V5SW5mb6FpZGV2aWNlS2V5pQECAyYgASFYIChE6307anxgIcxYtc2kCEHY_PzCWlJCuz44CMlBJj4gIlggiF1tM8YPvmoBkpqzZzGRafOGC9abT9bNan_adlWVxspnZG9jVHlwZXFldS5ldXJvcGEuZWMuYXYuMWx2YWxpZGl0eUluZm-jZnNpZ25lZMB0MjAyNS0xMC0yMlQwOTo0NToxOVppdmFsaWRGcm9twHQyMDI1LTEwLTIyVDA5OjQ1OjE5Wmp2YWxpZFVudGlswHQyMDI2LTAxLTIwVDEwOjQ1OjE5WlhAzgmzf9DHxLA5bbwCZ9e0KoXil_ceVLbg3tqXvrROso3piJT0-5-JLPwnaJ0GsKQY2P7VPm9hbsWVvBDbrBjxbGxkZXZpY2VTaWduZWSiam5hbWVTcGFjZXPYGEGgamRldmljZUF1dGihb2RldmljZVNpZ25hdHVyZdKEQ6EBJqD2WEAqwin5LgWRCG4H_zcVE5ko3Ssz_f3ovtDYOtJAYOD-bZQtSvSQMsJr1WydETJGASX-qxYpRjd1KCdAF2RZOctw

For i = 1 (the second part), the data represents this CBOR data structure:

```
{
  "i": 1,
  "n": 4,
  "p":
h'14a22955e9f54a54369a6ede5b10d7b010e096fde2301d0603551d0e0416041430528004222 5ec41f3766dc2f773fb8a1549f2b43
0130603551d25040c300a06082b060105050703033 00a06082a8648ce3d0403020348003045022100cfa706ca2b600c741805f9fcee
45b65b0e2c27225a8133f688496a61ced0c9ac0220152a4161cf345471fe9a856095e4550c5ea339555af4e538ffcc9c3de75392b05
902463082024230820 1e9a003020102020145321e04f4daf5b6b608628b8836a019762d5684a300a06082a8648ce3d040302306f310b
300906035504061302444b311330110603550406 1302444b311330110603550407 0c0a4bc3b862656e6861766e3121301f060355040a0c184469676974616c6973657
2696e677373747972656c73656e310c300a060355040b0c034b4541311a301806035504030c11444b5442205465737420526f6f7420
4341301e170d323530363132303633303432 5a170d323630363132303633303432 5a306f310b300906035504061302444b311330110
6035504070c0a4bc3b862656e6861766e3121301f060355040a0c184469676974616c69736572696e677373747972656c73656e310c
300a060355040b0c034b4541311a301806035504030c11444b5442205465737420526f6f7420434130593013060 72a8648ce3d02010
6082a8648ce3d030107034200 04f57e12b9db5b93329dfc0f645f023ca9d8df2ee4d3dc3eeef84d2b32c548d6c66dfd857b82d799f8
faf8d40f491b4a77207374a1b7966eaf246ec32d81dee047a3633061300f0603551d130101ff040530030101ff300e0603551d0f010
1ff040403020106301f0603551d2304183016801 4408e3bd80e22195eb40b728facfafcd9e171fa32301d0603551d0e04160414408e
3bd80e22195eb40b728facfafcd9e171fa32300a06082a8648ce3d040302'
}
```

The four p binary strings (parts for i = 0, 1, 2, 3) are concatenated into a single binary string (Q), which represents this CBOR data structure:

```
{
  "m": "Qu3Mukt4wwh7vp8k7-KqQA",
  "f": 1761126319,
  "t": 1761126499,
  "d":
h'a367646f6354797065716575 2e6575726f70612e65632e61762e316c6973737565725369676e6564a26a6e616d65537061636573a
17165752e6575726f70612e65632e61762e3181d8185860a4686469676573744944036672616e646f6d5820ddbe55707848b9b1b624
cd2a88b12d4b8b6f4eb7183fb8dae923b7fc657f5cea71656c656d656e744964656e7469666965726b6167655f6f7665725f31386c6
56c656d656e7456616c7565f56a697373756572417574688443a10126a118218359024e3082024a308201f0a003020102021442 47c8
90625cba94ea6bf4b1cd6d0197836cbdcf300a06082a8648ce3d040302306d310b300906035504061302444b3113301106035504070
c0a4bc3b862656e6861766e3121301f060355040a0c184469676974616c69736572696e677373747972656c73656e310c300a060355
040b0c034b4541311830160603550403 0c0f444b5442204973737569 6e67204341301e170d323530363138313432333531 5a170d323
630363138313432333531 5a3074310b300906035504061302444b311330110603550407 0c0a4bc3b862656e6861766e3121301f0603
55040a0c184469676974616c69736572696e677373747972656c73656e310c300a060355040b0c034b4541311f301d06035504030c1
6444b5442204372656465 6e7469616c204973737565723059301306072a8648ce3d020106082a8648ce3d030107034200 4a470559f
a910bb964a9f3485acf2cb7c75c45fa6715a585f112e4c51bb0e3313de144cf4ccf3902ef254dbbf0cfa6047709175654cb6f78a98b
3b0f63da7081fa3673065300e0603551d0f0101ff040403020780301f0603551d23041830168014a22955e9f54a54369a6ede5b10d7
b010e096fde2301d0603551d0e0416041430528004222 5ec41f3766dc2f773fb8a1549f2b430130603551d25040c300a06082b06010
505070303300a06082a8648ce3d040302034800304502 2100cfa706ca2b600c741805f9fcee45b65b0e2c27225a8133f688496a61ce
