Workgroup: OpenID Digital Credentials Protocols
Published: 3 April 2025
Authors: O. Terbu *Mattr* | T. Lodderstedt *SPRIND* | K. Yasuda *SPRIND* | T. Looker *Mattr*

OpenID for Verifiable Presentations - draft 25

Abstract

This specification defines a protocol for requesting and presenting Verifiable Credentials.

Table of Contents

1. Introduction

This specification defines a mechanism on top of OAuth 2.0 [RFC6749] that enables presentation of Verifiable Credentials as Verifiable Presentations. Verifiable Credentials and Verifiable Presentations can be of any format, including, but not limited to W3C Verifiable Credentials Data Model [VC_DATA], ISO mdoc [ISO.18013-5], IETF SD-JWT VC [I-D.ietf-oauth-sd-jwt-vc], and AnonCreds [Hyperledger.Indy].

OAuth 2.0 [RFC6749] is used as a base protocol as it provides the required rails to build a simple, secure, and developer-friendly Credential presentation layer on top of it. Moreover, implementers can, in a single interface, support Credential presentation and the issuance of Access Tokens for access to APIs based on Verifiable Credentials in the Wallet. OpenID Connect [OpenID.Core] deployments can also extend their implementations using this specification with the ability to transport Verifiable Presentations.

This specification can also be combined with [SIOPv2], if implementers require OpenID Connect features, such as the issuance of Self-Issued ID Tokens [SIOPv2].

1.1. Requirements Notation and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Terminology

This specification uses the terms "Access Token", "Authorization Request", "Authorization Response", "Client", "Client Authentication", "Client Identifier", "Grant Type", "Response Type", "Token Request" and "Token Response" defined by OAuth 2.0 [RFC6749], the terms "End-User" and "Entity" as defined by OpenID Connect Core [OpenID.Core], the terms "Request Object" and "Request URI" as defined by [RFC9101], the term "JSON Web Token (JWT)" defined by JSON Web Token (JWT) [RFC7519], the term "JOSE Header" defined by JSON Web Signature (JWS) [RFC7515], the term "JSON Web Encryption (JWE)" defined by [RFC7516], and the term "Response Mode" defined by OAuth 2.0 Multiple Response Type Encoding Practices [OAuth.Responses].

Base64url-encoded denotes the URL-safe base64 encoding without padding defined in Section 2 of [RFC7515].

This specification also defines the following terms. In the case where a term has a definition that differs, the definition below is authoritative.

Biometrics-based Holder Binding: Ability of the Holder to prove legitimate possession of a Verifiable Credential by demonstrating a certain biometric trait, such as a fingerprint or face. One example of a Verifiable Credential with biometric Holder Binding is a mobile driving license [ISO.18013-5], which contains a portrait of the Holder.

Claims-based Holder Binding: Ability of the Holder to prove legitimate possession of a Verifiable Credential by proving certain claims, e.g., name and date of birth, for example by presenting another Verifiable Credential. Claims-based Holder Binding allows long-term, cross-device use of a Credential as it does not depend on cryptographic key material stored on a certain device. One example of such a Verifiable Credential could be a Diploma.

Credential: A set of one or more claims about a subject made by a Credential Issuer. Note that the definition of the term "Credential" in this specification is different from that in [OpenID.Core].

Credential Format Identifier: An identifier to denote a specific Credential Format in the context of this specification. This identifier implies the use of parameters specific to the respective Credential Format.

Credential Issuer: An entity that issues Verifiable Credentials. Also called Issuer.

Cryptographic Holder Binding: Ability of the Holder to prove legitimate possession of a Verifiable Credential by proving control over the same private key during the issuance and presentation. Mechanism might depend on the Credential Format. For example, in jwt_vc_json Credential Format, a Verifiable Credential with Cryptographic Holder Binding contains a public key or a reference to a public key that matches to the private key controlled by the Holder.

Digital Credentials API: The Digital Credentials API (DC API) refers to the W3C Digital Credentials API [W3C.Digital_Credentials_API] on the Web Platform and its equivalent native APIs on App Platforms (such as Credential Manager on Android).

Holder: An entity that receives Verifiable Credentials and has control over them to present them to the Verifiers as Verifiable Presentations.

Holder Binding: Ability of the Holder to prove legitimate possession of a Verifiable Credential.

Issuer-Holder-Verifier Model: A model for exchanging claims, where claims are issued in the form of Verifiable Credentials independent of the process of presenting them as Verifiable Presentation to the Verifiers. An issued Verifiable Credential can (but must not necessarily) be used multiple times.

Origin: An identifier for the calling website or native application, asserted by the web or app platform. A web origin is the combination of a scheme/protocol, host, and port, with port being omitted when it matches the default port of the scheme. An app platform may use a linked web origin, or use a platform-specific URI for the app origin.

For example, the verifier for the organization MyExampleOrg is served from https://verify.example.com. The web origin is `https://verify.example.com` with `https` being the scheme, `verify.example.com` being the host, and the port is not explicitly included as 443 is the default port for the protocol `https`. The native applications origin on some platforms will also be `https://verify.example.com` and on other platforms, may be `platform:pkg-key-hash:Z4OFzVVSZrzTRa3eg79hUuHy12MVW0vzPDf4q4zaPs0`.

Presentation: Data that is presented to a specific Verifier, derived from one or more Verifiable Credentials that can be from the same or different Credential Issuers.

VP Token: An artifact containing one or more Verifiable Presentations returned as a response to an Authorization Request. The structure of VP Tokens is defined in Section 8.1.

Verifier: 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 (RP) in [OpenID.Core].

Verifiable Credential (VC): An Issuer-signed Credential whose authenticity can be cryptographically verified. Can be of any format used in the Issuer-Holder-Verifier Model, including, but not limited to those defined in [VC_DATA] (VCDM), [ISO.18013-5] (mdoc), [I-D.ietf-oauth-sd-jwt-vc] (SD-JWT VC), and [Hyperledger.Indy] (AnonCreds).

Verifiable Presentation (VP): A Holder-signed Credential whose authenticity can be cryptographically verified to provide Cryptographic Holder Binding. Can be of any format used in the Issuer-Holder-Verifier Model, including, but not limited to those defined in [VC_DATA] (VCDM), [ISO.18013-5] (mdoc), [I-D.ietf-oauth-sd-jwt-vc] (SD-JWT VC), and [Hyperledger.Indy] (AnonCreds).

W3C Verifiable Credential: A Verifiable Credential compliant to the [VC_DATA] specification.

W3C Verifiable Presentation: A Verifiable Presentation compliant to the [VC_DATA] specification.

Wallet: An entity used by the Holder to receive, store, present, and manage Verifiable Credentials and key material. There is no single deployment model of a Wallet: Verifiable Credentials and keys can both be stored/managed locally, or by using a remote self-hosted service, or a remote third-party service. In the context of this specification, the Wallet acts as an OAuth 2.0 Authorization Server (see [RFC6749]) towards the Credential Verifier which acts as the OAuth 2.0 Client.

3. Overview

This specification defines a mechanism on top of OAuth 2.0 to request and present Verifiable Credentials as Verifiable Presentations.

As the primary extension, OpenID for Verifiable Presentations 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.

This specification supports any Credential format used in the Issuer-Holder-Verifier Model, including, but not limited to those defined in [VC_DATA] (VCDM), [ISO.18013-5] (mdoc), [I-D.ietf-oauth-sd-jwt-vc] (SD-JWT VC), and [Hyperledger.Indy] (AnonCreds). Credentials of multiple formats can be presented in the same transaction. The examples given in the main part of this specification use W3C Verifiable Credentials, while examples in other Credential formats are given in Appendix B.

Implementations can use any pre-existing OAuth 2.0 Grant Type and Response Type in conjunction with this specification to support different deployment architectures.

OpenID for Verifiable Presentations supports scenarios where the Authorization Request is sent both when the Verifier is interacting with the End-User using the device that is the same or different from the device on which requested Credential(s) are stored.

This specification supports the response being sent using a redirect but also using an HTTP POST request. This enables the response to be sent across devices, or when the response size exceeds the redirect URL character size limitation.

Implementations can also be built on top of OpenID Connect Core, which is also based on OAuth 2.0. To benefit from the Self-Issued ID Token feature, this specification can also be combined with the Self-Issued OP v2 specification [SIOPv2].

Any of the OAuth 2.0 related specifications, such as [RFC9126] and [RFC9101], and Best Current Practice (BCP) documents, such as [RFC8252] and [RFC9700], can be implemented on top of this specification.

In summary, OpenID for Verifiable Presentations is a framework that requires profiling to achieve interoperability. Profiling means defining:

- what optional features are used or mandatory to implement, e.g., response encryption;
- which values are permitted for parameters, e.g., credential format identifiers;
- optionally, extensions for new features.

3.1. Same Device Flow

Below is a diagram of a flow where the End-User presents a Credential to a Verifier interacting with the End-User on the same device that the device the Wallet resides on.

The flow utilizes simple redirects to pass Authorization Request and Response between the Verifier and the Wallet. The Verifiable Presentations are returned to the Verifier in the fragment part of the redirect URI, when Response Mode is `fragment`.

Note: The diagram does not illustrate all the optional features of this specification.

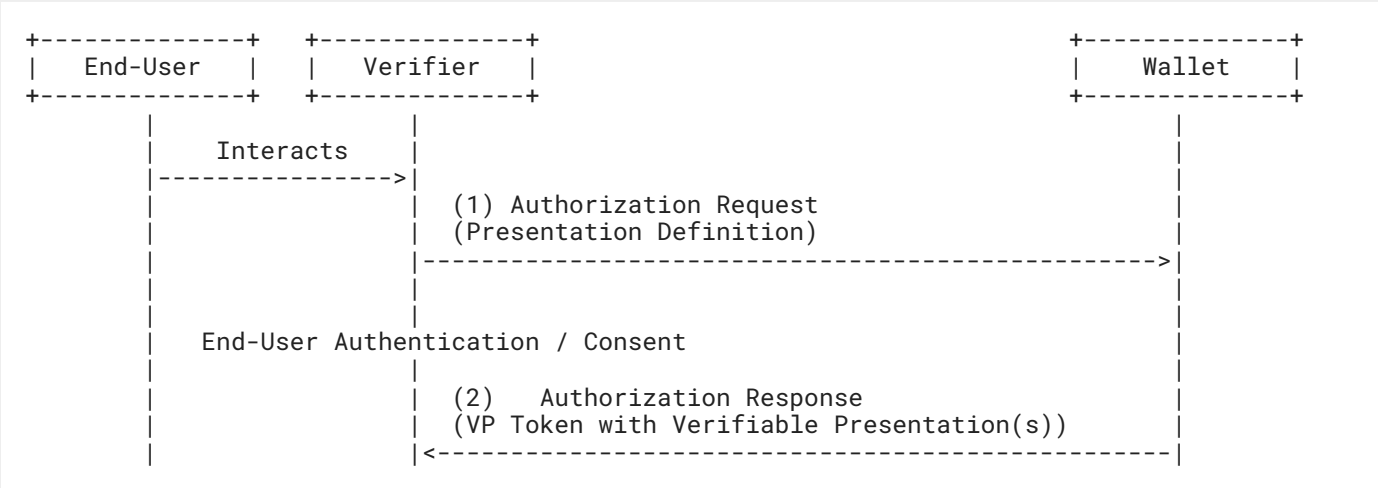

Figure 1: Same Device Flow

(1) The Verifier sends an Authorization Request to the Wallet. It contains a Presentation Definition as defined in [DIF.PresentationExchange] that describes the requirements of the Credential(s) that the Verifier is requesting to be presented. Such requirements could include what type of Credential(s), in what format(s), which individual Claims within those Credential(s) (Selective Disclosure), etc. The Wallet processes the Authorization Request and determines what Credentials are available matching the Verifier's request. The Wallet also authenticates the End-User and gathers consent to present the requested Credentials.

(2) The Wallet prepares the Verifiable Presentation(s) of the Verifiable Credential(s) that the End-User has consented to. It then sends to the Verifier an Authorization Response where the Verifiable Presentation(s) are contained in the `vp_token` parameter.

3.2. Cross Device Flow

Below is a diagram of a flow where the End-User presents a Credential to a Verifier interacting with the End-User on a different device as the device the Wallet resides on.

In this flow, the Verifier prepares an Authorization Request and renders it as a QR Code. The End-User then uses the Wallet to scan the QR Code. The Verifiable Presentations are sent to the Verifier in a direct HTTP POST request to a URL controlled by the Verifier. The flow uses the Response Type `vp_token` in conjunction with the Response Mode `direct_post`, both defined in this specification. In order to keep the size of the QR Code small and be able to sign and optionally encrypt the Request Object, the actual Authorization Request contains just a Request URI according to [RFC9101], which the wallet uses to retrieve the actual Authorization Request data.

Note: The diagram does not illustrate all the optional features of this specification.

Note: The usage of the Request URI as defined in [RFC9101] does not depend on any other choices made in the protocol extensibility points, i.e., it can be used in the Same Device Flow, too.

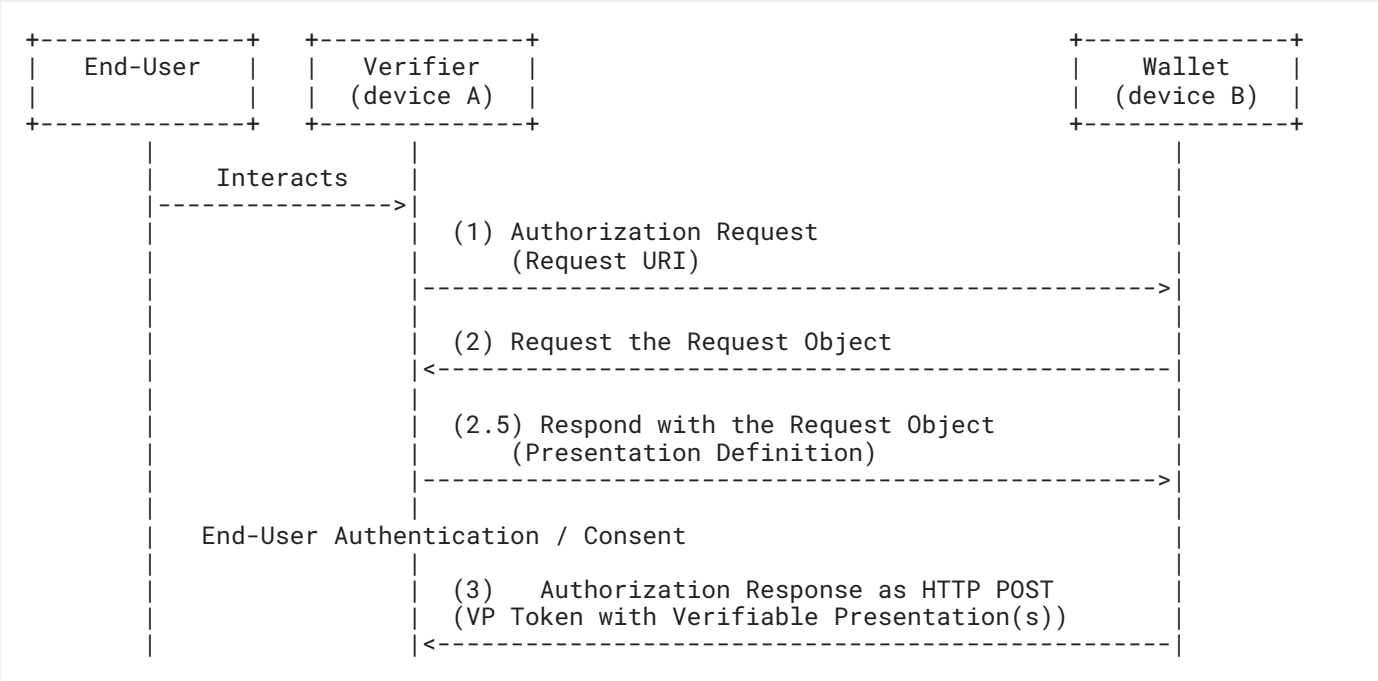

Figure 2: Cross Device Flow

(1) The Verifier sends to the Wallet an Authorization Request that contains a Request URI from where to obtain the Request Object containing Authorization Request parameters.

(2) The Wallet sends an HTTP GET request to the Request URI to retrieve the Request Object.