d0c9ac0220152a4161cf345471fe9a856095e4550c5ea339555af4e538ffcc9c3de75392b059024630820242308201e9a0030201020
2145321e04f4daf5b6b608628b8836a019762d5684a300a06082a8648ce3d040302306f310b300906035504061302444b3113301106
035504070c0a4bc3b862656e6861766e3121301f060355040a0c184469676974616c69736572696e677373747972656c73656e310c3
00a060355040b0c034b4541311a301806035504030c11444b5442205465737420526f6f74204341301e170d323530363133230363330
34325a170d323630363132303633303432 5a306f310b300906035504061302444b3113301106035504070c0a4bc3b862656e6861766
e3121301f060355040a0c184469676974616c69736572696e677373747972656c73656e310c300a060355040b0c034b4541311a3018
06035504030c11444b5442205465737420526f6f74204341305930130 6072a8648ce3d020106082a8648ce3d030107034200 4f57e1
```

```
2b9db5b93329dfc0f645f023ca9d8df2ee4d3dc3eeef84d2b32c548d6c66dfd857b82d799f8faf8d40f491b4a77207374a1b7966eaf
246ec32d81dee047a3633061300f0603551d130101ff040530030101ff300e0603551d0f0101ff040403020106301f0603551d23041
830168014408e3bd80e22195eb40b728facfafcd9e171fa32301d0603551d0e04160414408e3bd80e22195eb40b728facfafcd9e171
fa32300a06082a8648ce3d040302034700304402206 03fcb50dbc50cd9fcfdfb51df532db3d279f480315b7732fe2113d73b07f9760
220229ec8f016b9e602829bb3f893fb2752561b4b72f625aa6c4886946958acc1e9590245308202413082 01e7a0030201020214531b
a3f8817761867bbf26ef36910197792134ee300a06082a8648ce3d040302306f310b3009060355040 61302444b31133011060355040
70c0a4bc3b862656e6861766e3121301f060355040a0c184469676974616c69736572696e677373747972656c73656e310c300a0603
55040b0c034b4541311a301806035504030c11444b5442205465737420526f6f74204341301e170d3235303631363134323530385a1
70d3236303631363134323530385a306d310b3009060355040 61302444b3113301106035504070c0a4bc3b862656e6861766e312130
1f060355040a0c184469676974616c69736572696e677373747972656c73656e310c300a060355040b0c034b4541311830160603550
4030c0f444b54422049737375696e67204341305930130 6072a8648ce3d020106082a8648ce3d030107034200043a815d5d9bde19bd
721d837c78e74356b5be928a8d637ccc469e1571d2c16bc2a5cd26c4ed079b4fd5b20c4a5f044f0fa216ee5a2842c8b471ea8a60cbc
309f0a3633061300f0603551d130101ff040530030101ff300e0603551d0f0101ff040403020106301f0603551d2304183016801440
8e3bd80e22195eb40b728facfafcd9e171fa32301d0603551d0e04160414a22955e9f54a54369a6ede5b10d7b010e096fde2300a060
82a8648ce3d040302034800304502204 1a5efff888692d5c85c7b66a27b16481d52011eba7d382aa8a75af98a7abd12022100c20b0b
e257823425790950a12b267d1e88804f4a013432a6e6e6c5f5cb5bf76d5901e6d8185901e1a667766572736 96f6e63312e306f64696
7657374416c676f726974686d675348412d3235366c76616c756544696765737473a17165752e6575726f70612e65632e61762e31a5
015820bfd749e3c152e2b338ee9b6798f807dfa1ff29c5974dfaad565f8108cd7dcf4c0258205283db09f7d7d80e77e3fb3efc9c287
6152790a94cd548963dbec00b16f79b3d035820835b6d563fbc94f73722ae1e178c5edfde07af938987eb0e046eb33774dab8f60458
2024cd3bd931debf1d2d0fef9c5fcbaf3a013462e7742799941b4f1c4f88ef0fd5055820a62166eeb63768dee24f28684078adfae8b
e7657edab487cce17db718e66fd026d6465766963654b6579496e666fa1696465766963654b6579a501020326200121582028 44eb7d
3b6a7c6021cc58b5cda40841d8fcfcc25a5242bb3e3808c941263e20225820885d6d33c60fbe6a01929ab367319169f3860bd69b4fd
6cd6a7fda765595c6ca67646f63547970657165752e6575726f70612e65632e61762e316c76616c6964697479496e666fa366736967
6e6564c074323032352d31302d32325430393a34353a31395a6976616c696446726f6dc074323032352d31302d32325430393a34353
a31395a6a76616c6964556e74696cc074323032362d30312d32305431303a34353a31395a5840ce09b37fd0c7c4b0396dbc0267d7b4
2a85e297f71e54b6e0deda97beb44eb28de98894f4fb9f892cfc27689d06b0a418d8fed53e6f616ec595bc10dbac18f16c6c6465766
963655369676e6564a26a6e616d65537061636573d81841a06a6465766963654 1757468a16f6465766963655369676e6174757265d2
8443a10126a0f658402ac229f92e0591086e07ff3715139928dd2b33fdfde8bed0d83ad24060e0fe6d942d4af49032c26bd56c9d113
2460125feab162946377528274017645939cb70'
}
```