(2.5) The HTTP GET response returns the Request Object containing Authorization Request parameters. It especially contains a Presentation Definition as defined in [DIF.PresentationExchange] that describes the requirements of the Credential(s) that the Verifier is requesting to be presented. Such requirements could include what type of Credential(s), in what format(s), which individual Claims within those Credential(s) (Selective Disclosure), etc. The Wallet processes the Request Object and determines what Credentials are available matching the Verifier's request. The Wallet also authenticates the End-User and gathers her consent to present the requested Credentials.

(3) The Wallet prepares the Verifiable Presentation(s) of the Verifiable Credential(s) that the End-User has consented to. It then sends to the Verifier an Authorization Response where the Verifiable Presentation(s) are contained in the `vp_token` parameter.

4. Scope

OpenID for Verifiable Presentations extends existing OAuth 2.0 mechanisms as following:

- A new `presentation_definition` Authorization Request parameter that uses the [DIF.PresentationExchange] syntax is defined to request presentation of Verifiable Credentials in arbitrary formats. See Section 5 for more details.
- A new query language, the Digital Credentials Query Language (DCQL), is defined to enable requesting Verifiable Credentials in an easier and more flexible way. See Section 6 for more details.
- A new `dcql_query` Authorization Request parameter is defined to request presentation of Verifiable Credentials in the JSON-encoded DCQL format. See Section 5 for more details.
- A new `vp_token` response parameter is defined to return Verifiable Presentations to the Verifier in either Authorization or Token Response depending on the Response Type. See Section 8 for more details.

- New Response Types `vp_token` and `vp_token id_token` are defined to request Verifiable Credentials to be returned in the Authorization Response (standalone or along with a Self-Issued ID Token [SIOPv2]). See Section 8 for more details.
- A new OAuth 2.0 Response Mode `direct_post` is defined to support sending the response across devices, or when the size of the response exceeds the redirect URL character size limitation. See Section 8.2 for more details.
- The `format` parameter is used throughout the protocol in order to enable customization according to the specific needs of a particular Credential format. Examples in Appendix B are given for Credential formats as specified in [VC_DATA], [ISO.18013-5], [I-D.ietf-oauth-sd-jwt-vc], and [Hyperledger.Indy].
- The concept of a Client Identifier Scheme to enable deployments of this specification to use different mechanisms to obtain and validate metadata of the Verifier beyond the scope of [RFC6749].

Presentation of Verifiable Credentials using OpenID for Verifiable Presentations can be combined with the End-User authentication using [SIOPv2], and the issuance of OAuth 2.0 Access Tokens.

5. Authorization Request

The Authorization Request follows the definition given in [RFC6749] taking into account the recommendations given in [RFC9700].

The Verifier MAY send an Authorization Request as a Request Object either by value or by reference, as defined in the JWT-Secured Authorization Request (JAR) [RFC9101]. Verifiers MUST include the `typ` Header Parameter in Request Objects with the value `oauth-authz-req+jwt`, as defined in [RFC9101]. Wallets MUST NOT process Request Objects where the `typ` Header Parameter is not present or does not have the value `oauth-authz-req+jwt`.

This specification defines a new mechanism for the cases when the Wallet wants to provide to the Verifier details about its technical capabilities to allow the Verifier to generate a request that matches the technical capabilities of that Wallet. To enable this, the Authorization Request can contain a `request_uri_method` parameter with the value `post` that signals to the Wallet that it can make an HTTP POST request to the Verifier's `request_uri` endpoint with information about its capabilities as defined in Section 5.11. The Wallet MAY continue with JAR when it receives `request_uri_method` parameter with the value `post` but does not support this feature.

The Verifier articulates requirements of the Credential(s) that are requested using `presentation_definition` and `presentation_definition_uri` parameters that contain a Presentation Definition JSON object as defined in Section 7 of [DIF.PresentationExchange]. Wallet implementations MUST process Presentation Definition JSON object and select candidate Verifiable Credential(s) using the evaluation process described in Section 8 of [DIF.PresentationExchange] unless implementing only a profile of [DIF.PresentationExchange] that provides rules on how to evaluate and process [DIF.PresentationExchange].

The Verifier communicates a Client Identifier Scheme that indicate how the Wallet is supposed to interpret the Client Identifier and associated data in the process of Client identification, authentication, and authorization as a prefix in the `client_id` parameter. This enables deployments of this specification to use different mechanisms to obtain and validate Client metadata beyond the scope of [RFC6749]. A certain Client Identifier Scheme MAY require the Verifier to sign the Authorization Request as means of authentication and/or pass additional parameters and require the Wallet to process them.

Depending on the Client Identifier Scheme, the Verifier can communicate a JSON object with its metadata using the `client_metadata` parameter which contains name/value pairs.

This specification enables the Verifier to send both Presentation Definition JSON object and Client Metadata JSON object by value or by reference.

Additional request parameters, other than those defined in this section, MAY be defined and used, as described in [RFC6749]. The Wallet MUST ignore any unrecognized parameters, other than the `transaction_data` parameter. One exception to this rule is `transaction_data` parameter, and the wallets that do not support this parameter MUST reject requests that contain it.

5.1. New Parameters

This specification defines the following new request parameters:

`presentation_definition`: A JSON object containing a Presentation Definition. See Section 5.4 for more details.
`presentation_definition_uri`: A string containing an HTTPS URL pointing to a resource where a Presentation Definition JSON object can be retrieved. See Section 5.5 for more details.
`dcql_query`: A JSON object containing a DCQL query as defined in Section 6.

Exactly one of the following parameters MUST be present in the Authorization Request: `dcql_query`, `presentation_definition`, `presentation_definition_uri`, or a `scope` value representing a Presentation Definition.

In the context of an authorization request according to [RFC6749], parameters containing objects are transferred as JSON-serialized strings (using the application/x-www-form-urlencoded format as usual for request parameters).

`client_metadata`: OPTIONAL. A JSON object containing the Verifier metadata values. It MUST be UTF-8 encoded. The following metadata parameters MAY be used:

- `jwks`: OPTIONAL. A JWKS as defined in [RFC7591]. It MAY contain one or more public keys, such as those used by the Wallet as an input to a key agreement that may be used for encryption of the Authorization Response (see Section 8.3), or where the Wallet will require the public key of the Verifier to generate the Verifiable Presentation. This allows the Verifier to pass ephemeral keys specific to this Authorization Request. Public keys included in this parameter MUST NOT be used to verify the signature of signed Authorization Requests.
- `vp_formats`: REQUIRED when not available to the Wallet via another mechanism. As defined in Section 11.1.
- `authorization_signed_response_alg`: OPTIONAL. As defined in [JARM], with an adjustment to the default behavior when this parameter is absent: instead of defaulting to `RS256`, the Authorization Response is not signed.
- `authorization_encrypted_response_alg`: OPTIONAL. As defined in [JARM].
- `authorization_encrypted_response_enc`: OPTIONAL. As defined in [JARM].

Authoritative data the Wallet is able to obtain about the Client from other sources, for example those from an OpenID Federation Entity Statement, take precedence over the values passed in `client_metadata`.

Other metadata parameters MUST be ignored unless a profile of this specification explicitly defines them as usable in the `client_metadata` parameter.

`request_uri_method`: OPTIONAL. A string determining the HTTP method to be used when the `request_uri` parameter is included in the same request. Two case-sensitive valid values are defined in this specification: `get` and `post`. If `request_uri_method` value is `get`, the Wallet MUST send the request to retrieve the Request Object using the HTTP GET method, i.e., as defined in [RFC9101]. If `request_uri_method` value is `post`, a supporting Wallet MUST send the request using the HTTP POST method as detailed in Section 5.11. If the `request_uri_method` parameter is not present, the Wallet MUST process the `request_uri` parameter as defined in [RFC9101]. Wallets not supporting the `post` method will send a GET request to the Request URI (default behavior as defined in [RFC9101]). `request_uri_method` parameter MUST NOT be present if a `request_uri` parameter is not present.

If the Verifier set the `request_uri_method` parameter value to `post` and there is no other means to convey its capabilities to the Wallet, it SHOULD add the `client_metadata` parameter to the Authorization Request. This enables the Wallet to assess the Verifier's capabilities, allowing it to transmit only the relevant capabilities through the `wallet_metadata` parameter in the Request URI POST request.

`transaction_data`: OPTIONAL. Array of strings, where each string is a base64url encoded JSON object that contains a typed parameter set with details about the transaction that the Verifier is requesting the End-User to authorize. See Section 8.4 for details. The Wallet MUST return an error if a request contains even one unrecognized transaction data type or transaction data not conforming to the respective type definition. In addition to the parameters determined by the type of transaction data, each `transaction_data` object consists of the following parameters defined by this specification:

- `type`: REQUIRED. String that identifies the type of transaction data . This value determines parameters that can be included in the `transaction_data` object. The specific values are out of scope of this specification. It is RECOMMENDED to use collision-resistant names for `type` values.
- `credential_ids`: REQUIRED. Array of strings each referencing a Credential requested by the Verifier that can be used to authorize this transaction. In [DIF.PresentationExchange], the string matches the `id` field in the Input Descriptor. In the Digital Credentials Query Language, the string matches the `id` field in the Credential Query. If there is more than one element in the array, the Wallet MUST use only one of the referenced Credentials for transaction authorization.

Each document specifying details of a transaction data type defines what Credential(s) can be used to authorize those transactions. Those Credential(s) can be issued specifically for the transaction authorization use case or re-use existing Credential(s) used for user identification. A mechanism for Credential Issuers to express that a particular Credential can be used for authorization of transaction data is out of scope for this specification.

The following is a non-normative example of a transaction data content, after base64url decoding one of the strings in the `transaction_data` parameter:

```
{
  "type": "example_type",
  "credential_ids": [ "id card credential" ],
  // other transaction data type specific parameters
}
```

5.2. Existing Parameters

The following additional considerations are given for pre-existing Authorization Request parameters:

`nonce`: REQUIRED. Defined in [OpenID.Core]. It is used to securely bind the Verifiable Presentation(s) provided by the Wallet to the particular transaction. See Section 14.1 for details. Values MUST only contain ASCII URL safe characters (uppercase and lowercase letters, decimal digits, hyphen, period, underscore, and tilde).

`scope`: OPTIONAL. Defined in [RFC6749]. The Wallet MAY allow Verifiers to request presentation of Verifiable Credentials by utilizing a pre-defined scope value. See Section 5.6 for more details.

`response_mode`: REQUIRED. Defined in [OAuth.Responses]. This parameter is used (through the new Response Mode `direct_post`) to ask the Wallet to send the response to the Verifier via an HTTPS connection (see Section 8.2 for more details). It is also used to request signing and encrypting (see Section 8.3 for more details).

`client_id`: REQUIRED. Defined in [RFC6749]. This specification defines additional requirements to enable the use of Client Identifier Schemes as described in Section 5.10.

5.3. Examples

The Verifier MAY send an Authorization Request using either of these 3 options:

1. Passing as URL with encoded parameters
2. Passing a request object as value
3. Passing a request object by reference 2 and 3 are defined in the JWT-Secured Authorization Request (JAR) [RFC9101].

The following is a non-normative example of an Authorization Request with URL encoded parameters:

```
GET /authorize?
  response_type=vp_token
  &client_id=redirect_uri%3Ahttps%3A%2F%2Fclient.example.org%2Fcb
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
  &presentation_definition=...
  &transaction_data=...
  &nonce=n-0S6_WzA2Mj HTTP/1.1
```

The following is a non-normative example of Authorization Request with request object as value:

```
GET /authorize?
  client_id=redirect_uri%3Ahttps%3A%2F%2Fclient.example.org%2Fcb
  &request=eyJrd...
```

Where the contents of `request` consist of base64url-encoding and signing (in the example with RS256 algo) this json:

```
{
  "iss": "redirect_uri:https://client.example.org/cb",
  "aud": "https://self-issued.me/v2",
  "response_type": "vp_token",
  "client_id": "redirect_uri:https://client.example.org/cb",
  "redirect_uri": "https//client.example.org/cb",
  "presentation_definition": {
    "id": "example_jwt_vc",
    "input_descriptors": [
      {
        "id": "id_credential",
        "format": {
          "jwt_vc_json": {
            "proof_type": [
              "JsonWebSignature2020"
            ]
          }
        },
        "constraints": {
          "fields": [
            {
              "path": [
                "$.vc.type"
              ],
              "filter": {
                "type": "array",
                "contains": {
                  "const": "IDCredential"
                }
              }
            }
          ]
        }
      }
    ]
  },
  "nonce": "n-0S6_WzA2Mj"
}
```

The following is a non-normative example of Authorization Request with request object as reference:

```
GET /authorize?
  client_id=x509_san_dns%3Aclient.example.org
  &request_uri=https%3A%2F%2Fclient.example.org%2Frequest%2Fvapof4ql2i7m41m68uep
  &request_uri_method=post HTTP/1.1
```

Later, the wallet might send the following non-normative example request to the `request_uri`:

```
POST /request/vapof4ql2i7m41m68uep HTTP/1.1
Host: client.example.org
Content-Type: application/x-www-form-urlencoded
wallet_metadata=%7B%22vp_formats_supported%22%3A%7B%22jwt_vc_json%22%3A%7B%22alg_values_su
pported
%22%3A%5B%22ES256K%22%2C%22ES384%22%5D%7D%2C%22jwt_vp_json%22%3A%7B%22alg_values_supported
%22%3A%
5B%22ES256K%22%2C%22EdDSA%22%5D%7D%7D%7D&
wallet_nonce=qPmxiNFCR3QTm19POc8u
```

5.4. `presentation_definition` Parameter

This parameter contains a Presentation Definition JSON object conforming to the syntax defined in Section 7 of [DIF.PresentationExchange].

The following is a non-normative example how `presentation_definition` parameter can simply be used to request the presentation of a Credential of a certain type:

```
{
  "id": "vp token example",
  "input_descriptors": [
    {
      "id": "id card credential",
      "format": {
        "ldp_vc": {
          "proof_type": [
            "Ed25519Signature2018"
          ]
        }
      },
      "constraints": {
        "fields": [
          {
            "path": [
              "$.type"
            ],
            "filter": {
              "type": "string",
              "pattern": "IDCardCredential"
            }
          }
        ]
      }
    }
  ]
}
```

The following non-normative example shows how the Verifier can request selective disclosure or certain claims from a Credential of a particular type.

```
{
  "id": "example with selective disclosure",
  "input_descriptors": [
    {
      "id": "ID card with constraints",
      "format": {
        "ldp_vc": {
          "proof_type": [
            "Ed25519Signature2018"
          ]
        }
      },
      "constraints": {
        "limit_disclosure": "required",
        "fields": [
          {
            "path": [
              "$.type"
            ],
            "filter": {
              "type": "string",
              "pattern": "IDCardCredential"
            }
          },
          {
            "path": [
              "$.credentialSubject.given_name"
            ]
          },
          {
            "path": [
              "$.credentialSubject.family_name"
            ]
          },
          {
            "path": [
              "$.credentialSubject.birthdate"
            ]
          }
        ]
      }
    }
  ]
}
```

The following non-normative example shows how the Verifiers can also ask for alternative Verifiable Credentials being presented:

```
{
  "id": "alternative credentials",
  "submission_requirements": [
    {
      "name": "Citizenship Information",
      "rule": "pick",
      "count": 1,
      "from": "A"
    }
  ],
  "input_descriptors": [
    {
      "id": "id card credential",
```

```
      "group": [
        "A"
      ],
      "format": {
        "ldp_vc": {
          "proof_type": [
            "Ed25519Signature2018"
          ]
        }
      },
      "constraints": {
        "fields": [
          {
            "path": [
              "$.type"
            ],
            "filter": {
              "type": "string",
              "pattern": "IDCardCredential"
            }
          }
        ]
      }
    },
    {
      "id": "passport credential",
      "format": {
        "jwt_vc_json": {
          "alg": [
            "RS256"
          ]
        }
      },
      "group": [
        "A"
      ],
      "constraints": {
        "fields": [
          {
            "path": [
              "$.vc.type"
            ],
            "filter": {
              "type": "string",
              "pattern": "PassportCredential"
            }
          }
        ]
      }
    }
  ]
}
```

The Verifiable Credential and Verifiable Presentation formats supported by the Wallet should be published in its metadata using the metadata parameter `vp_formats_supported` (see Section 10).

The formats supported by a Verifier may be set up using the metadata parameter `vp_formats` (see Section 11.1). The Wallet MUST ignore any `format` property inside a `presentation_definition` object if that `format` was not included in the `vp_formats` property of the metadata.