The value *d* represents this CBOR *Document:*

```
{
  "docType": "eu.europa.ec.av.1",
  "issuerSigned": {
    "nameSpaces": {
      "eu.europa.ec.av.1": [
        24(<<
          {
            "digestID": 3,
            "random": h'ddbe55707848b9b1b624cd2a88b12d4b8b6f4eb7183fb8dae923b7fc657f5cea',
            "elementIdentifier": "age_over_18",
            "elementValue": true
          }
        >>)
      ]
    },
```

```
    "issuerAuth": [// COSE signature
      h'a10126', // COSE signature protected header:
                 // CBOR encoding of {1: -7}, i.e. alg_id = ECDSA with SHA-256 (ES256)
      {          // COSE signature unprotected header:
        33:      // COSE header 33 is X.509 certificate chain,
                 // these are the three certificates described in section 9.3.
             [ h'3082024a308201f0...', h'30820242308201e9...', h'30820241308201e7...']
      },
   //Object below is the signature payload, i.e. the Mobile Security Object
h'd8185901e1a66776657273696f6e63312e306f646967657374416c676f726974686d675348412d3235366c76616c756544696765737473a17165752e6575726f70612e65632e61762e31a5015820bfd749e3c152e2b338ee9b6798f807dfa1ff29c5974dfaad565f8108cd7dcf4c0258205283db09f7d7d80e77e3fb3efc9c2876152790a94cd548963dbec00b16f79b3d035820835b6d563fbc94f73722ae1e178c5edfde07af938987eb0e046eb33774dab8f604582024cd3bd931debf1d2d0fef9c5fcbaf3a013462e7742799941b4f1c4f88ef0fd5055820a62166eeb63768dee24f28684078adfae8be7657edab487cce17db718e66fd026d6465766963654b6579496e666fa1696465766963654b6579a501020326200121582028 44eb7d3b6a7c6021cc58b5cda40841d8fcfcc25a5242bb3e3808c941263e202258200885d6d33c60fbe6a01929ab367319169f3860bd69b4fd6cd6a7fda765595c6ca67646f63547970657165752e6575726f70612e65632e61762e316c76616c6964697479496e666fa3667369676e6564c074323032352d31302d32325430393a34353a31395a6976616c696446726f6dc074323032352d31302d32325430393a34353a31395a6a76616c6964556e74696cc074323032362d30312d32305431303a34353a31395a',
     // And finally, the COSE binary signature value, formed using the issuer private key:
h'ce09b37fd0c7c4b0396dbc0267d7b42a85e297f71e54b6e0deda97beb44eb28de98894f4fb9f892cfc27689d06b0a418d8fed53e6f616ec595bc10dbac18f16c'
    ]
  },
  "deviceSigned": {
    "nameSpaces": 24(<<{}>>),// empty map: all disclosed attributes are included in issuerSigned
    "deviceAuth": {
      "deviceSignature": 18(/ COSE_Sign1 / [
        / protected / <<{ 1: -7 }>>,   // alg_id = ECDSA with SHA-256 (ES256)
        / unprotected / { },           // no header elements in unprotected header
        null,                          // Payload not included – 'detached' signature
h'2ac229f92e0591086e07ff3715139928dd2b33fdfde8bed0d83ad24060e0fe6d942d4af49032c26bd56c9d1132460125feab16294637752827401764593 9cb70' // ECDSA Signature, P-256, 64 bytes
      ])
    }
  }
}
```

The included Mobile Security Object (h'd818...') is:

```
24(<<
  {
    "version": "1.0",
    "digestAlgorithm": "SHA-256",
    "valueDigests": {
      "eu.europa.ec.av.1": {
        1: h'bfd749e3c152e2b338ee9b6798f807dfa1ff29c5974dfaad565f8108cd7dcf4c',
```

```
        2: h'5283db09f7d7d80e77e3fb3efc9c2876152790a94cd548963dbec00b16f79b3d',
        3: h'835b6d563fbc94f73722ae1e178c5edfde07af938987eb0e046eb33774dab8f6',
        4: h'24cd3bd931debf1d2d0fef9c5fcbaf3a013462e7742799941b4f1c4f88ef0fd5',
        5: h'a62166eeb63768dee24f28684078adfae8be7657edab487cce17db718e66fd02'
      }
    },
    "deviceKeyInfo": {
      "deviceKey": {  // the wallet key, that this PoA is bound to
        1: 2,         // kty = EC2 – elliptic curve key types with x- and y-coordinate pair
        3: -7,        // alg = ECDSA w/ SHA-256
        -1: 1,        // crv = NIST P-256 also known as secp256r1
        -2: h'2844eb7d3b6a7c6021cc58b5cda40841d8fcfcc25a5242bb3e3808c941263e20', // x
        -3: h'885d6d33c60fbe6a01929ab367319169f3860bd69b4fd6cd6a7fda765595c6ca'  // y
        // (x, y) is a point on the curve that constitutes the public key.
      }
    },
    "docType": "eu.europa.ec.av.1",
    "validityInfo": {
      "signed": 0("2025-10-22T09:45:19Z"),
      "validFrom": 0("2025-10-22T09:45:19Z"),
      "validUntil": 0("2026-01-20T10:45:19Z")
    }
  }
>>)
```

Here the five digests represent five different age_over_NN attributes, that can be disclosed for this PoA. As can be seen from IssuerSigned, digest with id = 3 refers to the age_over_18 attribute, which is the only age_over_NN attribute revealed in this QR-code.

The COSE signature in IssuerSigned was created when the PoA was issued by the Issuer service (DIGST). The signature can be verified with the public key from the PoA Issuer's certificate – see section 9.3. The validity of this signature proves, that no data in Mobile Security Object was changed or tampered with.

7.2 Device signature validation

The device signature can be verified with the public key included in the Mobile Security Object of the IssuerSigned object (as DeviceKey). This key is bound to the device on which the DKTB app is installed.

To verify the device signature, as a verifier we first need to construct the SessionTranscript from *m, f,* and *t* from QRPayload and, in turn, DeviceAuthenticationBytes, the payload for the device signature.

Remembering the SessionTranscript form for Signed QR:

```
SessionTranscript = [
  DeviceEngagementBytes : bstr,             ; Always null for Signed QR presentations
  EReaderKeyBytes       : bstr,             ; Always null for Signed QR presentations
```

```
  Handover              : SignedQRHandover
]

SignedQRHandover = [
  mdocGeneratedNonce    : tstr,             ; 16 random bytes, Base64URL-encoded
  validFrom             : uint,             ; Unix timestamp the QR-code was generated at
  validTo               : uint              ; Unix timestamp the QR-code is no longer valid at
]
```

Using the values for *m*, *f*, and *t* (as provided in section 7.1) for mdocGeneratedNonce, validFrom, and validTo, respectively, the handover becomes:

```
SignedQRHandover = [
  "Qu3Mukt4wwh7vp8k7-KqQA",
  1761126319,
  1761126499
]
```

Using this, gives the following transcript:

```
SessionTranscript = [
  null,
  null,
  ["Qu3Mukt4wwh7vp8k7-KqQA", 1761126319, 1761126499 ]
]
```

CBOR encoding this array, and then Base64Url-encoding the output gives the following value:

```
g_b2g3ZRdTNNdWt0NHd3aDd2cDhrNy1LcVFBGmj4p68aaPioYw
```

This can be verified at [CBORZ].

Using the SessionTranscript constructed above and remembering DeviceAuthentication:

```
DeviceAuthentication = [
   "DeviceAuthentication",
   SessionTranscript,
   DocType,                ; Same as Document DocType, i.e. “eu.europa.ec.av.1”
   DeviceNameSpacesBytes   ; Always tagged bstr of empty map: #6.24(bstr .cbor {} )
]
```

We can now construct the CBOR representation of DeviceAuthentication as:

```
DeviceAuthentication = [
   "DeviceAuthentication",
   [ null, null, ["Qu3Mukt4wwh7vp8k7-KqQA", 1761126319, 1761126499 ]],
   "eu.europa.ec.av.1",
   #6.24(bstr .cbor {} )
]
```

Remember the meaning of #6.24(bstr .cbor {}): A tag (CBOR major type 6) with number 24 (meaning bstr) over CBOR encoding of an empty map.

We can now construct DeviceAuthenticationBytes as #6.24(bstr .cbor DeviceAuthentication), giving the value (Base64Url):

```
2BhYUYR0RGV2aWNlQXV0aGVudGljYXRpb26D9vaDdlF1M011a3Q0d3doN3ZwOGs3LUtxUUEaaPinrxpo-
KhjcWV1LmV1cm9wYS5lYy5hdi4x2BhBoA
```

Testing with [CBORZ] gives:

```
24(<<
  [
    "DeviceAuthentication",
    [
      null,
      null,
      [
        "Qu3Mukt4wwh7vp8k7-KqQA",
        1761126319,
        1761126499
      ]
    ],
    "eu.europa.ec.av.1",
    24(<<
      {
      }
    >>)
  ]
>>)
```

The device COSE signature can now be verified using the public key from MSO DeviceKeyInfo and the constructed DeviceAuthenticationBytes as payload.

7.3 Attribute validation

In practice, validate disclosed mdoc attributes, e.g. the age_over_18 attribute as in the example above, by doing the following for each disclosed attribute, i.e. IssuerSignedItemBytes in the IssuerNameSpaces map:

1. Use the digestID as index to lookup the corresponding signed digest value in the ValueDigests map of the Mobile Security Object.
2. Compute the digest over IssuerSignedItemBytes using the hash algorithm given by digestAlgorithm, which is always SHA-256 for PoA.
3. Compare the digest from the MSO ValueDigests map with the computed digest (byte-array comparison).
4. If the two digests are not equal, reject the PoA.

It is recommended to compute digest directly over IssuerSignedItemBytes taken from the PoA, and not to parse the data, and attempt to reconstruct IssuerSignedItem from its components. If the latter approach is used, make sure to remember adding the tag, to use the correct ordering of map elements, and not to use a CBOR indefinite-length map before computing IssuerSignedItemBytes.