Note: When a Verifier is requesting the presentation of a Verifiable Presentation containing a Verifiable Credential, the Verifier MUST indicate in the `vp_formats` parameter the supported formats of both Verifiable Credential and Verifiable Presentation.

5.5. `presentation_definition_uri` Parameter

`presentation_definition_uri` is used to retrieve the Presentation Definition from the resource at the specified URL, rather than being passed by value. The Wallet MUST send an HTTP GET request without additional parameters. The resource MUST be exposed without further need to authenticate or authorize.

The protocol for the `presentation_definition_uri` MUST be HTTPS.

The following is a non-normative example of an HTTP GET request sent after the Wallet received `presentation_definition_uri` parameter with the value `https://server.example.com/presentationdefs?ref=idcard_presentation_request`:

```
GET /presentationdefs?ref=idcard_presentation_request HTTP/1.1
Host: server.example.com
```

The following is a non-normative example of an HTTP GET response sent by the Verifier in response to the above HTTP GET request:

```
HTTP/1.1 200 OK
...
Content-Type: application/json

{
  "id": "vp token example",
  "input_descriptors": [
    {
      "id": "id card credential",
      "format": {
        "ldp_vc": {
          "proof_type": [
            "Ed25519Signature2018"
          ]
        }
      },
      "constraints": {
        "fields": [
          {
            "path": [
              "$.type"
            ],
            "filter": {
              "type": "string",
              "pattern": "IDCardCredential"
            }
          }
        ]
      }
    }
  ]
}
```

5.6. Using scope Parameter to Request Verifiable Credential(s)

Wallets MAY support requesting presentation of Verifiable Credentials using OAuth 2.0 scope values.

Such a scope value MUST be an alias for - a well-defined DCQL query, or - a well-defined Presentation Definition (for [DIF.PresentationExchange]) that will be referred to in the `presentation_submission` response parameter.

The specific scope values, and the mapping between a certain scope value and the respective DCQL query or Presentation Definition is out of scope of this specification.

Possible options include normative text in a separate specification defining scope values along with a description of their semantics or machine-readable definitions in the Wallet's server metadata, mapping a scope value to an equivalent Presentation Definition JSON object.

If [DIF.PresentationExchange] is used, the definition of the scope value MUST allow the Verifier to determine the identifiers of the Presentation Definition and Input Descriptor(s) in the `presentation_submission` response parameter (`definition_id` and `descriptor_map.id` respectively) as well as the Credential formats and types in the `vp_token` response parameter defined in Section 8.1.

It is RECOMMENDED to use collision-resistant scopes values.

The following is a non-normative example of an Authorization Request using the scope value `com.example.IDCardCredential_presentation`, which is an alias for the first Presentation Definition example given in Section 5.4:

```
GET /authorize?
  response_type=vp_token
  &client_id=https%3A%2F%2Fclient.example.org%2Fcb
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
  &scope=com.example.healthCardCredential_presentation
  &nonce=n-0S6_WzA2Mj HTTP/1.1
```

5.7. Response Type `vp_token`

This specification defines the Response Type `vp_token`.

`vp_token`: When supplied as the `response_type` parameter in an Authorization Request, a successful response MUST include the `vp_token` parameter. The Wallet SHOULD NOT return an OAuth 2.0 Authorization Code, Access Token, or Access Token Type in a successful response to the grant request. The default Response Mode for this Response Type is `fragment`, i.e., the Authorization Response parameters are encoded in the fragment added to the `redirect_uri` when redirecting back to the Verifier. The Response Type `vp_token` can be used with other Response Modes as defined in [OAuth.Responses]. Both successful and error responses SHOULD be returned using the supplied Response Mode, or if none is supplied, using the default Response Mode.

See Section 8 on how the `response_type` value determines the response used to return a VP Token.

5.8. Passing Authorization Request Across Devices

There are use-cases when the Authorization Request is being displayed on a device different from a device on which the requested Credential is stored. In those cases, an Authorization Request can be passed across devices by being rendered as a QR Code.

The usage of the Response Mode `direct_post` (see Section 8.2) in conjunction with `request_uri` is RECOMMENDED, since Authorization Request size might be large and might not fit in a QR code.

5.9. aud of a Request Object

When the Verifier is sending a Request Object as defined in [RFC9101], the aud Claim value depends on whether the recipient of the request can be identified by the Verifier or not:

- the aud Claim MUST equal to the `issuer` Claim value, when Dynamic Discovery is performed.
- the aud Claim MUST be "https://self-issued.me/v2", when Static Discovery metadata is used.

Note: "https://self-issued.me/v2" is a symbolic string and can be used as an aud Claim value even when this specification is used standalone, without SIOPv2.

5.10. Client Identifier Scheme and Verifier Metadata Management

This specification defines the concept of a Client Identifier Scheme that indicates how the Wallet is supposed to interpret the Client Identifier and associated data in the process of Client identification, authentication, and authorization. The Client Identifier Scheme enables deployments of this specification to use different mechanisms to obtain and validate metadata of the Verifier beyond the scope of [RFC6749]. The term Client Identifier Scheme is used since the Verifier is acting as an OAuth 2.0 Client.

The Client Identifier Scheme is a string that MAY be communicated by the Verifier in a prefix within the `client_id` parameter in the Authorization Request. A fallback to pre-registered Clients as in [RFC6749] remains in place as a default mechanism in case no Client Identifier Scheme was provided. A certain Client Identifier Scheme may require the Verifier to sign the Authorization Request as means of authentication and/or pass additional parameters and require the Wallet to process them.

5.10.1. Syntax

In the `client_id` Authorization Request parameter and other places where the Client Identifier is used, the Client Identifier Schemes are prefixed to the usual Client Identifier, separated by a `:` (colon) character:

```
<client_id_scheme>:<orig_client_id>
```

Here, `<client_id_scheme>` is the Client Identifier Scheme and `<orig_client_id>` is an identifier for the Client within the namespace of that scheme. See Section 5.10.4 for Client Identifier Schemes defined by this specification.

Wallets MUST use the presence of a `:` (colon) character to determine whether a Client Identifier Scheme is used. If a `:` character is present, the Wallet MUST interpret the Client Identifier according to the Client Identifier Scheme, here defined as the string before the (first) `:` character. If the Wallet does not support the Client Identifier Scheme, the Wallet MUST refuse the request.

For example, an Authorization Request might contain `client_id=verifier_attestation:example-client` to indicate that the `verifier_attestation` Client Identifier Scheme is to be used and that within this scheme, the Verifier can be identified by the string `example-client`. The presentation would contain the full `verifier_attestation:example-client` string as the audience (intended receiver) and the same full string would be used as the Client Identifier anywhere in the OAuth flow.

Note that the Verifier needs to determine which Client Identifier Schemes the Wallet supports prior to sending the Authorization Request in order to choose a supported scheme.

Depending on the Client Identifier Scheme, the Verifier can communicate a JSON object with its metadata using the `client_metadata` parameter which contains name/value pairs.

5.10.2. Fallback

If a `:` character is not present in the Client Identifier, the Wallet MUST treat the Client Identifier as referencing a pre-registered client. This is equivalent to the [RFC6749] default behavior, i.e., the Client Identifier needs to be known to the Wallet in advance of the Authorization Request. The Verifier metadata is obtained using [RFC7591] or through out-of-band mechanisms.

For example, if an Authorization Request contains `client_id=example-client`, the Wallet would interprete the Client Identifier as referring to a pre-registered client.

From this definition, it follows that pre-registered clients MUST NOT contain a `:` character in their Client Identifier.

5.10.3. Security Considerations

Confusing Verifiers using a Client Identifier Scheme with those using none can lead to attacks. Therefore, Wallets MUST always use the full Client Identifier, including the prefix if provided, within the context of the Wallet or its responses to identify the client. This refers in particular to places where the Client Identifier is used in [RFC6749] and in the presentation returned to the Verifier.

5.10.4. Defined Client Identifier Schemes

This specification defines the following Client Identifier Schemes, followed by the examples where applicable:

- `redirect_uri`: This value indicates that the Client Identifier (without the prefix `redirect_uri:`) is the Verifier's Redirect URI (or Response URI when Response Mode `direct_post` is used). The Verifier MAY omit the `redirect_uri` Authorization Request parameter (or `response_uri` when Response Mode `direct_post` is used). All Verifier metadata parameters MUST be passed using the `client_metadata` parameter defined in Section 5.1. An example Client Identifier value is `redirect_uri:https://client.example.org/cb`. Requests using the `redirect_uri` Client Identifier Scheme cannot be signed because there is no method for the Wallet to obtain a trusted key for verification. Therefore, implementations requiring signed requests cannot use the `redirect_uri` Client ID scheme.

The following is a non-normative example of an unsigned request with the `redirect_uri` Client Identifier Scheme:

```
HTTP/1.1 302 Found
Location: https://wallet.example.org/universal-link?
  response_type=vp_token
  &client_id=redirect_uri%3Ahttps%3A%2F%2Fclient.example.org%2Fcb
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
  &presentation_definition=...
  &nonce=n-0S6_WzA2Mj
  &client_metadata=%7B%22vp_formats%22%3A%7B%22jwt_vp_json%22%3A%
    7B%22alg%22%3A%5B%22EdDSA%22%2C%22ES256K%22%5D%7D%2C%22ldp_vp
    %22%3A%7B%22proof_type%22%3A%5B%22Ed25519Signature2018%22%5D%
    7D%7D%7D
```

- `https`: This value indicates that the Client Identifier is an Entity Identifier defined in OpenID Federation [OpenID.Federation]. Since the Entity Identifier is already defined to start with `https:`, this Client Identifier Scheme MUST NOT be prefixed additionally. Processing rules given in [OpenID.Federation] MUST be followed. Automatic Registration as defined in [OpenID.Federation] MUST be used. The Authorization Request MAY also contain a `trust_chain` parameter. The final Verifier metadata is obtained from the Trust Chain after applying the policies, according to [OpenID.Federation]. The `client_metadata` parameter, if present in the Authorization Request, MUST be ignored when this Client Identifier scheme is used. Example Client Identifier: `https://federation-verifier.example.com`.
- `did`: This value indicates that the Client Identifier is a DID defined in [DID-Core]. Since the DID URI is already defined to start with `did:`, this Client Identifier Scheme MUST NOT be prefixed additionally. The request MUST be signed with a private key associated with the DID. A public key to verify the signature MUST be obtained from the `verificationMethod` property of a DID Document. Since DID Document may include multiple public keys, a particular public key used to sign the request in question MUST be identified by the `kid` in the

JOSE Header. To obtain the DID Document, the Wallet MUST use DID Resolution defined by the DID method used by the Verifier. All Verifier metadata other than the public key MUST be obtained from the `client_metadata` parameter as defined in Section 5.1. Example Client Identifier: `did:example:123#1`.

The following is a non-normative example of a header and a body of a signed Request Object when Client Identifier scheme is a `did`:

Header

```
{
  "typ": "oauth-authz-req+jwt",
  "alg": "RS256",
  "kid": "did:example:123#1"
}
```

Body

```
{
  "client_id": "did:example:123",
  "response_type": "vp_token",
  "redirect_uri": "https://client.example.org/callback",
  "nonce": "n-0S6_WzA2Mj",
  "presentation_definition": { ... },
  "client_metadata": {
    "vp_formats": {
      "jwt_vp": {
        "alg": [
          "EdDSA",
          "ES256K"
        ]
      },
      "ldp_vp": {
        "proof_type": [
          "Ed25519Signature2018"
        ]
      }
    }
  }
}
```

- `verifier_attestation`: This Client Identifier Scheme allows the Verifier to authenticate using a JWT that is bound to a certain public key as defined in Section 12. When the Client Identifier Scheme is `verifier_attestation`, the Client Identifier MUST equal the `sub` claim value in the Verifier attestation JWT. The request MUST be signed with the private key corresponding to the public key in the `cnf` claim in the Verifier attestation JWT. This serves as proof of possesion of this key. The Verifier attestation JWT MUST be added to the `jwt` JOSE Header of the request object (see Section 12). The Wallet MUST validate the signature on the Verifier attestation JWT. The `iss` claim value of the Verifier Attestation JWT MUST identify a party the Wallet trusts for issuing Verifier Attestation JWTs. If the Wallet cannot establish trust, it MUST refuse the request. If the issuer of the Verifier Attestation JWT adds a `redirect_uris` claim to the attestation, the Wallet MUST ensure the `redirect_uri` request parameter value exactly matches one of the `redirect_uris` claim entries. All Verifier metadata other than the public key MUST be obtained from the `client_metadata` parameter. Example Client Identifier: `verifier_attestation:verifier.example`.
- `x509_san_dns`: When the Client Identifier Scheme is `x509_san_dns`, the Client Identifier MUST be a DNS name and match a `dNSName` Subject Alternative Name (SAN) [RFC5280] entry in the leaf certificate passed with the request. The request MUST be signed with the private key corresponding to the public key in the leaf

X.509 certificate of the certificate chain added to the request in the `x5c` JOSE header [RFC7515] of the signed request object. The Wallet MUST validate the signature and the trust chain of the X.509 certificate. All Verifier metadata other than the public key MUST be obtained from the `client_metadata` parameter. If the Wallet can establish trust in the Client Identifier authenticated through the certificate, e.g. because the Client Identifier is contained in a list of trusted Client Identifiers, it may allow the client to freely choose the `redirect_uri` value. If not, the FQDN of the `redirect_uri` value MUST match the Client Identifier without the prefix `x509_san_dns:`. Example Client Identifier: `x509_san_dns:client.example.org`.

- `origin`: This Client Identifier Scheme is defined in Appendix A.2. The Wallet MUST NOT accept this Client Identifier Scheme in requests. In OpenID4VP over the Digital Credentials API, the audience of the Credential Presentation is always the origin value prefixed by `origin:`, for example `origin:https://verifier.example.com/`.
- `x509_hash`: When the Client Identifier Scheme is `x509_hash`, the Client Identifier MUST be a hash and match the hash of the leaf certificate passed with the request. The request MUST be signed with the private key corresponding to the public key in the leaf X.509 certificate of the certificate chain added to the request in the `x5c` JOSE header parameter [RFC7515] of the signed request object. The value of `x509_hash` is the base64url encoded value of the SHA-256 hash of the DER-encoded X.509 certificate. The Wallet MUST validate the signature and the trust chain of the X.509 leaf certificate. All verifier metadata other than the public key MUST be obtained from the `client_metadata` parameter. Example Client Identifier: `x509_hash:Uvo3HtuIxuhC92rShpgqcT3YXwrqRxWEviRiA0OZszk`

To use the Client Identifier Schemes `https`, `did`, `verifier_attestation`, `x509_san_dns` and `x509_hash`, Verifiers MUST be confidential clients. This might require changes to the technical design of native apps as such apps are typically public clients.

Other specifications can define further Client Identifier Schemes. It is RECOMMENDED to use collision-resistant names for such values.

5.11. Request URI Method `post`

This request is handled by the Request URI endpoint of the Verifier.

The request MUST use the HTTP POST method with the `https` scheme, and the content type `application/x-www-form-urlencoded` and the accept header set to `application/oauth-authz-req+jwt`. The names and values in the body MUST be encoded using UTF-8.

The following parameters are defined to be included in the request to the Request URI Endpoint:

`wallet_metadata`: OPTIONAL. A String containing a JSON object containing metadata parameters as defined in Section 10.

`wallet_nonce`: OPTIONAL. A String value used to mitigate replay attacks of the Authorization Request. When received, the Verifier MUST use it as the `wallet_nonce` value in the signed authorization request object. Value can be a base64url-encoded, fresh, cryptographically random number with sufficient entropy.

If the Wallet requires the Verifier to encrypt the Request Object, it SHOULD use the `jwks` or `jwks_uri` parameter within the `wallet_metadata` parameter to pass the public key for the input to the key agreement. Other mechanisms to pass the encryption key can be used as well. If the Wallet requires an encrypted Authorization Response, it SHOULD specify supported encryption algorithms using the `authorization_encryption_alg_values_supported` and `authorization_encryption_enc_values_supported` parameters.

Additionally, if the Client Identifier Scheme permits signed Request Objects, the Wallet SHOULD list supported cryptographic algorithms for securing the Request Object through the `request_object_signing_alg_values_supported` parameter. Conversely, the Wallet MUST NOT include this

parameter if the Client Identifier Scheme precludes signed Request Objects.

Additional parameters MAY be defined and used in the request to the Request URI Endpoint. The Verifier MUST ignore any unrecognized parameters.