In the example, only age_over_18 is disclosed – with the value true. This is represented by the following part (IssuerSignedItem) of the CBOR Document:

```
24(<<
  {
    "digestID": 3,
    "random": h'ddbe55707848b9b1b624cd2a88b12d4b8b6f4eb7183fb8dae923b7fc657f5cea',
    "elementIdentifier": "age_over_18",
```

```
  "elementValue": true
}>>)
```

The corresponding IssuerSignedItemBytes, which can be obtained by Base64Url-encoding the CBOR encoding of syntax above, are:

```
2BhYYKRoZGlnZXN0SUQDZnJhbmRvbVgg3b5VcHhIubG2JM0qiLEtS4tvTrcYP7ja6SO3_GV_XOpxZWxlbWVudElkZW50aWZpZXJrYWdlX29
2ZXJfMThsZWxlbWVudFZhbHVl9Q
```

Computing the digest, by calculating the SHA-256 hash over the Base64Url-encoded string, yields the following hex value:

```
835b6d563ffc94f73722ae1e178c5edfde07af938987fb0e046eb33774dab8f6
```

We see that this value matches that of the sixth (as denoted by the digestID) entry in the ValueDigests map of the MSO, thus confirming the validity of the age_over_18 attribute.

8 Appendix D: Guidelines for validating Authorization Response in OID4VP

When the response is received, verifier must handle error responses (according to [OID4VP] section 8.5).

Successful responses contain the vp_token parameter which must be validated as follows, before user age is verified.

A. Validate that the http POST message contains *vp_token* and *apu*[4] parameters as of [AVP].
B. Construct *DeviceResponse* (see Appendix E, section 9, for an example) by Base64Url decoding vp_token and CBOR parsing the resulting byte stream.
C. Validate that DeviceResponse parses into CBOR map with members *version, documents*, and *status.*
D. Validate that *status* equals 0 (OK) and that *documents* array contains exactly one *Document.*
E. Derive mdocGeneratedNonce string as UTF8(Base64URLDecode(*apu*)).
F. Extract the x5chain from *Document/IssuerSigned/IssuerAuth* unprotected header and validate the certificate chain (see section 9.3). Validate that the last certificate in the chain is a self-signed certificate present in the issuer trust list. Validate that the chain is ordered, and that the issuer certificate is first certificate in the chain.
G. Validate the issuer signature (*IssuerAuth)* using the public key from the issuer certificate from G.
H. Calculate the digest value for every IssuerSignedItem returned in the *Document/IssuerNameSpaces* structure and verify that these calculated digests equal the corresponding digest values in the signed MSO (*DigestIDs*) (see section 7.3 for more information).
I. Verify that the DocType in the MSO and DocType in the *Document* structure both equal "eu.europa.ec.av.1".
J. Validate the elements in the *ValidityInfo* structure, i.e. verify:
    a. that the 'signed' date is within the validity period of the certificate in the MSO header and
    b. that the '*validFrom*' element shall be earlier than or equal to the current timestamp, and
    c. that the '*validUntil*' element shall be equal to or later than the current timestamp.
K. Validate the device signature (*DeviceAuth)* as follows:
    a. Construct the *SessionTranscript* as described in Appendix A: Data structures (section 5.6).
    b. Use *SessionTranscript* to construct *DeviceAuthentication* as described in Appendix A (section 5.4).
    c. Validate the *DeviceAuth* COSE_Sign1 signature using CBOR encoding of *DeviceAuthentication* as external payload and public key from MSO (*DeviceKeyInfo/DeviceKey).*
L. Validate that *mdocGeneratedNonce* is not replayed:
    a. Maintain a set of all presented *mdocGeneratedNonce* values, that are valid according to *validUntil.*
    b. Verify that *mdocGeneratedNonce* is not in this set.
    c. Add *mdocGeneratedNonce* to the set
M. Validate that the disclosed age_over_NN attribute(s) satisfy the age requirement of the application.

[4] Please note, that it has not yet been decided exactly how this parameter (holding the mdocGeneratedNonce) is transferred to the verifier.

9 Appendix E: OID4VP example

This section will give a detailed example of presenting PoA over the OID4VP protocol according to [AVP], using the sample verifier provided by the EU [AVV].

The following section is based on and aligned with the EU Age Verification Profile [AVP], ensuring consistency with the EU approach to age verification and the interoperability profile developed in this context. This includes specific requirements for the implementation of the OID4VP protocol. However, as the DKTB project evolves, additional functionalities and technical mechanisms supported by the OID4VP protocol may be introduced to enhance functionality and support future developments within the DKTB ecosystem as well as the broader EUDI Wallet ecosystem.

9.1 Authorization Request

A Verifier requests a proof that the user is over the age of 18 by making an Authorization Request as shown below. Specifically, the following requests the user to present the age_over_18 attribute from their PoA in the ISO mdoc format (as specified by the contents of the dcql_query). The Authorization Request should be presented to the User either as a (navigable) link or rendered as a QR-code that can be scanned by the User's device.

```
Authorization Request (Base64) :
YXY6Ly8/cmVzcG9uc2VfdHlwZT12cF90b2tlbiZyZXNwb25zZV9tb2RlPWRpcmVjdF9wb3N0JmNsaWVudF9pZD1yZWRpcmVjdF91cmklM0F
odHRwcyUzQSUyRiUyRnZlcmlmaWVyLWJhY2tlbmQuYWdldmVyaWZpY2F0aW9uLmRldiUyRndhbGxldCUyRmRpcmVjdF9wb3N0JTJGSzdHdG
1jUXQ0Q25FbUN5dkVRZTVhT2lPWEg1TzZWSkl5R3VkbDRwTVFiMXhiSk5yVWNOZU1QZ2Q2c3ZNcnhURHlLQ3BLQzZXejd0WmgyMll4dkpSR
GcmcmVzcG9uc2VfdXJpPWh0dHBzJTNBJTJGJTJGdmVyaWZpZXItYmFja24uZGV2JTJGd2FsbGV0JTJGZGly
ZWN0X3Bvc3QlMkZLN0d0bWNRdDRDbkVtQ3l2RVFlNWFPaU9YSDVPNlZKSXlHdWRsNHBNUWIxeGJKTnJVY05lTVBnZDZzdk1yeFREeUtDcEt
DNld6N3RaaDIyWXh2SlJEZyZkY3FsX3F1ZXJ5PSU3QiUyMmNyZWRlbnRpYWxzJTIyJTNBJTVCJTdCJTIyaWQlMjIlM0ElMjJwcm9vZl9vZl
9hZ2UlMjIlMkMlMjJmb3JtYXQlMjIlM0ElMjJtc29fbWRvYyUyMiUyQyUyMm1ldGElMjIlM0ElN0IlMjJkb2N0eXBlX3ZhbHVlJTIyJTNBJ
TIyZXUuZXVyb3BhLmVjLmF2LjElMjIlN0QlMkMlMjJjbGFpbXMlMjIlM0ElNUIlN0IlMjJwYXRoJTIyJTNBJTVCJTIyZXUuZXVyb3BhLmVj
LmF2LjElMjIlMkMlMjJhZ2Vfb3Zlcl8xOCUyMiU1RCU3RCU1RCU3RCU1RCU3RCZub25jZT0wNDkzMjg4Yi00NzhlLTRhZWYtYmViMi1mNzE
5MzFmYzI2MDMmc3RhdGU9SzdHdG1jUXQ0Q25FbUN5dkVRZTVhT2lPWEg1TzZWSkl5R3VkbDRwTVFiMXhiSk5yVWNOZU1QZ2Q2c3ZNcnhUR
HlLQ3BLQzZXejd0WmgyMll4dkpSRGc=

Text (URL decoded and indented for readability):

av://
?response_type=vp_token
&response_mode=direct_post
&client_id=redirect_uri:https://verifier-backend.ageverification.dev/wallet/direct_post/K7GtmcQt4CnEm[...]
&response_uri=https://verifier-backend.ageverification.dev/wallet/direct_post/K7GtmcQt4CnEm[...]
&dcql_query={
  "credentials": [
    {
      "id": "proof_of_age",
      "format": "mso_mdoc",
      "meta": {
        "doctype_value": "eu.europa.ec.av.1"
      },
```

```
      "claims": [
        {
          "path": [
            "eu.europa.ec.av.1",
            "age_over_18"
          ]
        }
      ]
    }
  ]
}
&nonce=0493288b-478e-4aef-beb2-f71931fc2603
&state=K7GtmcQt4CnEmCyvEQe5aOiOXH5O6VJIyGudl4pMQb1xbJNrUcNeMPgd6svMrxTDyKCpKC6Wz7tZh22YxvJRDg
```