The following is a non-normative example of a request:

```
POST /request HTTP/1.1
Host: client.example.org
Content-Type: application/x-www-form-urlencoded


wallet_metadata=%7B%22vp_formats_supported%22%3A%7B%22jwt_vc_json%22%3A%7B%22alg_values_su
pported

%22%3A%5B%22ES256K%22%2C%22ES384%22%5D%7D%2C%22jwt_vp_json%22%3A%7B%22alg_values_supported
%22%3A%
  5B%22ES256K%22%2C%22EdDSA%22%5D%7D%7D%7D&
  wallet_nonce=qPmxiNFCR3QTm19POc8u
```

5.11.1. Request URI Response

The Request URI response MUST be an HTTP response with the content type "application/oauth-authz-req+jwt" and the body being a signed, optionally encrypted, request object as defined in [RFC9101]. The request object MUST fulfill the requirements as defined in Section 5.

The following is a non-normative example of a request object:

```
{
  "client_id": "x509_san_dns:client.example.org",
  "response_uri": "https://client.example.org/post",
  "response_type": "vp_token",
  "response_mode": "direct_post",
  "presentation_definition": {...},
  "nonce": "n-0S6_WzA2Mj",
  "wallet_nonce": "qPmxiNFCR3QTm19POc8u",
  "state" : "eyJhb...6-sVA"
}
```

The Wallet MUST process the request as defined in [RFC9101]. Additionally, if the Wallet passed a `wallet_nonce` in the POST request, the Wallet MUST validate whether the request object contains the respective nonce value in a `wallet_nonce` claim. If it does not, the Wallet MUST terminate request processing.

The Wallet MUST extract the set of Authorization Request parameters from the Request Object. The Wallet MUST only use the parameters in this Request Object, even if the same parameter was provided in an Authorization Request query parameter. The Client Identifier value in the `client_id` Authorization Request parameter and the Request Object `client_id` claim value MUST be identical, including the Client Identifier Scheme. If any of these conditions are not met, the Wallet MUST terminate request processing.

The Wallet then validates the request as specified in OAuth 2.0 [RFC6749].

5.11.2. Request URI Error Response

If the Verifier responds with any HTTP error response, the Wallet MUST terminate the process.

6. Digital Credentials Query Language (DCQL)

The Digital Credentials Query Language (DCQL, pronounced ['dakl]) is a JSON-encoded query language that allows the Verifier to request Verifiable Presentations that match the query. The Verifier MAY encode constraints on the combinations of credentials and claims that are requested. The Wallet evaluates the query against the Verifiable Credentials it holds and returns Verifiable Presentations matching the query.

A valid DCQL query is defined as a JSON-encoded object with the following top-level properties:

`credentials`: REQUIRED. A non-empty array of Credential Queries as defined in Section 6.1 that specify the requested Verifiable Credentials.
`credential_sets`: OPTIONAL. A non-empty array of credential set queries as defined in Section 6.2 that specifies additional constraints on which of the requested Verifiable Credentials to return.

Note: Future extensions may define additional properties both on the top level and in the rest of the DCQL data structure. Implementations MUST ignore any unknown properties.

6.1. Credential Query

A Credential Query is an object representing a request for a presentation of one or more matching Credentials.

Each entry in `credentials` MUST be an object with the following properties:

`id`: REQUIRED. A string identifying the Credential in the response and, if provided, the constraints in `credential_sets`. The value MUST be a non-empty string consisting of alphanumeric, underscore (_) or hyphen (-) characters. Within the Authorization Request, the same `id` MUST NOT be present more than once.
`format`: REQUIRED. A string that specifies the format of the requested Verifiable Credential. Valid Credential Format Identifier values are defined in Appendix B.
`multiple`: OPTIONAL. A boolean which indicates whether multiple Credentials can be returned for this Credential Query. If omitted, the default value is `false`.
`meta`: OPTIONAL. An object defining additional properties requested by the Verifier that apply to the metadata and validity data of the Credential. The properties of this object are defined per Credential Format. Examples of those are in Appendix B.4.4.1 and Appendix B.3.2.1. If omitted, no specific constraints are placed on the metadata or validity of the requested Credential.
`trusted_authorities`: OPTIONAL. A non-empty array of objects as defined in Section 6.1.1 that specifies expected authorities or trust frameworks that certify Issuers, that the Verifier will accept. Every Credential returned by the Wallet SHOULD match at least one of the conditions present in the corresponding `trusted_authorities` array if present.

Note that Relying Parties must verify that the issuer of a received presentation is trusted on their own and this feature mainly aims to help data minimization by not revealing information that would likely be rejected.

`claims`: OPTIONAL. A non-empty array of objects as defined in Section 6.3 that specifies claims in the requested Credential. Verifiers MUST NOT point to the same claim more than once in a single query. Wallets SHOULD ignore such duplicate claim queries.
`claim_sets`: OPTIONAL. A non-empty array containing arrays of identifiers for elements in `claims` that specifies which combinations of `claims` for the Credential are requested. The rules for selecting claims to send are defined in Section 6.3.1.1.

Note that multiple Credential Queries in a request MAY request a presentation of the same Credential.

6.1.1. Trusted Authorities Query

A Trusted Authorities Query is an object representing information that helps to identify an authority or the trust framework that certifies Issuers. A Credential is identified as a match to a Trusted Authorities Query if it matches with one of the provided values in one of the provided types. How exactly the matching works is defined for the different types below

Note that direct Issuer matching can also work using claim value matching if supported (e.g., value matching the `iss` claim in an SD-JWT) if the mechanisms for `trusted_authorities` are not applicable but might be less likely to work due to the constraints on value matching (see #selecting_claims for more details).

Each entry in `trusted_authorities` MUST be an object with the following properties:

`type`: REQUIRED. A string uniquely identifying the type of information about the issuer trust framework. Types defined by this specification are listed below.
`values`: REQUIRED. An array of strings, where each string (value) contains information specific to the used Trusted Authorities Query type that allows to identify an issuer, trust framework, or a federation that an issuer belongs to.

Below are descriptions for the different Type Identifiers (string), the description on how to interpret and perform the matching logic for each provided value.

Note that depending on the trusted authorities type used, the underlying mechanisms can have different privacy implications. More detail on privacy considerations for the trusted authorities can be found in Section 15.3.

6.1.1.1. Authority Key Identifier

Type: `"aki"`
Value: Contains the KeyIdentifier of the AuthorityKeyIdentifier as defined in Section 4.2.1.1 of [RFC5280], encoded as base64url. The raw byte representation of this element MUST match with the AuthorityKeyIdentifier element of an X.509 certificate in the certificate chain present in the credential (e.g., in the header of an mdoc or SD-JWT). Note that the chain can consist of a single certificate and the credential can include the entire X.509 chain or parts of it.

Below is a non-normative example of such an entry of type `aki`:

```
{
  "type": "aki",
  "values": ["s9tIpPmhxdiuNkHMEWNpYim8S8Y"]
}
```

6.1.1.2. ETSI Trusted List

Type: `"etsi_tl"`
Value: The identifier of a Trusted List as specified in ETSI TS 119 612 [ETSI.TL]. An ETSI Trusted List contains references to other Trusted Lists, creating a list of trusted lists, or entries for Trust Service Providers with corresponding service description and X.509 Certificates. The trust chain of a matching Credential MUST contain at least one X.509 Certificate that matches one of the entries of the Trusted List or its cascading Trusted Lists.

Below is a non-normative example of such an entry of type `etsi_tl`:

```
{
  "type": "etsi_tl",
  "values": ["https://lotl.example.com"]
}
```

6.1.1.3. OpenID Federation

Type: `"openid_fed"`
Value: The `Entity Identifier` as defined in Section 1 of [OpenID.Federation] that is bound to an entity in a federation. While this Entity Identifier could be any entity in that ecosystem, this entity would usually have the Entity Configuration of a Trust Anchor. A valid trust path, including the given Entity Identifier, must be constructible from a matching credential.

Below is a non-normative example of such an entry of type `openid_fed`:

```
{
  "type": "openid_fed",
  "values": ["https://trustanchor.example.com"]
}
```

6.2. Credential Set Query

A Credential Set Query is an object representing a request for one or more credentials to satisfy a particular use case with the Verifier.

Each entry in `credential_sets` MUST be an object with the following properties:

`options` REQUIRED: A non-empty array, where each value in the array is a list of Credential Query identifiers representing one set of Credentials that satisfies the use case. The value of each element in the `options` array is an array of identifiers which reference elements in `credentials`.
`required` OPTIONAL. A boolean which indicates whether this set of Credentials is required to satisfy the particular use case at the Verifier. If omitted, the default value is `true`.
`purpose` OPTIONAL. A string, number or object specifying the purpose of the query. This specification does not define a specific structure or specific values for this property. The purpose is intended to be used by the Verifier to communicate the reason for the query to the Wallet. The Wallet MAY use this information to show the user the reason for the request.

6.3. Claims Query

Each entry in `claims` MUST be an object with the following properties:

`id`: REQUIRED if `claim_sets` is present in the Credential Query; OPTIONAL otherwise. A string identifying the particular claim. The value MUST be a non-empty string consisting of alphanumeric, underscore (_) or hyphen (-) characters. Within the particular `claims` array, the same `id` MUST NOT be present more than once.
`path`: REQUIRED The value MUST be a non-empty array representing a claims path pointer that specifies the path to a claim within the Verifiable Credential, as defined in Section 7.
`values`: OPTIONAL. An array of strings, integers or boolean values that specifies the expected values of the claim. If the `values` property is present, the Wallet SHOULD return the claim only if the type and value of the claim both match exactly for at least one of the elements in the array. Details of the processing rules are defined in Section 6.3.1.1.

6.3.1. Selecting Claims and Credentials

The following section describes the logic that applies for selecting claims and for selecting credentials.

6.3.1.1. Selecting Claims

The following rules apply for selecting claims via `claims` and `claim_sets`:

- If `claims` is absent, the Verifier is requesting no claims that are selectively disclosable; the Wallet MUST return only the claims that are mandatory to present (e.g., SD-JWT and Key Binding JWT for a Credential of format IETF SD-JWT VC).
- If `claims` is present, but `claim_sets` is absent, the Verifier requests all claims listed in `claims`.
- If both `claims` and `claim_sets` are present, the Verifier requests one combination of the claims listed in `claim_sets`. The order of the options conveyed in the `claim_sets` array expresses the Verifier's preference for what is returned; the Wallet SHOULD return the first option that it can satisfy. If the Wallet cannot satisfy any of the options, it MUST NOT return any claims.
- `claim_sets` MUST NOT be present if `claims` is absent.

When a Claims Query contains a restriction on the values of a claim, the Wallet SHOULD NOT return the claim if its value does not match at least one of the elements in `values` i.e., the claim should be treated the same as if it did not exist in the Credential. Implementing this restriction may not be possible in all cases, for example, if the Wallet does not have access to the claim value before presentation or user consent or if another component routing the request to the Wallet does not have access to the claim value. It is ultimately up to the Wallet and/or the End-User if the value matching request is followed. Therefore, Verifiers MUST treat restrictions expressed using `values` as a best-effort way to improve user privacy, but MUST NOT rely on it for security checks.

The purpose of the `claim_sets` syntax is to provide a way for a verifier to describe alternative ways a given credential can satisfy the request. The array ordering expresses the Verifier's preference for how to fulfill the request. The first element in the array is the most preferred and the last element in the array is the least preferred. Verifiers SHOULD use the principle of least information disclosure to influence how they order these options. For example, a proof of age request should prioritize requesting an attribute like `age_over_18` over an attribute like `birth_date`. The `claim_sets` syntax is not intended to define options the user can choose from, see Section 6.3.1.3 for more information. The Wallet is recommended to return the first option it can satisfy since that is the preferred option from the Verifier. However, there can be reasons to deviate. Non-exhaustive examples of such reasons are:

- scenarios where the Verifier did not order the options according to least information disclosure
- operational reasons why returning a different option than the first option has UX benefits for the Wallet.

If the Wallet cannot deliver all claims requested by the Verifier according to these rules, it MUST NOT return the respective Credential.

For Credential Formats that do not support selective disclosure, the case of both `claims` and `claim_sets` being absent is interpreted as requesting a presentation of the "full credential" since all claims are mandatory to present.

6.3.1.2. Selecting Credentials

The following rules apply for selecting Credentials via `credentials` and `credential_sets`:

- If `credential_sets` is not provided, the Verifier requests presentations for all Credentials in `credentials` to be returned.
- Otherwise, the Verifier requests presentations of Credentials to be returned satisfying
  - all of the Credential Set Queries in the `credential_sets` array where the `required` attribute is true or omitted, and
  - optionally, any of the other Credential Set Queries.

To satisfy a Credential Set Query, the Wallet MUST return presentations of a set of Credentials that match to one of the `options` inside the Credential Set Query.

Credentials not matching the respective constraints expressed within `credentials` MUST NOT be returned, i.e., they are treated as if they would not exist in the Wallet.

If the Wallet cannot deliver all non-optional Credentials requested by the Verifier according to these rules, it MUST NOT return any Credential(s).

6.3.1.3. User Interface Considerations

While this specification provides the mechanisms for requesting different sets of claims and Credentials, it does not define details about the user interface of the Wallet, for example, if and how users can select which combination of Credentials to present. However, it is typically expected that the Wallet presents the End-User with a choice of which Credential(s) to present if multiple of the sets of Credentials in `options` can satisfy the request.

6.3.1.4. Security Considerations

While the Verifier can specify various constraints both on the claims level and the Credential level as shown above, it MUST NOT rely on the Wallet to enforce these constraints. The Wallet is not controlled by the Verifier and the Verifier MUST perform its own security checks on the returned Credentials and presentations.

7. Claims Path Pointer

A claims path pointer is a pointer into the Verifiable Credential, identifying one or more claims. A claims path pointer MUST be a non-empty array of strings, nulls and non-negative integers. A claims path pointer can be processed, which means it is applied to a credential. The results of processing are the referenced claims.

7.1. Semantics for JSON-based credentials

This section defines the semantics of a claims path pointer when applied to a JSON-based credential.

A string value indicates that the respective key is to be selected, a null value indicates that all elements of the currently selected array(s) are to be selected; and a non-negative integer indicates that the respective index in an array is to be selected. The path is formed as follows:

Start with an empty array and repeat the following until the full path is formed.

- To address a particular claim within an object, append the key (claim name) to the array.
- To address an element within an array, append the index to the array (as a non-negative, 0-based integer).
- To address all elements within an array, append a null value to the array.

7.1.1. Processing

In detail, the array is processed from left to right as follows:

1. Select the root element of the Credential, i.e., the top-level JSON object.
2. Process the query of the claims path pointer array from left to right:
    1. If the component is a string, select the element in the respective key in the currently selected element(s). If any of the currently selected element(s) is not an object, abort processing and return an error. If the key does not exist in any element currently selected, remove that element from the selection.
    2. If the component is null, select all elements of the currently selected array(s). If any of the currently selected element(s) is not an array, abort processing and return an error.
    3. If the component is a non-negative integer, select the element at the respective index in the currently selected array(s). If any of the currently selected element(s) is not an array, abort processing and return

an error. If the index does not exist in a selected array, remove that array from the selection.

4. If the component is anything else, abort processing and return an error.

3. If the set of elements currently selected is empty, abort processing and return an error.

The result of the processing is the set of selected JSON elements.

7.2. Semantics for ISO mdoc-based credentials

This section defines the semantics of a claims path pointer when applied to a credential in ISO mdoc format.

A claims path pointer into an mdoc contains two elements of type string. The first element refers to a namespace and the second element refers to a data element identifier.

7.2.1. Processing

In detail, the array is processed as follows:

1. If the claims path pointer does not contain exactly two components or one of the components is not a string abort processing and return an error.
2. Select the namespace referenced by the first component. If the namespace does not exist in the mdoc abort processing and return an error.
3. Select the data element referenced by the second component. If the data element does not exist in the credential abort processing and return an error.

The result of the processing is the selected data element value as CBOR data item.