[AVP] dictates the values for response_type and response_mode. While the response type should always be “vp_token” when using OID4VP, options for response_mode other than “direct_post” may be supported soon.

[AVP] dictates the format of client_id and response_uri: Response_uri is the URI, where Wallet should POST its response, i.e. the requested proof. This URI also identifies the client by use of the [OID4VP] client identifier scheme *redirect_uri.* This scheme defines the client_id format "redirect_uri:<response_uri>". Additional options for client_id may be introduced at a later stage (see also [OID4VP section 5.9.3]).

The Verifier MUST include a nonce parameter in the Authorization Request. The nonce must be unique and have a high amount of entropy (e.g. 128 bit). The device signature includes the nonce, and it serves the purpose of binding the device response cryptographically to the request, thereby preventing a class of replay attacks.

The Verifier MAY optionally include a state parameter. If included, the response by DKTB to the response_uri must include the state parameter value. This allows the verifier to connect a response to a specific request/transaction. Note that in the example, the correlation is obtained by including an identifier ("K7Gtm...") in the response URI, which is a valid alternative, and means that the state parameter is not strictly necessary here.

9.2 Authorization Response

In response to the example Authorization Request above, DKTB responds with the following:

```
POST /wallet/direct_post/K7GtmcQt4CnEm[...] HTTP/1.1
Host: verifier-backend.ageverification.dev
Content-Type: application/x-www-form-urlencoded

  state=K7GtmcQt4CnEmCyvEQe5aOiOXH5O6VJIyGudl4pMQb1xbJNrUcNeMPgd6svMrxTDyKCpKC6Wz7tZh22YxvJRDg
  &vp_token=
o2d2ZXJzaW9uYzEuMGlkb2N1bWVudHOBo2dkb2NUeXBlcWV1LmV1cm9wYS5lYy5hdi4xbGlzc3VlclNpZ25lZKJqbmFtZVNwYWNlc6FxZXU
uZXVyb3BhLmVjLmF2LjGB2BhYT6RoZGlnZXN0SUQDZnJhbmRvbVDK2UtqWNmsIZCHp7vSaLItcWVsZW1lbnRJZGVudGlmaWVya2FnZV9vdm
VyXzE4bGVsZW1lbnRWYWx1ZfVqaXNzdWVyQXV0aIRDoQEmoRghg1kCTjCCAkowggHwoAMCAQICFEJHyJBiXLqU6mv0sc1tAZeDbL3PMAoGC
CqGSM49BAMCMG0xCzAJBgNVBAYTAkRLMRMwEQYDVQQHDApLw7hiZW5oYXZuMSEwHwYDVQQKDBhEaWdpdGFsaXNlcmluZ3NzdHlyZWxzZW4x
DDAKBgNVBAsMA0tFQTEYMBYGA1UEAwwPREtUQiBJc3N1aW5nIENBMB4XDTI1MDYxODE0MjM1MVoXDTI2MDYxODE0MjM1MVowdDELMAkGA1U
EBhMCREsxEzARBgNVBAcMCkvDuGJlbmhhdm4xITAfBgNVBAoMGERpZ2l0YWxpc2VyaW5nc3N0eXJlbHNlbjEMMAoGA1UECwwDS0VBMR8wHQ
YDVQQDDBZES1RCIENyZWRlbnRpYWwgSXNzdWVyMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEpHBVn6kQu5ZKnzSFrPLLfHXEX6ZxWlhfE
```

```
S5MUbsOMxPeFEz0zPOQLvJU278M-
mBHcJF1ZUy294qYs7D2PacIH6NnMGUwDgYDVR0PAQH_BAQDAgeAMB8GA1UdIwQYMBaAFKIpVen1SlQ2mm7eWxDXsBDglv3iMB0GA1UdDgQW
BBQwUoAEIiXsQfN2bcL3c_uKFUnytDATBgNVHSUEDDAKBggrBgEFBQcDAzAKBggqhkjOPQQDAgNIADBFAiEAz6cGyitgDHQYBfn87kW2Ww4
sJyJagTP2iElqYc7QyawCIBUqQWHPNFRx_pqFYJXkVQxeozlVWvTlOP_MnD3nU5KwWQJFMIICQTCCAeegAwIBAgIUUxuj-
IF3YYZ7vybvNpEBl3khNO4wCgYIKoZIzj0EAwIwbzELMAkGA1UEBhMCREsxEzARBgNVBAcMCkvDuGJlbmhhdm4xITAfBgNVBAoMGERpZ2l0
YWxpc2VyaW5nc3N0eXJlbHNlbjEMMAoGA1UECwwDS0VBMRowGAYDVQQDDBFES1RCIFRlc3QgUm9vdCBDQTAeFw0yNTA2MTYxNDI1MDhaFw0
yNjA2MTYxNDI1MDhaMG0xCzAJBgNVBAYTAkRLMRMwEQYDVQQHDApLw7hiZW5oYXZuMSEwHwYDVQQKDBhEaWdpdGFsaXNlcmluZ3NzdHlyZW
xzZW4xDDAKBgNVBAsMA0tFQTEYMBYGA1UEAwwPREtUQiBJc3N1aW5nIENBMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEOoFdXZveGb1yH
YN8eOdDVrW-
koqNY3zMRp4VcdLBa8KlzSbE7QebT9WyDEpfBE8PohbuWihCyLRx6opgy8MJ8KNjMGEwDwYDVR0TAQH_BAUwAwEB_zAOBgNVHQ8BAf8EBAM
CAQYwHwYDVR0jBBgwFoAUQI472A4iGV60C3KPrPr82eFx-
jIwHQYDVR0OBBYEFKIpVen1SlQ2mm7eWxDXsBDglv3iMAoGCCqGSM49BAMCA0gAMEUCIEGl7_-
IhpLVyFx7ZqJ7FkgdUgEeun04KqinWvmKer0SAiEAwgsL4leCNCV5CVChKyZ9HoiAT0oBNDKm5ubF9ctb921ZAkYwggJCMIIB6aADAgECAh
RTIeBPTa9ba2CGKLiDagGXYtVoSjAKBggqhkjOPQQDAjBvMQswCQYDVQQGEwJESzETMBEGA1UEBwwKS8O4YmVuaGF2bjEhMB8GA1UECgwYR
GlnaXRhbGlzZXJpbmdzc3R5cmVsc2VuMQwwCgYDVQQLDANLRUExGjAYBgNVBAMMEURLVEIgVGVzdCBSb290IENBMB4XDTI1MDYxMjA2MzA0
MloXDTI2MDYxMjA2MzA0MlowbzELMAkGA1UEBhMCREsxEzARBgNVBAcMCkvDuGJlbmhhdm4xITAfBgNVBAoMGERpZ2l0YWxpc2VyaW5nc3N
0eXJlbHNlbjEMMAoGA1UECwwDS0VBMRowGAYDVQQDDBFES1RCIFRlc3QgUm9vdCBDQTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABPV-
ErnbW5MynfwPZF8CPKnY3y7k09w-7vhNKzLFSNbGbf2Fe4LXmfj6-
NQPSRtKdyBzdKG3lm6vJG7DLYHe4EejYzBhMA8GA1UdEwEB_wQFMAMBAf8wDgYDVR0PAQH_BAQDAgEGMB8GA1UdIwQYMBaAFECOO9gOIhle
tAtyj6z6_NnhcfoyMB0GA1UdDgQWBBRAjjvYDiIZXrQLco-s-vzZ4XH6MjAKBggqhkjOPQQDAgNHADBEAiBgP8tQ28UM2fz9-
1HfUy2z0nn0gDFbdzL-
IRPXOwf5dgIgIp7I8Ba55gKCm7P4k_snUlYbS3L2JapsSIaUaViswelZAoDYGFkCe6ZndmVyc2lvbmMxLjBvZGlnZXN0QWxnb3JpdGhtZ1N
IQS0yNTZsdmFsdWVEaWdlc3RzoXFldS5ldXJvcGEuZWMuYXYuMagBWCCq-f-
3pLGKTIJp0WW9zYkVv1BTZ7IVp7XqVrLh53qKigJYIFHEjn8Mketf_dFpqZvsxqfH9yljIvdZlrMQQr6UEP50A1ggGCp5kZ3JGdh1LWc61z
zbmCs5DyY6NhXKsQeXkr9neYUEWCA7f9uu0DTv4_KlnIGBCy5RkRb6vmoWLbNMti8mWXsPgwVYIGnCnQPYugLLWVYXk594WSuuYgwID5TC7
Zfgrwo8xM2rBlgg3qJAjZ7ldLkiYcQaMkoBZnzPL4-WP4cxsn3f3vyyW4IHWCB4uTESOdG3NBu-
7ZGzGzb43sAK8lvi5MG9cAfg082lKAhYIAillhWRqiwAnokCSkZWqZmf5inIk1zLXE6JN52oMDbRbWRldmljZUtleUluZm-
iaWRldmljZUtleaUBAiABIVggIKMqXkY7xF41JnZUNHGU8002aMPXgFa4LolnulNONiAiWCCuoSJiMvompE90YkDHasR5XWAZkjJgj5EmTc
cnMG0tOAMmcWtleUF1dGhvcml6YXRpb25zoWpuYW1lU3BhY2VzgXFldS5ldXJvcGEuZWMuYXYuMWdkb2NUeXBlcWV1LmV1cm9wYS5lYy5hd
i4xbHZhbGlkaXR5SW5mb6Nmc2lnbmVkwHQyMDI1LTA5LTE2VDE4OjI2OjUxWml2YWxpZEZyb23AdDIwMjUtMDktMTZUMTg6MjY6NTFaanZh
bGlkVW50aWzAdDIwMjUtMDktMTdUMTg6MjY6NTFaWEDybeXImk8FL1vXHScs2Zfx_GMp_il4nD0LtSbpL6wI7fQ47iiRbCfL-
YImk9PLNI1UQiU9Hawd8Gr8At8yzs8_bGRldmljZVNpZ25lZKJqbmFtZVNwYWNlc9gYQaBqZGV2aWNlQXV0aKFvZGV2aWNlU2lnbmF0dXJl
hEOhASag9lhAALv6kWLdbQNjy3Sy9WXKlO6pPVafW29TmUFO5URNk9jqRzyx0TSPxRqG5OD-
Js3v_hObObtwDiYmDpZJLviD1WZzdGF0dXMA
```

The VP Token, which contains the requested PoA, is a Base64Url-encoding of a mdoc DeviceResponse (as CBOR). The contents are shown in CBOR annotation below.

```
DeviceResponse = {
   "version" : tstr,                    ; Version, always "1.0" for PoA
 ? "documents" : [+Document],           ; Always exactly one document for PoA
 ? "documentErrors": [+DocumentError]   ; Never used for [AVP]
   "status" : uint                      ; Status code
}
```

```
{
  "version": "1.0",
  "documents": [
```

```
{
  "docType": "eu.europa.ec.av.1",
  "issuerSigned": {
    "nameSpaces": {
      "eu.europa.ec.av.1": [
        24(<<
          {
            "digestID": 3,
            "random": h'cad94b6a58d9ac219087a7bbd268b22d',
            "elementIdentifier": "age_over_18",
            "elementValue": true
          }
        >>)
      ]
    },
    "issuerAuth": [ //COSE_Sign1 signature
      h'a10126',
      {
        33: [  //X.509 certificate chain
h'3082024a308201f0a00302010202144247c890625cba94ea6bf4b1cd6d0197836cbdcf300a06082a8648ce3d040302306d310b300
906035504061302444b311330110603550407 0c0a4bc3b862656e6861766e3121301f060355040a0c184469676974616c6973657269
6e677373747972656c73656e310c300a060355040b0c034b4541311830160603550403 0c0f444b54422049737375696e67204341301
e170d3235303631383134323335315a170d3236303631383134323335315a3074310b300906035504061302444b3113301106035504
070c0a4bc3b862656e6861766e3121301f060355040a0c184469676974616c69736572696e677373747972656c73656e310c300a060
355040b0c034b4541311f301d06035504030c16444b54422043726564656e7469616c204973737565723059301306072a8648ce3d02
0106082a8648ce3d03010703420004a470559fa910bb964a9f3485acf2cb7c75c45fa6715a585f112e4c51bb0e3313de144cf4ccf39
02ef254dbbf0cfa6047709175654cb6f78a98b3b0f63da7081fa3673065300e0603551d0f0101ff040403020780301f0603551d2304
1830168014a22955e9f54a54369a6ede5b10d7b010e096fde2301d0603551d0e041604143052800422 25ec41f3766dc2f773fb8a154
9f2b430130603551d25040c300a06082b06010505070303300a06082a8648ce3d040302034800304502210 0cfa706ca2b600c741805
f9fcee45b65b0e2c27225a8133f688496a61ced0c9ac0220152a4161cf345471fe9a856095e4550c5ea339555af4e538ffcc9c3de75
392b0',
h'30820241308201e7a0030201020214531ba3f8817761867bbf26ef36910197792134ee300a06082a8648ce3d040302306f310b300
906035504061302444b311330110603550407 0c0a4bc3b862656e6861766e3121301f060355040a0c184469676974616c6973657269
6e677373747972656c73656e310c300a060355040b0c034b4541311a301806035504030c11444b54422054657374205 26f6f7420434
1301e170d3235303631363134323530385a170d3236303631363134323530385a306d310b300906035504061302444b311330110603
5504070c0a4bc3b862656e6861766e3121301f060355040a0c184469676974616c69736572696e677373747972656c73656e310c300
a060355040b0c034b4541311830160603550403 0c0f444b54422049737375696e67204341305930130 6072a8648ce3d020106082a86
48ce3d030107034200043a815d5d9bde19bd721d837c78e74356b5be928a8d637ccc469e1571d2c16bc2a5cd26c4ed079b4fd5b20c4
a5f044f0fa216ee5a2842c8b471ea8a60cbc309f0a3633061300f0603551d130101ff040530030101ff300e0603551d0f0101ff0404
03020106301f0603551d23041830168014408e3bd80e22195eb40b728facfafcd9e171fa32301d0603551d0e04160414a22955e9f54
a54369a6ede5b10d7b010e096fde2300a06082a8648ce3d040302034800304502204 1a5efff888692d5c85c7b66a27b16481d52011e
ba7d382aa8a75af98a7abd12022100c20b0be257823425790950a12b267d1e88804f4a013432a6e6e6c5f5cb5bf76d',
h'30820242308201e9a00302010202145321e04f4daf5b6b608628b8836a019762d5684a300a06082a8648ce3d040302306f310b300
906035504061302444b311330110603550407 0c0a4bc3b862656e6861766e3121301f060355040a0c184469676974616c6973657269
6e677373747972656c73656e310c300a060355040b0c034b4541311a301806035504030c11444b54422054657374205 26f6f7420434
1301e170d3235303631323036333034325a170d3236303631323036333034325a306f310b300906035504061302444b311330110603
5504070c0a4bc3b862656e6861766e3121301f060355040a0c184469676974616c69736572696e677373747972656c73656e310c300
```

```
a060355040b0c034b4541311a301806035504030c11444b5442205465737420526f6f74204341305930130607 2a8648ce3d02010608
2a8648ce3d03010703420004f57e12b9db5b93329dfc0f645f023ca9d8df2ee4d3dc3eeef84d2b32c548d6c66dfd857b82d799f8faf
8d40f491b4a77207374a1b7966eaf246ec32d81dee047a3633061300f0603551d130101ff040530030101ff300e0603551d0f0101ff
040403020106301f0603551d23041830168014408e3bd80e22195eb40b728facfafcd9e171fa32301d0603551d0e04160414408e3bd
80e22195eb40b728facfafcd9e171fa32300a06082a8648ce3d0403020347003044022060 3fcb50dbc50cd9fcfdfb51df532db3d279
f480315b7732fe2113d73b07f9760220229ec8f016b9e602829bb3f893fb2752561b4b72f625aa6c4886946958acc1e9'f
]f          }, // Payload, i.e. Mobile Security Object
h'd81859027ba66776657273696f6e63312e306f646967657374416c676f726974686d675348412d3235366c76616c7565446967657
37473a17165752e6575726f70612e65632e61762e31a8015820aaf9ffb7a4b18a4c8269d165bdcd8915bf505367b215a7b5ea56b2e1
e77a8a8a02582051c48e7f0c91eb5ffdd169a99becc6a7c7f7296322f75996b31042be9410fe74035820182a79919dc919d8752d673
ad73cdb982b390f263a3615cab1079792bf6779850458203b7fdbaed034efe3f2a59c81810b2e519116fabe6a162db34cb62f26597b
0f8305582069c29d03d8ba02cb595617939f78592bae620c080f94c2ed97e0af0a3cc4cdab065820dea2408d9ee574b92261c41a324
a01667ccf2f8f963f8731b27ddfdefcb25b8207582078b9311239d1b7341bbeed91b31b36f8dec00af25be2e4c1bd7007e0d3cda528
08582008a5961591aa2c009e89024a4656a9999fe629c8935ccb5c4e89379da83036d16d6465766963654b6579496e666fa26964657
66963654b6579a50102200121582020a32a5e463bc45e35267654347194f34d3668c3d78056b82e8967ba534e3620225820aea12262
32fa26a44f746240c76ac4795d60199232608f91264dc727306d2d380326716b6579417574686f72697a6174696f6e73a16a6e616d6
5537061636573817165752e6575726f70612e65632e61762e3167646f63547970657165752e6575726f70612e65632e61762e316c76
616c6964697479496e666fa3667369676e6564c074323032352d30392d31365431383a32363a35315a6976616c696446726f6dc0743
23032352d30392d31365431383a32363a35315a6a76616c6964556e74696cc074323032352d30392d31375431383a32363a35315a',
6h'f26de5c89a4f052f5bd71d272cd997f1fc6329fe29789c3d0bb526e92fac08edf438ee28916c27cbf9822693d3cb348d5442253d
1dac1df06afc02df32cecf3f' // Signature6        ]
      },
      "deviceSigned": {
        "nameSpaces": 24(<< { } >>),
        "deviceAuth": {
          "deviceSignature": [
            h'a10126', // Protected header: CBOR encoding of {1: -7}, i.e. alg_id = ES256
            { },       // Unprotected header, empty
            null,
h'00bbfa9162dd6d0363cb74b2f565ca94eea93d569f5b6f5399414ee5444d93d8ea473cb1d1348fc51a86e4e0fe26cdeffe139b39b
b700e26260e96492ef883d5'
          ]
        }
      }
    }
  ],
  "status": 0 //device response status: OK
}
```

9.3 Issuer Auth certificate chain and issuer signature validation

The included certificate chain contains X.509 certificates issued to:

1. C=DK, L=København, O=Digitaliseringsstyrelsen, OU=KEA, CN=DKTB Credential Issuer
2. C=DK, L=København, O=Digitaliseringsstyrelsen, OU=KEA, CN=DKTB Issuing CA
3. C=DK, L=København, O=Digitaliseringsstyrelsen, OU=KEA, CN=DKTB Test Root CA

The certificates are included below.