7.3. Claims Path Pointer Example

The following shows a non-normative, simplified example of a JSON-based Credential:

```
{
  "name": "Arthur Dent",
  "address": {
    "street_address": "42 Market Street",
    "locality": "Milliways",
    "postal_code": "12345"
  },
  "degrees": [
    {
      "type": "Bachelor of Science",
      "university": "University of Betelgeuse"
    },
    {
      "type": "Master of Science",
      "university": "University of Betelgeuse"
    }
  ],
  "nationalities": ["British", "Betelgeusian"]
}
```

The following shows examples of claims path pointers and the respective selected claims:

- `["name"]`: The claim `name` with the value `Arthur Dent` is selected.
- `["address"]`: The claim `address` with its sub-claims as the value is selected.
- `["address", "street_address"]`: The claim `street_address` with the value `42 Market Street` is selected.
- `["degrees", null, "type"]`: All `type` claims in the `degrees` array are selected.

- `["nationalities", 1]`: The second nationality is selected.

7.4. DCQL Examples

The following is a non-normative example of a DCQL query that requests a Verifiable Credential of the format `dc+sd-jwt` with a type value of `https://credentials.example.com/identity_credential` and the claims `last_name`, `first_name`, and `address.street_address`:

```
{
  "credentials": [
    {
      "id": "my_credential",
      "format": "dc+sd-jwt",
      "meta": {
        "vct_values": [ "https://credentials.example.com/identity_credential" ]
      },
      "claims": [
          {"path": ["last_name"]},
          {"path": ["first_name"]},
          {"path": ["address", "street_address"]}
      ]
    }
  ]
}
```

Additional, more complex examples can be found in Appendix C.

8. Response

A VP Token is only returned if the corresponding Authorization Request contained a `dcql_query` parameter, a `presentation_definition` parameter, a `presentation_definition_uri` parameter, or a `scope` parameter representing a Presentation Definition Section 5.

A VP Token can be returned in the Authorization Response or the Token Response depending on the Response Type used. See Section 5.7 for more details.

If the Response Type value is `vp_token`, the VP Token is returned in the Authorization Response. When the Response Type value is `vp_token id_token` and the `scope` parameter contains `openid`, the VP Token is returned in the Authorization Response alongside a Self-Issued ID Token as defined in [SIOPv2].

If the Response Type value is `code` (Authorization Code Grant Type), the VP Token is provided in the Token Response.

The expected behavior is summarized in the following table:

response_type parameter value	Response containing the VP Token
`vp_token`	Authorization Response
`vp_token id_token`	Authorization Response
`code`	Token Response

Table 1

Table 1: OpenID for Verifiable Presentations `response_type` values

The behavior with respect to the VP Token is unspecified for any other individual Response Type value, or a combination of Response Type values.

8.1. Response Parameters

When a VP Token is returned, the respective response includes the following parameters:

`vp_token`: REQUIRED. The structure of this parameter depends on the query language used to request the presentations in the Authorization Request:

- If DCQL was used, this is a JSON-encoded object containing entries where: the key is the `id` value used for a Credential Query in the DCQL query; and the value is an array of one or more Verifiable Presentations that match the respective Credential Query. When `multiple` is omitted, or set to `false`, the array MUST contain only one Verifiable Presentation. There MUST NOT be any entry in the JSON-encoded object for optional Credential Queries when there are no matching Credentials for the respective Credential Query. Each Verifiable Presentation is represented as a string or object, depending on the format as defined in Appendix B. The same rules as above apply for encoding the Verifiable Presentations.
- In case [DIF.PresentationExchange] was used, it is a string or JSON object that MUST contain a single Verifiable Presentation or an array of strings and JSON objects each of them containing a Verifiable Presentation. Each Verifiable Presentation MUST be represented as a string (that is a base64url-encoded value) or a JSON object depending on a format as defined in Appendix B. When a single Verifiable Presentation is returned, the array syntax MUST NOT be used. If Appendix B defines a rule for encoding the respective Credential format in the Credential Response, this rules MUST also be followed when encoding Credentials of this format in the `vp_token` response parameter. Otherwise, this specification does not require any additional encoding when a Credential format is already represented as a JSON object or a string.

`presentation_submission`: REQUIRED if [DIF.PresentationExchange] was used for the request; MUST NOT be used otherwise. The `presentation_submission` element as defined in [DIF.PresentationExchange]. It contains mappings between the requested Verifiable Credentials and where to find them within the returned VP Token. This is expressed via elements in the `descriptor_map` array, known as Input Descriptor Mapping Objects. These objects contain a field called `path`, which, for this specification, MUST have the value `$` (top level root path) when only one Verifiable Presentation is contained in the VP Token, and MUST have the value `$[n]` (indexed path from root) when there are multiple Verifiable Presentations, where `n` is the index to select. Additional parameters can be defined by Credential Formats, see Appendix B for details.

Other parameters, such as `state` or `code` (from [RFC6749]), or `id_token` (from [OpenID.Core]), and `iss` (from [RFC9207]) can be included in the response as defined in the respective specifications. `state` values MUST only contain ASCII URL safe characters (uppercase and lowercase letters, decimal digits, hyphen, period, underscore, and tilde). For the implementation considerations of a `state` parameter, see Section 13.3.

If [DIF.PresentationExchange] was used for the request, the `presentation_submission` element MUST be included as a separate response parameter alongside the VP token. Clients MUST ignore any `presentation_submission` element included inside a Verifiable Presentation.

Including the `presentation_submission` parameter as a separate response parameter allows the Wallet to provide the Verifier with additional information about the format and structure in advance of the processing of the VP Token, and can be used even with the Credential formats that do not allow for the direct inclusion of `presentation_submission` parameters inside a Credential itself.

Additional response parameters MAY be defined and used, as described in [RFC6749]. The Client MUST ignore any unrecognized parameters.

The following is a non-normative example of an Authorization Response when the Response Type value in the Authorization Request was `vp_token`:

```
HTTP/1.1 302 Found
Location: https://client.example.org/cb#
  presentation_submission=...
  &vp_token=...
```

8.1.1. Examples (DCQL)

The following is a non-normative example of the contents of a VP Token containing a single Verifiable Presentation in the SD-JWT VC format after a request using DCQL like the one shown in Section 7.4 (shortened for brevity):

```
{
  "my_credential": ["eyJhbGci...QMA"]
}
```

The following is a non-normative example of the contents of a VP Token containing multiple Verifiable Presentations in the SD-JWT VC format when the Credential Query has `multiple` set to `true` (shortened for brevity):

```
{
  "my_credential": ["eyJhbGci...QMA", "eyJhbGci...QMA", ...]
}
```

8.1.2. Examples (Presentation Exchange)

The following is a non-normative example of a VP Token containing a single Verifiable Presentation after a request using [DIF.PresentationExchange]:

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1"
  ],
  "type": [
    "VerifiablePresentation"
  ],
  "verifiableCredential": [
    {
      "@context": [
        "https://www.w3.org/2018/credentials/v1",
        "https://www.w3.org/2018/credentials/examples/v1"
      ],
      "id": "https://example.com/credentials/1872",
      "type": [
        "VerifiableCredential",
        "IDCardCredential"
      ],
      "issuer": {
        "id": "did:example:issuer"
      },
      "issuanceDate": "2010-01-01T19:23:24Z",
      "credentialSubject": {
        "given_name": "Fredrik",
        "family_name": "Strömberg",
        "birthdate": "1949-01-22"
      },
      "proof": {
        "type": "Ed25519Signature2018",
        "created": "2021-03-19T15:30:15Z",
        "jws": "eyJhb...JQdBw",
        "proofPurpose": "assertionMethod",
        "verificationMethod": "did:example:issuer#keys-1"
      }
    }
  ],
  "id": "ebc6f1c2",
  "holder": "did:example:holder",
  "proof": {
    "type": "Ed25519Signature2018",
    "created": "2021-03-19T15:30:15Z",
    "challenge": "n-0S6_WzA2Mj",
    "domain": "https://client.example.org/cb",
    "jws": "eyJhbG...IAoDA",
    "proofPurpose": "authentication",
    "verificationMethod": "did:example:holder#key-1"
  }
}
```


The following is a non-normative example of a `presentation_submission` parameter sent alongside a VP Token in the example above. It corresponds to the second Presentation Definition example in Section 5.4:

```
{
  "id": "Presentation example 1",
  "definition_id": "Example with selective disclosure",
  "descriptor_map": [
    {
      "id": "ID card with constraints",
      "format": "ldp_vp",
      "path": "$",
      "path_nested": {
        "format": "ldp_vc",
        "path": "$.verifiableCredential[0]"
      }
    }
  ]
}
```

A `descriptor_map` element MUST contain a `path_nested` parameter referring to the actual Credential carried in the respective Verifiable Presentation.

The following is a non-normative example of a VP Token containing multiple Verifiable Presentations:

```
[
  {
    "@context": [
      "https://www.w3.org/2018/credentials/v1"
    ],
    "type": [
      "VerifiablePresentation"
    ],
    "verifiableCredential": [
      {
        "@context": [
          "https://www.w3.org/2018/credentials/v1",
          "https://www.w3.org/2018/credentials/examples/v1"
        ],
        "id": "https://example.com/credentials/1872",
        "type": [
          "VerifiableCredential",
          "IDCardCredential"
        ],
        "issuer": {
          "id": "did:example:issuer"
        },
        "issuanceDate": "2010-01-01T19:23:24Z",
        "credentialSubject": {
          "given_name": "Fredrik",
          "family_name": "Strömberg",
          "birthdate": "1949-01-22"
        },
        "proof": {
          "type": "Ed25519Signature2018",
          "created": "2021-03-19T15:30:15Z",
          "jws": "eyJhb...IAoDA",
          "proofPurpose": "assertionMethod",
          "verificationMethod": "did:example:issuer#keys-1"
        }
      }
    ],
    "id": "ebc6f1c2",
    "holder": "did:example:holder",
    "proof": {
```

```
      "type": "Ed25519Signature2018",
      "created": "2021-03-19T15:30:15Z",
      "challenge": "n-0S6_WzA2Mj",
      "domain": "https://client.example.org/cb",
      "jws": "eyJhb...JQdBw",
      "proofPurpose": "authentication",
      "verificationMethod": "did:example:holder#key-1"
    }
  },
  "eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImRjK3NkLWp3dCIsICJraWQiOiAiZG9jLXNp
   Z25lci0wNS0yNS0yMDIyIn0.eyJfc2QiOiBbIjA5dktySk1PbHlUV00wc2pwdV9wZE9C
   VkJRMk0xeTNLaHBINTE1blhrcFkiLCAiMnJzakdiYUMwa3k4bVQwcEpyUGlvV1RxMF9k
   YXcxc1g3NnBvVWxnQ3diSSIsICJFa084ZGhXMGRIRUpidlVIbEVfVkNldUM5dVJFTE9p
   ZUxaaGg3WGJVVHRBIiwgIklsRHpJS2VpWmREd3BxcEs2WmZieXBoRnZ6NUZnbldhLXNO
   NndxUVhDaXciLCAiSnpZakg0c3ZsaUgwUjNQeUVNZmVadTZKdDY5dTVxZWhabzdGN0VQ
   WWxTRSIsICJQb3JGYnBLdVZ1Nnh5bUphZ3ZrRnNGWEFiUm9jMkpHbEFVQTJCQTRvN2NJ
   IiwgIlRHZjRvTGJnd2Q1SlFhSHlLVlFaVTlVZEdFMHc1cnREc3JaemZVYW9tTG8iLCAi
   amRyVEU4WWNiWTRFaWZ1Z2loaUFlX0JQZWt4SlFaSUNlaVVRd1k5UXF4SSIsICJqc3U5
   eVZ1bHdRUWxoRmxNXzNKbHpNYVNGemdsaFFHMERwZmF5UXdMVUs0Il0sICJpc3MiOiAi
   aHR0cHM6Ly9leGFtcGxlLmNvbS9pc3N1ZXIiLCAiaWF0IjogMTY4MzAwMDAwMCwgImV4
   cCI6IDE4ODMwMDAwMDAsICJ2Y3QiOiAiaHR0cHM6Ly9jcmVkZW50aWFscy5leGFtcGxl
   LmNvbS9pZGVudGl0eV9jcmVkZW50aWFsIiwgIl9zZF9hbGciOiAic2hhLTI1NiIsICJj
   bmYiOiB7Imp3ayI6IHsia3R5IjogIkVDIiwgImNydiI6ICJQLTI1NiIsICJ4IjogIlRD
   QUVSMTladnUzT0hGNGo0VzR2ZlNWb0hJUDFJTGlsRGxzN3ZDZUdlbWMiLCAieSI6ICJa
   eGppV1diWk1RR0hWV0tWUTRoYlNJaXJzVmZ1ZWNDRTZ0NGpUOUYySFpRIn19fQ.BfCz7
   bTCeExVLQUXuOUNSlPTD_xMraWG1nb7hPx-jbHXNY9hnu6sl3idGcqHF8LIKWD9OOvO-
   oJMS0XfLlBwsg~WyJRZ19PNjR6cUF4ZTQxMmExMDhpcm9BIiwgImFkZHJlc3MiLCB7In
   N0cmVldF9hZGRyZXNzIjogIjEyMyBNYWluIFN0IiwgImxvY2FsaXR5IjogIkFueXRvd2
   4iLCAicmVnaW9uIjogIkFueXN0YXRlIiwgImNvdW50cnkiOiAiVVMifV0~eyJhbGciOi
   AiRVMyNTYiLCAidHlwIjogImtiK2p3dCJ9.eyJub25jZSI6ICIxMjM0NTY3ODkwIiwgI
   mF1ZCI6ICJodHRwczovL2V4YW1wbGUuY29tL3ZlcmlmaWVyIiwgImlhdCI6IDE3MDk1N
   zYwMzcsICJzZF9oYXNoIjogIkQtaGVBamp0Q2Z4bkhjTDJyckZtNXNreTBjYXlZakhYd
   zd3U2dwU3N2bzQifQ.ytKc24j5CGv8So1Iyo5LhPROWCpg-p4YwtyNCt0L1QsSh7MPhS
   mUtBzlOgb3xvMde0uk3MUpLLUVO96zlrp6zg"
]
```

The following is a non-normative example of a `presentation_submission` parameter sent alongside a VP Token in the example above. It does not correspond to any Presentation Definition example in this specification:

```
{
  "id": "Presentation example 2",
  "definition_id": "Example with multiple VPs",
  "descriptor_map": [
    {
      "id": "ID Card with constraints",
      "format": "ldp_vp",
      "path": "$[0]",
      "path_nested": {
        "format": "ldp_vc",
        "path": "$.verifiableCredential[0]"
      }
    },
    {
      "id": "Example credential disclosing only address",
      "format": "dc+sd-jwt",
      "path": "$[1]"
    }
  ]
}
```

8.2. Response Mode "direct_post"

The Response Mode `direct_post` allows the Wallet to send the Authorization Response to an endpoint controlled by the Verifier via an HTTP POST request.

It has been defined to address the following use cases:

- Verifier and Wallet are located on different devices; thus, the Wallet cannot send the Authorization Response to the Verifier using a redirect.
- The Authorization Response size exceeds the URL length limits of user agents, so flows relying only on redirects (such as Response Mode `fragment`) cannot be used. In those cases, the Response Mode `direct_post` is the way to convey the Verifiable Presentations to the Verifier without the need for the Wallet to have a backend.

The Response Mode is defined in accordance with [OAuth.Responses] as follows:

`direct_post`: In this mode, the Authorization Response is sent to the Verifier using an HTTP POST request to an endpoint controlled by the Verifier. The Authorization Response MUST be encoded in the request body using the format defined by the `application/x-www-form-urlencoded` HTTP content type. The parameters in the request body MUST all be encoded using UTF-8. The verifier can request that the wallet redirects the user to the verifier using the response as defined below.

The following new Authorization Request parameter is defined to be used in conjunction with Response Mode `direct_post`:

`response_uri`: REQUIRED when the Response Mode `direct_post` is used. The URL to which the Wallet MUST send the Authorization Response using an HTTP POST request as defined by the Response Mode `direct_post`. The Response URI receives all Authorization Response parameters as defined by the respective Response Type. When the `response_uri` parameter is present, the `redirect_uri` Authorization Request parameter MUST NOT be present. If the `redirect_uri` Authorization Request parameter is present when the Response Mode is `direct_post`, the Wallet MUST return an `invalid_request` Authorization Response error. The `response_uri` value MUST be a value that the client would be permitted to use as `redirect_uri` when following the rules defined in Section 5.10.

Note: When the specification text refers to the usage of Redirect URI in the Authorization Request, that part of the text also applies when Response URI is used in the Authorization Request with Response Mode `direct_post`.

Note: The Verifier's component providing the user interface (Frontend) and the Verifier's component providing the Response URI need to be able to map authorization requests to the respective authorization responses. The Verifier MAY use the `state` Authorization Request parameter to add appropriate data to the Authorization Response for that purpose, for details see Section 13.4.

Additional request parameters MAY be defined and used with the Response Mode `direct_post`. The Wallet MUST ignore any unrecognized parameters.

The following is a non-normative example of the payload of a Request Object with Response Mode `direct_post`:

```
{
  "client_id": "redirect_uri:https://client.example.org/post",
  "response_uri": "https://client.example.org/post",
  "response_type": "vp_token",
  "response_mode": "direct_post",
  "presentation_definition": {...},
  "nonce": "n-0S6_WzA2Mj",
  "state": "eyJhb...6-sVA"
}
```

The following non-normative example of an Authorization Request refers to the Authorization Request Object from above through the `request_uri` parameter. The Authorization Request can be displayed to the End-User either directly (as a link) or as a QR Code:

```
https://wallet.example.com?
  client_id=https%3A%2F%2Fclient.example.org%2Fcb
  &request_uri=https%3A%2F%2Fclient.example.org%2F567545564
```

The following is a non-normative example of the Authorization Response that is sent via an HTTP POST request to the Verifier's Response URI:

```
POST /post HTTP/1.1
Host: client.example.org
Content-Type: application/x-www-form-urlencoded

  presentation_submission=...&
  vp_token=...&
  state=eyJhb...6-sVA
```

The following is a non-normative example of an Authorization Error Response that is sent as an HTTP POST request to the Verifier's Response URI:

```
POST /post HTTP/1.1
Host: client.example.org
Content-Type: application/x-www-form-urlencoded

  error=invalid_request&
  error_description=unsupported%20client_id_scheme&
  state=eyJhb...6-sVA
```

If the Response URI has successfully processed the Authorization Response or Authorization Error Response, it MUST respond with an HTTP status code of 200 with `Content-Type` of `application/json` and a JSON object in the response body.

The following new parameter is defined for use in the JSON object returned from the Response Endpoint to the Wallet:

`redirect_uri`: OPTIONAL. String containing a URI. When this parameter is present the Wallet MUST redirect the user agent to this URI. This allows the Verifier to continue the interaction with the End-User on the device where the Wallet resides after the Wallet has sent the Authorization Response to the Response URI. It can be used by the Verifier to prevent session fixation (Section 14.2) attacks. The Response URI MAY return the `redirect_uri` parameter in response to successful Authorization Responses or for Error Responses.

Additional response parameters MAY be defined and used. The Wallet MUST ignore any unrecognized parameters.

Note: Response Mode `direct_post` without the `redirect_uri` could be less secure than Response Modes with redirects. For details, see (Section 14.2).

The value of the redirect URI is an absolute URI as defined by [RFC3986] Section 4.3 and is chosen by the Verifier. The Verifier MUST include a fresh, cryptographically random value in the URL. This value is used to ensure only the receiver of the redirect can fetch and process the Authorization Response. The value can be added as a path component, as a fragment or as a parameter to the URL. It is RECOMMENDED to use a cryptographic random value of 128 bits or more. For implementation considerations see Section 13.4.

The following is a non-normative example of the response from the Verifier to the Wallet upon receiving the Authorization Response at the Response URI (using a `response_code` parameter from Section 13.4):

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

{
  "redirect_uri":
"https://client.example.org/cb#response_code=091535f699ea575c7937fa5f0f454aee"
}
```

If the response does not contain the `redirect_uri` parameter, the Wallet is not required to perform any further steps.

Note: In the Response Mode `direct_post` or `direct_post.jwt`, the Wallet can change the UI based on the Verifier's callback to the Wallet following the submission of the Authorization Response.

Additional parameters MAY be defined and used in the response from the Response Endpoint to the Wallet. The Wallet MUST ignore any unrecognized parameters.

8.3. Signed and/or Encrypted Responses

This section defines how an Authorization Response containing a VP Token can be signed and/or encrypted at the application level when the Response Type value is `vp_token` or `vp_token id_token`. Encrypting the Authorization Response can prevent personal data in the Authorization Response from leaking, when the Authorization Response is returned through the front channel (e.g., the browser).

To sign, encrypt, or both sign and encrypt the Authorization Response, implementations MUST use the JWT Secured Authorization Response Mode for OAuth 2.0 (JARM) [JARM], and when only encrypting, the JARM extension described below.

This specification also defines how to encrypt an unsigned Authorization Response by adapting the mechanisms defined in [JARM]. The JSON containing the Authorization Response parameters can be encrypted as the payload of the JWE.

The advantage of an encrypted but not signed Authorization Response is that it prevents the signing key from being used as a correlation factor. It can also be a challenge to establish trust in the signing key to ensure authenticity. For security considerations with encrypted but unsigned responses, see Section 14.5.

If the JWT is only a JWE, the following processing rules MUST be followed:

- `iss`, `exp` and `aud` MUST be omitted in the JWT Claims Set of the JWE, and the processing rules as per [JARM] Section 2.4 related to these claims do not apply.
- The processing rules as per [JARM] Section 2.4 related to JWS processing MUST be ignored.

Note that for the ECDH JWE algorithms (from section 4.6 of [RFC7518]), the `apu` and `apv` values are inputs into the key derivation process that is used to derive the content encryption key. Regardless of algorithm used, the values are always part of the AEAD tag computation so will still be bound to the encrypted response. The following is a non-normative example of the payload of a JWT used in an Authorization Response that is encrypted and not signed:

```
{
  "vp_token": "eyJhb...YMetA",
  "presentation_submission": {
    "definition_id": "example_jwt_vc",
    "id": "example_jwt_vc_presentation_submission",
    "descriptor_map": [
      {
        "id": "id_credential",
        "path": "$",
        "format": "jwt_vp_json",
        "path_nested": {
          "path": "$.vp.verifiableCredential[0]",
          "format": "jwt_vc"
        }
      }
    ]
  }
}
```

The JWT response document MUST include the `vp_token` and, if [DIF.PresentationExchange] was used in the request, the `presentation_submission` parameters as defined in Section 8.1.

The key material used for encryption and signing SHOULD be determined using existing metadata mechanisms.

To obtain Verifier's public key for the input to the key agreement to encrypt the Authorization Response, the Wallet MUST use `jwks` claim within the `client_metadata` request parameter, or within the metadata defined in the Entity Configuration when [OpenID.Federation] is used, or other mechanisms.

To sign the Authorization Response, the Wallet MUST use a private key that corresponds to a public key made available in its metadata.

Note: For encryption, implementers have a variety of options available through JOSE, including the use of Hybrid Public Key Encryption (HPKE) as detailed in [I-D.ietf-jose-hpke-encrypt].

8.3.1. Response Mode "direct_post.jwt"

This specification also defines a new Response Mode `direct_post.jwt`, which allows for JARM to be used with Response Mode `direct_post` defined in Section 8.2.

The Response Mode `direct_post.jwt` causes the Wallet to send the Authorization Response using an HTTP POST request instead of redirecting back to the Verifier as defined in Section 8.2. The Wallet adds the `response` parameter containing the JWT as defined in Section 4.1. of [JARM] and Section 8.3 in the body of an HTTP POST request using the `application/x-www-form-urlencoded` content type. The names and values in the body MUST be encoded using UTF-8.

If a Wallet is unable to generate a JARM response, it MAY send an error response without using JARM as per Section 8.2.

The following is a non-normative example of a response using the `presentation_submission` and `vp_token` values from Appendix B.1.1. (line breaks for display purposes only):

```
POST /post HTTP/1.1
Host: client.example.org
Content-Type: application/x-www-form-urlencoded

  response=eyJra...9t2LQ
```

The following is a non-normative example of the payload of the JWT used in the example above before base64url encoding and signing:

```
{
  "iss": "did:example:ebfeb1f712ebc6f1c276e12ec21",
  "aud": "redirect_uri:https://client.example.org/cb",
  "exp": 1573029723,
  "vp_token": "eyJhb...YMetA",
  "presentation_submission": {
    "definition_id": "example_jwt_vc",
    "id": "example_jwt_vc_presentation_submission",
    "descriptor_map": [
      {
        "id": "id_credential",
        "path": "$",
        "format": "jwt_vp_json",
        "path_nested": {
          "path": "$.vp.verifiableCredential[0]",
          "format": "jwt_vc"
        }
      }
    ]
  }
}
```

8.4. Transaction Data

The transaction data mechanism enables a binding between the user's identification/authentication and the user's authorization, for example to complete a payment transaction, or to sign specific document(s) using QES (Qualified Electronic Signatures). This is achieved by signing the transaction data used for user authorization with the user-controlled key used for proof of possession of the Credential being presented as a means for user identification/authentication.

The Wallet that received the `transaction_data` parameter in the request MUST include a representation or reference to the data in the respective credential presentation. How this is done is transaction data type specific. Credential Formats can give recommendations of how to handle transaction data, such as those in Appendix B.

If the Wallet does not support `transaction_data` parameter, it MUST return an error upon receiving a request that includes it.

8.5. Error Response

The error response follows the rules as defined in [RFC6749], with the following additional clarifications:

`invalid_scope`:

- Requested scope value is invalid, unknown, or malformed.

`invalid_request`:

- The request contains more than one out of the following three options to communicate a requested Credential: a `presentation_definition` parameter, a `presentation_definition_uri` parameter, or a scope value representing a Presentation Definition.
- The request uses the `vp_token` Response Type but does not request a Credential using any of the three options
- Requested Presentation Definition does not conform to the DIF PEv2 specification [DIF.PresentationExchange].
- The Wallet does not support the Client Identifier Scheme passed in the Authorization Request.
- The Client Identifier passed in the request did not belong to its Client Identifier scheme, or requirements of a certain scheme was violated, for example an unsigned request was sent with Client Identifier scheme `https`.

`invalid_client`:

- `client_metadata` parameter defined in Section 5.1 is present, but the Wallet recognizes Client Identifier and knows metadata associated with it.
- Verifier's pre-registered metadata has been found based on the Client Identifier, but `client_metadata` parameter is also present.

`access_denied`:

- The Wallet did not have the requested Credentials to satisfy the Authorization Request.
- The End-User did not give consent to share the requested Credentials with the Verifier.
- The Wallet failed to authenticate the End-User.

This document also defines the following additional error codes and error descriptions:

`vp_formats_not_supported`:

- The Wallet does not support any of the formats requested by the Verifier, such as those included in the `vp_formats` registration parameter.

`invalid_presentation_definition_uri`:

- The Presentation Definition URL cannot be reached.

`invalid_presentation_definition_reference`:

- The Presentation Definition URL can be reached, but the specified `presentation_definition` cannot be found at the URL.

`invalid_request_uri_method`:

- The value of the `request_uri_method` request parameter is neither `get` nor `post` (case-sensitive).

`invalid_transaction_data`:

- any of the following is true for at least one object in the `transaction_data` structure:
  - contains an unknown or unsupported transaction data type value,
  - is an object of known type but containing unknown fields,
  - contains fields of the wrong type for the transaction data type,

    - contains fields with invalid values for the transaction data type, or
    - is missing required fields for the transaction data type.
    - the credential_ids does not match
    - the referenced Credential(s) are not available in the Wallet

`wallet_unavailable`:

- The Wallet appears to be unavailable and therefore unable to respond to the request. It can be useful in situations where the user agent cannot invoke the Wallet and another component receives the request while the End-User wishes to continue the journey on the Verifier website. For example, this applies when using claimed HTTPS URIs handled by the Wallet provider in case the platform cannot or does not translate the URI into a platform intent to invoke the Wallet. In this case, the Wallet provider would return the Authorization Error Response to the Verifier and might redirect the user agent back to the Verifier website.

8.6. VP Token Validation

Verifiers MUST validate the VP Token in the following manner:

1. Validate the format of the VP Token as defined in Section 8.1 and verify the contents depending on the language used in the Authorization Request:
    1. If DCQL was used, ensure that the set of VPs returned satisfies all required Credential Sets (and optionally other Credential Sets).
    2. If [DIF.PresentationExchange] was used, determine the number of VPs returned in the VP Token and identify in which VP which requested VC is included, using the Input Descriptor Mapping Object(s) in the Presentation Submission.
2. Validate the integrity, authenticity, and Holder Binding of any Verifiable Presentation provided in the VP Token according to the rules of the respective Presentation format. See Section 14.1 for the checks required to prevent replay of a VP.
3. Perform the checks on the Credential(s) specific to the Credential Format (i.e., validation of the signature(s) on each VC).
4. Confirm that the returned Credential(s) meet all criteria defined in the query in the Authorization Request (e.g., Claims included in the presentation).
5. Perform the checks required by the Verifier's policy based on the set of trust requirements such as trust frameworks it belongs to (i.e., revocation checks), if applicable.

Note: Some of the processing rules of the Presentation Definition and the Presentation Submission are outlined in [DIF.PresentationExchange].

9. Wallet Invocation

The Verifier can use one of the following mechanisms to invoke a Wallet:

- Custom URL scheme as an `authorization_endpoint` (for example, `openid4vp://` as defined in Section 13.1.2)
- URL (including Domain-bound Universal Links/App link) as an `authorization_endpoint`

For a cross device flow, either of the above options MAY be presented as a QR code for the End-User to scan using a wallet or an arbitrary camera application on a user-device.

The Wallet can also be invoked from the web or a native app using the Digital Credentials API as described in Appendix A. As described in detail in Appendix A, DC API provides privacy, security (see Section 14.2), and user experience benefits (particularly in the cases where a user has multiple Wallets).

10. Wallet Metadata (Authorization Server Metadata)

This specification defines how the Verifier can determine Credential formats, proof types and algorithms supported by the Wallet to be used in a protocol exchange.

10.1. Additional Wallet Metadata Parameters

This specification defines new metadata parameters according to [RFC8414].

- `presentation_definition_uri_supported`: OPTIONAL. Boolean value specifying whether the Wallet supports the transfer of `presentation_definition` by reference, with true indicating support. If omitted, the default value is true.
- `vp_formats_supported`: REQUIRED. An object containing a list of name/value pairs, where the name is a string identifying a Credential format supported by the Wallet. Valid Credential format identifier values are defined in Appendix B. Other values may be used when defined in the profiles of this specification. The value is an object containing a parameter defined below:
  - `alg_values_supported`: OPTIONAL. An object where the value is an array of case sensitive strings that identify the cryptographic suites that are supported. Parties will need to agree upon the meanings of the values used, which may be context-specific. For specific values that can be used depending on the Credential format, see Appendix B. If `alg_values_supported` is omitted, it is unknown what cryptographic suites the wallet supports.

The following is a non-normative example of a `vp_formats_supported` parameter:

```
"vp_formats_supported": {
  "jwt_vc_json": {
    "alg_values_supported": [
      "ES256K",
      "ES384"
    ]
  },
  "jwt_vp_json": {
    "alg_values_supported": [
      "ES256K",
      "EdDSA"
    ]
  }
}
```

`client_id_schemes_supported`: OPTIONAL. Array of strings containing the values of the Client Identifier schemes that the Wallet supports. The values defined by this specification are `pre-registered` (which represents the behavior when no Client Identifier Scheme is used), `redirect_uri`, `https`, `verifier_attestation`, `did`, `x509_san_dns` and `x509_hash`. If omitted, the default value is `pre-registered`. Other values may be used when defined in the profiles of this specification.

Additional wallet metadata parameters MAY be defined and used, as described in [RFC8414]. The Verifier MUST ignore any unrecognized parameters.

10.2. Obtaining Wallet's Metadata

Verifier utilizing this specification has multiple options to obtain Wallet's metadata:

- Verifier obtains Wallet's metadata dynamically, e.g., using [RFC8414] or out-of-band mechanisms. See Section 10 for the details.
- Verifier has pre-obtained static set of Wallet's metadata. See Section 13.1.2 for the example.

11. Verifier Metadata (Client Metadata)

To convey Verifier metadata, Client metadata defined in Section 2 of [RFC7591] is used.

This specification defines how the Wallet can determine Credential formats, proof types and algorithms supported by the Verifier to be used in a protocol exchange.

11.1. Additional Verifier Metadata Parameters

This specification defines the following new Client metadata parameters according to [RFC7591], to be used by the Verifier:

`vp_formats`: REQUIRED. An object defining the formats and proof types of Verifiable Presentations and Verifiable Credentials that a Verifier supports. For specific values that can be used, see Appendix B. Deployments can extend the formats supported, provided Issuers, Holders and Verifiers all understand the new format.

Additional Verifier metadata parameters MAY be defined and used, as described in [RFC7591]. The Wallet MUST ignore any unrecognized parameters.

12. Verifier Attestation JWT

The Verifier Attestation JWT is a JWT especially designed to allow a Wallet to authenticate a Verifier in a secure and flexible manner. A Verifier Attestation JWT is issued to the Verifier by a party that wallets trust for the purpose of authentication and authorization of Verifiers. The way this trust established is out of scope of this specification. Every Verifier is bound to a public key, the Verifier MUST always present a Verifier Attestation JWT along with the proof of possession for this key. In the case of the Client Identifier Scheme `verifier_attestation`, the authorization request is signed with this key, which serves as proof of possession.

A Verifier Attestation JWT MUST contain the following claims:

- `iss`: REQUIRED. This claim identifies the issuer of the Verifier Attestation JWT. The `iss` value MAY be used to retrieve the issuer's public key. How the trust is established between Wallet and Issuer and how the public key is obtained for validating the attestation's signature is out of scope of this specification.
- `sub`: REQUIRED. The value of this claim MUST be the `client_id` of the client making the credential request.
- `iat`: OPTIONAL. A number representing the time at which the Verifier Attestation JWT was issued using the syntax defined in [RFC7519].
- `exp`: REQUIRED. A number representing the time at which the Verifier Attestation JWT expires using the syntax defined in [RFC7519]. The Wallet MUST reject any Verifier Attestation JWT with an expiration time that has passed, subject to allowable clock skew between systems.
- `nbf`: OPTIONAL. A number representing the time before which the token MUST NOT be accepted for processing.
- `cnf`: REQUIRED. This claim contains the confirmation method as defined in [RFC7800]. It MUST contain a JSON Web Key [RFC7517] as defined in Section 3.2 of [RFC7800]. This claim determines the public key for which's corresponding private key the Verifier MUST proof possession of when presenting the Verifier Attestation JWT. This additional security measure allows the Verifier to obtain a Verifier Attestation JWT from a trusted issuer and use it for a long time independent of that issuer without the risk of an adversary impersonating the Verifier by replaying a captured attestation.

Additional claims MAY be defined and used in the Verifier Attestation JWT, as described in [RFC7519]. The Wallet MUST ignore any unrecognized claims.

Verifier Attestation JWTs compliant with this specification MUST use the media type `application/verifier-attestation+jwt` as defined in Appendix D.6.1.

A Verifier Attestation JWT MUST set the `typ` JOSE header to `verifier-attestation+jwt`.

The Verifier Attestation JWT MAY be conveyed in the header of a JWS signed object (JOSE header).

This specification introduces a JOSE header, which can be used to add a JWT to such a header as follows:

- `jwt`: This JOSE header MUST contain a JWT.

In the context of this specification, such a JWT MUST set the `typ` JOSE header to `verifier-attestation+jwt`.

13. Implementation Considerations

13.1. Static Configuration Values of the Wallets

This section lists profiles of this specification that define static configuration values for Wallets and defines one set of static configuration values that can be used by the Verifier when it is unable to perform Dynamic Discovery.

13.1.1. Profiles that Define Static Configuration Values

The following is a list of profiles that define static configuration values of Wallets:

- OpenID4VC High Assurance Interoperability Profile
- JWT VC Presentation Profile
- [ISO.18013-7](https://www.iso.org/standard/82772.html)

13.1.2. A Set of Static Configuration Values bound to `openid4vp://`

The following is a non-normative example of a set of static configuration values that can be used with `vp_token` parameter as a supported Response Type, bound to a custom URL scheme `openid4vp://` as an Authorization Endpoint:

```
{
  "authorization_endpoint": "openid4vp:",
  "response_types_supported": [
    "vp_token"
  ],
  "vp_formats_supported": {
    "jwt_vp_json": {
      "alg_values_supported": ["ES256"]
    },
    "jwt_vc_json": {
      "alg_values_supported": ["ES256"]
    }
  },
  "request_object_signing_alg_values_supported": [
    "ES256"
  ]
}
```

13.2. Nested Verifiable Presentations

Current version of this document does not support presentation of a Verifiable Presentation nested inside another Verifiable Presentation, even though [DIF.PresentationExchange] specification theoretically supports this by stating that the nesting of `path_nested` objects "may be any number of levels deep".

One level of nesting `path_nested` objects is sufficient to describe a Verifiable Credential included inside a Verifiable Presentation.

13.3. State Management

The `state` parameter defined in Section 4.1.1 of [RFC6749] may be used by a verifier to link requests and responses. Also see Section 3.6 and Section 5.3.5 of [RFC6819], and [RFC9700].

When using Response Mode `direct_post`, also see Section 14.3.

13.4. Response Mode `direct_post`

The design of the interactions between the different components of the Verifier (especially Frontend and Response URI) when using Response Mode `direct_post` is at the discretion of the Verifier since it does not affect the interface between the Verifier and the Wallet.

In order to support implementers, this section outlines a possible design that fulfills the Security Considerations given in Section 14.

The design is illustrated in the following sequence diagram:

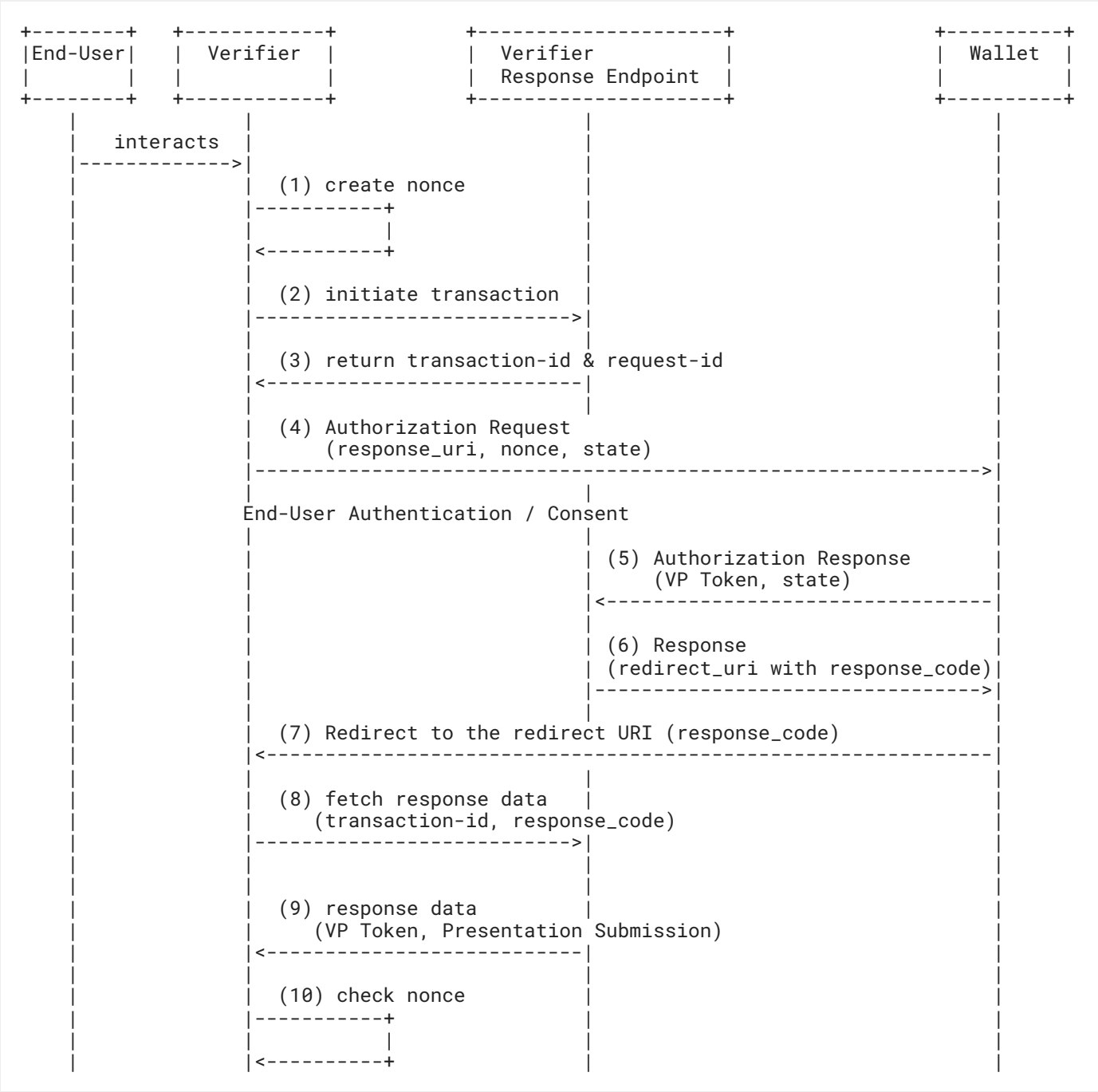

Figure 3: Reference Design for Response Mode `direct_post`

(1) The Verifier produces a nonce value by generating at least 16 fresh, cryptographically random bytes with sufficient entropy, associates it with the session and base64url encodes it.

(2) The Verifier initiates a new transaction at its Response URI.

(3) The Response URI will set up the transaction and respond with two fresh, cryptographically random numbers with sufficient entropy designated as `transaction-id` and `request-id`. Those values are used in the process to identify the authorization response (`request-id`) and to ensure only the Verifier can obtain the Authorization Response data (`transaction-id`).

(4) The Verifier then sends the Authorization Request with the `request-id` as `state` and the nonce value created in step (1) to the Wallet.

(5) After authenticating the End-User and getting their consent to share the request Credentials, the Wallet sends the Authorization Response with the parameters `vp_token`, `presentation_submission` (optional) and `state` to the `response_uri` of the Verifier.

(6) The Verifier's Response URI checks whether the `state` value is a valid `request-id`. If so, it stores the Authorization Response data linked to the respective `transaction-id`. It then creates a `response_code` as fresh, cryptographically random number with sufficient entropy that it also links with the respective Authorization Response data. It then returns the `redirect_uri`, which includes the `response_code` to the Wallet.

Note: If the Verifier's Response URI does not return a `redirect_uri`, processing at the Wallet stops at that step. The Verifier is supposed to fetch the Authorization Response without waiting for a redirect (see step 8).

(7) The Wallet sends the user agent to the Verifier (`redirect_uri`). The Verifier receives the Request and extracts the `response_code` parameter.

(8) The Verifier sends the `response_code` and the `transaction-id` from its session to the Response URI.

- The Response URI uses the `transaction-id` to look the matching Authorization Response data up, which implicitly validates the `transaction-id` associated with the Verifier's session.
- If an Authorization Response is found, the Response URI checks whether the `response_code` was associated with this Authorization Response in step (6).

Note: If the Verifier's Response URI did not return a `redirect_uri` in step (6), the Verifier will periodically query the Response URI with the `transaction-id` to obtain the Authorization Response once it becomes available.

(9) The Response URI returns the VP Token and Presentation Submission for further processing to the Verifier.

(10) The Verifier checks whether the nonce received in the Credential(s) in the VP Token in step (9) corresponds to the nonce value from the session. The Verifier then consumes the VP Token and invalidates the `transaction-id`, `request-id` and nonce in the session.

14. Security Considerations

14.1. Preventing Replay of the VP Token

An attacker could try to inject a VP Token (or an individual Verifiable Presentation), that was obtained from a previous Authorization Response, into another Authorization Response thus impersonating the End-User that originally presented that VP Token or the respective Verifiable Presentation.

Implementers of this specification MUST implement the controls as defined in this section to detect such an attack.

This specification assumes that a Verifiable Credential is always presented with a cryptographic proof of possession which can be a Verifiable Presentation. This cryptographic proof of possession MUST be bound by the Wallet to the intended audience (the Client Identifier of the Verifier) and the respective transaction (identified by the nonce parameter in the Authorization Request). The Verifier MUST verify this binding.

The Verifier MUST create a fresh, cryptographically random number with sufficient entropy for every Authorization Request, store it with its current session, and pass it in the nonce Authorization Request Parameter to the Wallet.

The Wallet MUST link every Verifiable Presentation returned to the Verifier in the VP Token to the `client_id` and the nonce values of the respective Authentication Request.

The Verifier MUST validate every individual Verifiable Presentation in an Authorization Response and ensure that it is linked to the values of the `client_id` and the `nonce` parameter it had used for the respective Authorization Request. If the response contains multiple Verifiable Presentations which do not contain the same nonce value, the response is rejected.

The `client_id` is used to detect the presentation of Verifiable Credentials to a party other than the one intended. This allows Verifiers take appropriate action in that case, such as not accepting the Verifiable Presentation. The `nonce` value binds the Presentation to a certain authentication transaction and allows the Verifier to detect injection of a Presentation in the flow, which is especially important in the flows where the Presentation is passed through the front-channel.

Note: Different formats for Verifiable Presentations and signature/proof schemes use different ways to represent the intended audience and the session binding. Some use claims to directly represent those values, others include the values into the calculation of cryptographic proofs. There are also different naming conventions across the different formats. In case [DIF.PresentationExchange] is used in the Authorization Request, the format of the respective presentation is determined from the format information in the presentation submission in the Authorization Response. If DCQL was used, the format was defined by the Verifier in the request.

The following is a non-normative example of the payload of a Verifiable Presentation of a format identifier `jwt_vp_json`:

```
{
  "iss": "did:example:ebfeb1f712ebc6f1c276e12ec21",
  "jti": "urn:uuid:3978344f-8596-4c3a-a978-8fcaba3903c5",
  "aud": "s6BhdRkqt3",
  "nonce": "343s$FSFDa-",
  "nbf": 1541493724,
  "iat": 1541493724,
  "exp": 1573029723,
  "vp": {
    "@context": [
      "https://www.w3.org/2018/credentials/v1",
      "https://www.w3.org/2018/credentials/examples/v1"
    ],
    "type": ["VerifiablePresentation"],

    "verifiableCredential": [""]
  }
}
```

In the example above, the requested `nonce` value is included as the `nonce` and `client_id` as the `aud` value in the proof of the Verifiable Presentation.

The following is a non-normative example of a Verifiable Presentation of a format identifier `ldp_vp` without a `proof` property:

```
{
  "@context": [ ... ],
  "type": "VerifiablePresentation",
  "verifiableCredential": [ ... ],
  "proof": {
    "type": "RsaSignature2018",
    "created": "2018-09-14T21:19:10Z",
    "proofPurpose": "authentication",
    "verificationMethod": "did:example:ebfeb1f712ebc6f1c276e12ec21#keys-1",
    "challenge": "343s$FSFDa-",
    "domain": "s6BhdRkqt3",
    "jws": "eyJhb...nKb78"
  }
}
```

In the example above, the requested nonce value is included as the `challenge` and `client_id` as the `domain` value in the proof of the Verifiable Presentation.

14.2. Session Fixation

To perform a Session Fixation attack, an attacker would start the process using a Verifier executed on a device under his control, capture the Authorization Request and relay it to the device of a victim. The attacker would then periodically try to conclude the process in his Verifier, which would cause the Verifier on his device to try to fetch and verify the Authorization Response.

Such an attack is impossible against flows implemented with the Response Mode `fragment` as the Wallet will always send the VP Token to the redirect endpoint on the same device where it resides. This means an attacker could extract a valid Authorization Request from a Verifier on his device and trick a Victim into performing the same Authorization Request on her device. But there is technically no way for an attacker to get hold of the resulting VP Token.

However, the Response Mode `direct_post` is susceptible to such an attack as the result is sent from the Wallet out-of-band to the Verifier's Response URI.

This kind of attack can be detected if the Response Mode `direct_post` is used in conjunction with the redirect URI, which causes the Wallet to redirect the flow to the Verifier's frontend at the device where the transaction was concluded. The Verifier's Response URI MUST include a fresh secret (Response Code) into the redirect URI returned to the Wallet and the Verifier's Response URI MUST require the frontend to pass the respective Response Code when fetching the Authorization Response. That stops session fixation attacks as long as the attacker is unable to get access to the Response Code.

Note that this protection technique is not applicable to cross-device scenarios because the browser used by the wallet will not have the original session. It is also not applicable in same-device scenarios if the Wallet uses a browser different from the one used on the presentation request (e.g. device with multiple installed browsers), because the original session will also not be available there. Appendix A provides an alternative Wallet invocation method using web/app platform APIs that avoids many of these issues.

See Section 13.4 for more implementation considerations.

When using the Response Mode `direct_post` without the further protection provided by the redirect URI, there is no session context for the Verifier to detect session fixation attempts. It is RECOMMENDED for the Verifiers to implement mechanisms to strengthen the security of the flow. For more details on possible attacks and mitigations see [I-D.ietf-oauth-cross-device-security].

14.3. Response Mode "direct_post"

14.3.1. Validation of the Response URI

The Wallet MUST ensure the data in the Authorization Response cannot leak through Response URIs. When using pre-registered Response URIs, the Wallet MUST comply with best practices for redirect URI validation as defined in [RFC9700]. The Wallet MAY also rely on a Client Identifier Scheme in conjunction with Client Authentication and integrity protection of the request to establish trust in the Response URI provided by a certain Verifier.

14.3.2. Protection of the Response URI

The Verifier SHOULD protect its Response URI from inadvertent requests by checking that the value of the received `state` parameter corresponds to a recent Authorization Request. It MAY also use JARM [JARM] to authenticate the originator of the request.

14.3.3. Protection of the Authorization Response Data

This specification assumes that the Verifier's Response URI offers an internal interface to other components of the Verifier to obtain (and subsequently process) Authorization Response data. An attacker could try to obtain Authorization Response Data from a Verifier's Response URI by looking up this data through the internal interface. This could lead to leakage valid Verifiable Presentations containing PII.

Implementations of this specification MUST have security mechanisms in place to prevent inadvertent requests against this internal interface. Implementation options to fulfill this requirement include:

- Authentication between the different parts within the Verifier
- Two cryptographically random numbers. The first being used to manage state between the Wallet and Verifier. The second being used to ensure that only a legitimate component of the Verifier can obtain the Authorization Response data.

14.4. End-User Authentication using Verifiable Credentials

Clients intending to authenticate the End-User utilizing a claim in a Verifiable Credential MUST ensure this claim is stable for the End-User as well locally unique and never reassigned within the Credential Issuer to another End-User. Such a claim MUST also only be used in combination with the Credential Issuer identifier to ensure global uniqueness and to prevent attacks where an attacker obtains the same claim from a different Credential Issuer and tries to impersonate the legitimate End-User.

14.5. Encrypting an Unsigned Response

If an encrypted Authorization Response has no additional integrity protection, an attacker might be able to alter Authorization Response parameters such as `presentation_submission` and generate a new encrypted Authorization Response for the Verifier, as encryption is performed using the public key of the Verifier which is likely to be widely known. Note this includes injecting a new VP Token. Since the contents of the VP Token are integrity protected, tampering the VP Token is detectable by the Verifier. For details, see Section 14.1.

14.6. DIF Presentation Exchange

14.6.1. Fetching Presentation Definitions by Reference

In many instances the referenced server will be operated by a known federation or other trusted operator, and the URL's domain name will already be widely known. Wallets using this URI can mitigate request forgeries by having a pre-configured set of trusted domain names and only fetching Presentation Definition from these sources. In addition, the Presentation Definitions could be signed by a trusted authority, such as the federation operator.

14.6.2. JSONPath and Arbitrary Scripting

Implementers MUST make sure that JSONPath used as part of `presentation_definition` and `presentation_submission` parameters cannot be used to execute arbitrary scripts on a server. This can be achieved, for example, by implementing the entire syntax of the query without relying on the parsers of programming language engine. For details, see Section 4 of [I-D.ietf-jsonpath-base].

14.6.3. Filters Property

Implementers should be careful with what is used as a filter property in [DIF.PresentationExchange]. For example, when using regular expressions or JSON Schemas as filters, implementers should ensure that computations and resource access are bounded with the security in mind to prevent attacks such as denial of service or unauthorized access.

14.7. TLS Requirements

Implementations MUST follow [BCP195].

Whenever TLS is used, a TLS server certificate check MUST be performed, per [RFC6125].

14.8. Incomplete or Incorrect Implementations of the Specifications and Conformance Testing

To achieve the full security benefits, it is important that the implementation of this specification, and the underlying specifications, are both complete and correct.

The OpenID Foundation provides tools that can be used to confirm that an implementation is correct and conformant:

https://openid.net/certification/conformance-testing-for-openid-for-verifiable-presentations/

15. Privacy Considerations

15.1. Authorization Requests with Request URI

If the Wallet is acting within a trust framework that allows the Wallet to determine whether a 'request_uri' belongs to a certain 'client_id', the Wallet is RECOMMENDED to validate the Verifier's authenticity and authorization given by 'client_id' and that the 'request_uri' corresponds to this Verifier. If the link cannot be established in those cases, the Wallet SHOULD refuse the request or ask the End-User for advise.

If no End-User interaction is required before sending the request, it is easy to request on a large scale and in an automated fashion the wallet capabilities from all visitors of a website. Even without personally identifiable information (PII) this can reveal some information about End-Users, like their nationality (e.g., a Wallet with special capabilities only used in one EU member state).

Mandatory End-User interaction before sending the request, like clicking a button, unlocking the wallet or even just showing a screen of the app, can make this less attractive/likely to being exploited.

Requests from the Wallet to the Verifier SHOULD be sent with the minimal amount of information possible, and in particular, without any HTTP headers identifying the software used for the request (e.g., HTTP libraries or their versions). The Wallet MUST NOT send PII or any other data that could be used for fingerprinting to the Request URI in order to prevent End-User tracking.

15.2. Authorization Error Response with the `wallet_unavailable` error code

In the event that another component is invoked instead of the Wallet, the End-User MUST be informed and give consent before the invoked component returns the `wallet_unavailable` Authorization Error Response to the Verifier.

15.3. Privacy implications of mechanisms to establish trust in Issuers

This specification introduces an extension point that allows for a Verifier to express expected Issuers or trust frameworks that certify Issuers. It is important to understand the implications that different mechanism to establish trust in Issuers can have on the privacy of the overall system.

Generally speaking, a distinction can be made between self-contained mechanisms, where all information necessary to validate if a credential matches the request is already present in the Wallet and Verifier, and those mechanisms that require some form of online resolution. Mechanisms that require online resolution can leak information that could be used to profile the usage of credentials and the overall ecosystem.

Especially the case where a Wallet has to retrieve information before being able to construct a presentation that matches the request could leak information about individual users to other parties. Wallets SHOULD NOT fetch URLs provided by the Verifier in a request that are unknown to the Wallet or hosted by a third party that the Wallet does not trust. The privacy concerns can be mitigated if the URLs are only used by the Wallet as identifiers but not fetched upon receiving the request from the Verifier.

Ecosystems that plan to leverage the trusted authorities mechanisms SHOULD make sure that the privacy properties of the mechanisms they choose to support matches with the desired privacy properties of the overall ecosystem.

16. Normative References

[BCP195] IETF, "BCP195", 2022, <https://www.rfc-editor.org/info/bcp195>.

[DID-Core] Sporny, M., Guy, A., Sabadello, M., and D. Reed, "Decentralized Identifiers (DIDs) v1.0", 3 August 2021, <https://www.w3.org/TR/2021/PR-did-core-20210803/>.

[DIF.PresentationExchange] Buchner, D., Zundel, B., Riedel, M., and K. H. Duffy, "Presentation Exchange 2.1.1", <https://identity.foundation/presentation-exchange/spec/v2.1.1/>.

[JARM] Lodderstedt, T. and B. Campbell, "JWT Secured Authorization Response Mode for OAuth 2.0 (JARM)", 9 November 2022, <https://openid.net/specs/oauth-v2-jarm-final.html>.

[OAuth.Responses] de Medeiros, B., Scurtescu, M., Tarjan, P., and M. Jones, "OAuth 2.0 Multiple Response Type Encoding Practices", 25 February 2014, <https://openid.net/specs/oauth-v2-multiple-response-types-1_0.html>.

[OpenID.Core] Sakimura, N., Bradley, J., Jones, M.B., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 2", 15 December 2023, <https://openid.net/specs/openid-connect-core-1_0.html>.

[OpenID.Federation] Ed., R. H., Jones, M. B., Solberg, A., Bradley, J., Marco, G. D., and V. Dzhuvinov, "OpenID Federation 1.0", 15 September 2024, <https://openid.net/specs/openid-federation-1_0.html>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <https://www.rfc-editor.org/info/rfc6125>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <https://www.rfc-editor.org/info/rfc7516>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/info/rfc7517>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://www.rfc-editor.org/info/rfc7518>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC7591] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015, <https://www.rfc-editor.org/info/rfc7591>.

[RFC7638] Jones, M. and N. Sakimura, "JSON Web Key (JWK) Thumbprint", RFC 7638, DOI 10.17487/RFC7638, September 2015, <https://www.rfc-editor.org/info/rfc7638>.

[RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <https://www.rfc-editor.org/info/rfc7800>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018, <https://www.rfc-editor.org/info/rfc8414>.

[SIOPv2] Yasuda, K., Jones, M. B., and T. Lodderstedt, "Self-Issued OpenID Provider V2", 28 November 2023, <https://openid.net/specs/openid-connect-self-issued-v2-1_0.html>.

[W3C.Digital_Credentials_API] Caceres, M., Cappalli, T., and S. Goto, "Digital Credentials API", <https://wicg.github.io/digital-credentials/>.

17. Informative References

[ETSI.TL] European Telecommunications Standards Institute (ETSI), "ETSI TS 119 612 V2.1.1 Electronic Signatures and Infrastructures (ESI); Trusted Lists", 2015, <https://www.etsi.org/deliver/etsi_ts/119600_119699/119612/02.01.01_60/ts_119612v020101p.pdf>.

[Hyperledger.Indy] Hyperledger Indy Project, "Hyperledger Indy Project", 2022, <https://www.hyperledger.org/use/hyperledger-indy>.

[I-D.ietf-jose-hpke-encrypt] Reddy.K, T., Tschofenig, H., Banerjee, A., Steele, O., and M. B. Jones, "Use of Hybrid Public Key Encryption (HPKE) with JSON Object Signing and Encryption (JOSE)", Work in Progress, Internet-Draft, draft-ietf-jose-hpke-encrypt-07, 18 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-jose-hpke-encrypt-07>.

[I-D.ietf-jsonpath-base] Gössner, S., Normington, G., and C. Bormann, "JSONPath: Query expressions for JSON", Work in Progress, Internet-Draft, draft-ietf-jsonpath-base-21, 24 September 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-jsonpath-base-21>.

[I-D.ietf-oauth-cross-device-security] Kasselman, P., Fett, D., and F. Skokan, "Cross-Device Flows: Security Best Current Practice", Work in Progress, Internet-Draft, draft-ietf-oauth-cross-device-security-09, 6 January 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-cross-device-security-09>.

[I-D.ietf-oauth-sd-jwt-vc] Terbu, O., Fett, D., and B. Campbell, "SD-JWT-based Verifiable Credentials (SD-JWT VC)", Work in Progress, Internet-Draft, draft-ietf-oauth-sd-jwt-vc-08, 3 December 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-sd-jwt-vc-08>.

[IANA.Hash.Algorithms] IANA, "Named Information Hash Algorithm Registry", <https://www.iana.org/assignments/named-information/named-information.xhtml>.

[IANA.JOSE] IANA, "JSON Object Signing and Encryption (JOSE)", <https://www.iana.org/assignments/jose>.

[IANA.OAuth.Parameters] IANA, "OAuth Parameters", <https://www.iana.org/assignments/oauth-parameters>.

[IANA.URI.Schemes] IANA, "Uniform Resource Identifier (URI) Schemes", <https://www.iana.org/assignments/uri-schemes>.

[ISO.18013-5] ISO/IEC JTC 1/SC 17 Cards and security devices for personal identification, "ISO/IEC 18013-5:2021 Personal identification — ISO-compliant driving license — Part 5: Mobile driving license (mDL) application", 2021, <https://www.iso.org/standard/69084.html>.

[ISO.18013-7] ISO/IEC JTC 1/SC 17 Cards and security devices for personal identification, "ISO/IEC DTS 18013-7 Personal identification — ISO-compliant driving license — Part 7: Mobile driving license (mDL) add-on functions", 2024, <https://www.iso.org/standard/82772.html>.

[ISO.23220-2] ISO/IEC JTC 1/SC 17 Cards and security devices for personal identification, "ISO/IEC DTS 23220-2 Personal identification — Building blocks for identity management via mobile devices, Part 2: Data objects and encoding rules for generic eID systems", 2024, <https://www.iso.org/standard/86782.html>.

[ISO.23220-4] ISO/IEC JTC 1/SC 17 Cards and security devices for personal identification, "ISO/IEC CD TS 23220-4 Personal identification — Building blocks for identity management via mobile devices, Part 4: Protocols and services for operational phase", 2024, <https://www.iso.org/standard/86782.html>.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <https://www.rfc-editor.org/info/rfc2046>.

[RFC6819] Lodderstedt, T., Ed., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, DOI 10.17487/RFC6819, January 2013, <https://www.rfc-editor.org/info/rfc6819>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC8252] Denniss, W. and J. Bradley, "OAuth 2.0 for Native Apps", BCP 212, RFC 8252, DOI 10.17487/RFC8252, October 2017, <https://www.rfc-editor.org/info/rfc8252>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[RFC9101] Sakimura, N., Bradley, J., and M. Jones, "The OAuth 2.0 Authorization Framework: JWT-Secured Authorization Request (JAR)", RFC 9101, DOI 10.17487/RFC9101, August 2021, <https://www.rfc-editor.org/info/rfc9101>.

[RFC9126] Lodderstedt, T., Campbell, B., Sakimura, N., Tonge, D., and F. Skokan, "OAuth 2.0 Pushed Authorization Requests", RFC 9126, DOI 10.17487/RFC9126, September 2021, <https://www.rfc-editor.org/info/rfc9126>.

[RFC9207] Meyer zu Selhausen, K. and D. Fett, "OAuth 2.0 Authorization Server Issuer Identification", RFC 9207, DOI 10.17487/RFC9207, March 2022, <https://www.rfc-editor.org/info/rfc9207>.

[RFC9700] Lodderstedt, T., Bradley, J., Labunets, A., and D. Fett, "Best Current Practice for OAuth 2.0 Security", BCP 240, RFC 9700, DOI 10.17487/RFC9700, January 2025, <https://www.rfc-editor.org/info/rfc9700>.

[VC_DATA] Sporny, M., Noble, G., Longley, D., Burnett, D. C., Zundel, B., and D. Chadwick, "Verifiable Credentials Data Model 1.1", 19 November 2019, <https://www.w3.org/TR/2022/REC-vc-data-model-20220303/>.

Appendix A. OpenID4VP over the Digital Credentials API

This section defines how to use OpenID4VP with the Digital Credentials API.

The name "Digital Credentials API" (DC API) encompasses the W3C Digital Credentials API [W3C.Digital_Credentials_API] as well as its native App Platform equivalents in operating systems (such as Credential Manager on Android). The DC API allows web sites and native apps acting as Verifiers to request the presentation of verifiable credentials. The API itself is agnostic to the Credential exchange protocol and can be used with different protocols. The Web Platform, working in conjunction with other layers, such as the app platform/operating system, and based on the permission of the End-User, will send the request data along with the Origin of the Verifier to the End-User's chosen Wallet.

OpenID4VP over the DC API utilizes the mechanisms of the DC API while also allowing to leverage advanced security features of OpenID4VP, if needed. It also defines the OpenID4VP request parameters that MAY be used with the DC API.

The DC API offers several advantages for implementers of both Verifiers and Wallets.

Firstly, the API serves as a privacy-preserving alternative to invoking Wallets via URLs, particularly custom URL schemes. The underlying app platform will only invoke a Wallet if the End-User confirms the request based on contextual information about the Credential Request and the requestor (Verifier).

Secondly, the session with the End-User will always continue in the initial context, typically a web browser tab, when the request has been fulfilled (or aborted), which results in an improved user experience.

Thirdly, cross-device requests benefit from the use of secure transports with proximity checks, which are handled by the OS platform, e.g., using FIDO CTAP 2.2 with hybrid transports.

And lastly, as part of the request, the Wallet is provided with information about the Verifier's Origin as authenticated by the user agent, which is important for phishing resistance.

A.1. Protocol

To use OpenID4VP with the Digital Credentials API (DC API), the exchange protocol value has the following format: `openid4vp-v<version>-<request-type>`. The `<version>` field is a numeric value, and `<request-type>` explicitly specifies the type of request. This approach eliminates the need for Wallets to perform implicit parameter matching to accurately identify the version and the