CN= DKTB Credential Issuer:

```
-----BEGIN CERTIFICATE-----
MIICSjCCAfCgAwIBAgIUQkfIkGJcupTqa/SxzW0Bl4Nsvc8wCgYIKoZIzj0EAwIw
bTELMAkGA1UEBhMCREsxEzARBgNVBAcMCkvDuGJlbmhhdm4xITAfBgNVBAoMGERp
Z2l0YWxpc2VyaW5nc3N0eXJlbHNlbjEMMAoGA1UECwwDS0VBMRgwFgYDVQQDDA9E
S1RCIElzc3VpbmcgQ0EwHhcNMjUwNjE4MTQyMzUxWhcNMjYwNjE4MTQyMzUxWjB0
MQswCQYDVQQGEwJESzETMBEGA1UEBwwKS8O4YmVuaGF2bjEhMB8GA1UECgwYRGln
aXRhbGlzZXJpbmdzc3R5cmVsc2VuMQwwCgYDVQQLDANLRUExHzAdBgNVBAMMFkRL
VEIgQ3JlZGVudGlhbCBJc3N1ZXIwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAASk
cFWfqRC7lkqfNIWs8st8dcRfpnFaWF8RLkxRuw4zE94UTPTM85Au8lTbvwz6YEdw
kXVlTLb3ipizsPY9pwgfo2cwZTAOBgNVHQ8BAf8EBAMCB4AwHwYDVR0jBBgwFoAU
oilV6fVKVDaabt5bENewEOCW/eIwHQYDVR0OBBYEFDBSgAQiJexB83Ztwvdz+4oV
SfK0MBMGA1UdJQQMMAoGCCsGAQUFBwMDMAoGCCqGSM49BAMCA0gAMEUCIQDPpwbK
K2AMdBgF+fzuRbZbDiwnIlqBM/aISWphztDJrAIgFSpBYc80VHH+moVgleRVDF6j
OVVa9OU4/8ycPedTkrA=
-----END CERTIFICATE-----
```

CN= DKTB Issuing CA:

```
-----BEGIN CERTIFICATE-----
MIICQTCCAeegAwIBAgIUUxuj+IF3YYZ7vybvNpEBl3khNO4wCgYIKoZIzj0EAwIw
bzELMAkGA1UEBhMCREsxEzARBgNVBAcMCkvDuGJlbmhhdm4xITAfBgNVBAoMGERp
Z2l0YWxpc2VyaW5nc3N0eXJlbHNlbjEMMAoGA1UECwwDS0VBMRowGAYDVQQDDBFE
S1RCIFRlc3QgUm9vdCBDQTAeFw0yNTA2MTYxNDI1MDhaFw0yNjA2MTYxNDI1MDha
MG0xCzAJBgNVBAYTAkRLMRMwEQYDVQQHDApLw7hiZW5oYXZuMSEwHwYDVQQKDBhE
aWdpdGFsaXNlcmluZ3NzdHlyZWxzZW4xDDAKBgNVBAsMA0tFQTEYMBYGA1UEAwwP
REtUQiBJc3N1aW5nIENBMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEOoFdXZve
Gb1yHYN8eOdDVrW+koqNY3zMRp4VcdLBa8KlzSbE7QebT9WyDEpfBE8PohbuWihC
yLRx6opgy8MJ8KNjMGEwDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMCAQYw
HwYDVR0jBBgwFoAUQI472A4iGV60C3KPrPr82eFx+jIwHQYDVR0OBBYEFKIpVen1
SlQ2mm7eWxDXsBDglv3iMAoGCCqGSM49BAMCA0gAMEUCIEGl7/+IhpLVyFx7ZqJ7
FkgdUgEeun04KqinWvmKer0SAiEAwgsL4leCNCV5CVChKyZ9HoiAT0oBNDKm5ubF
9ctb920=
-----END CERTIFICATE-----
```

CN= DKTB Test Root CA:

```
-----BEGIN CERTIFICATE-----
MIICQjCCAemgAwIBAgIUUyHgT02vW2tghii4g2oBl2LVaEowCgYIKoZIzj0EAwIw
bzELMAkGA1UEBhMCREsxEzARBgNVBAcMCkvDuGJlbmhhdm4xITAfBgNVBAoMGERp
Z2l0YWxpc2VyaW5nc3N0eXJlbHNlbjEMMAoGA1UECwwDS0VBMRowGAYDVQQDDBFE
S1RCIFRlc3QgUm9vdCBDQTAeFw0yNTA2MTIwNjMwNDJaFw0yNjA2MTIwNjMwNDJa
MG8xCzAJBgNVBAYTAkRLMRMwEQYDVQQHDApLw7hiZW5oYXZuMSEwHwYDVQQKDBhE
aWdpdGFsaXNlcmluZ3NzdHlyZWxzZW4xDDAKBgNVBAsMA0tFQTEaMBgGA1UEAwwR
```

```
REtUQiBUZXN0IFJvb3QgQ0EwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAAT1fhK5
21uTMp38D2RfAjyp2N8u5NPcPu74TSsyxUjWxm39hXuC15n4+vjUD0kbSncgc3Sh
t5ZuryRuwy2B3uBHo2MwYTAPBgNVHRMBAf8EBTADAQH/MA4GA1UdDwEB/wQEAwIB
BjAfBgNVHSMEGDAWgBRAjjvYDiIZXrQLco+s+vzZ4XH6MjAdBgNVHQ4EFgQUQI47
2A4iGV60C3KPrPr82eFx+jIwCgYIKoZIzj0EAwIDRwAwRAIgYD/LUNvFDNn8/ftR
31Mts9J59IAxW3cy/iET1zsH+XYCICKeyPAWueYCgpuz+JP7J1JWG0ty9iWqbEiG
lGlYrMHp
-----END CERTIFICATE-----
```

The signature (IssuerAuth) may be validated using the public key in the first of these certificates. Verifier must in addition validate the certificate chain, which e.g. entails verifying the signature on the certificates:

- Signature in "CN=DKTB Credential Issuer" should verify with public key from "CN=DKTB Issuing CA"
- Signature in "CN=DKTB DKTB Issuing CA" should verify with public key from "CN=DKTB Test Root CA"
- Signature in "CN=DKTB Test Root" should verify with public key from "CN=DKTB Test Root CA" (self-signed)

The Verifier MUST validate the certificate chain according to [RFC5280]. It is important that the X.509 system libraries on the specific platform is used for this, validation of certificate chains is a complex process.

10 References

[ARF]	Architecture Reference Framework for European Identity Wallets https://github.com/eu-digital-identity-wallet/eudi-doc-architecture-and-reference-framework
[AV]	European Age Verification Solution, Technical Specifications https://ageverification.dev/av-doc-technical-specification/docs/architecture-and-technical-specifications/
[AVP]	European Age Verification Solution, Age Verification Profile https://ageverification.dev/av-doc-technical-specification/docs/annexes/annex-A/annex-A-av-profile/
[AVV]	European Sample Age Verifier web application https://verifier.ageverification.dev/
[CBOR]	RFC 8949: Concise Binary Object Representation (CBOR) https://www.rfc-editor.org/rfc/rfc8949.html
[CBORZ]	CBOR zone website: https://cbor.zone/
[COSE]	RFC 8152 CBOR Object Signing and Encryption (COSE)
[ISO 18013-5]	ISO/IEC 18013-5, Personal identification, ISO-compliant driving license - Part 5: Mobile driving license (mDL) application. https://www.iso.org/standard/69084.html
[OID4VP]	OpenID for Verifiable Presentations https://openid.net/specs/openid-4-verifiable-presentations-1_0.html
[RFC5280]	RFC 5280 Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile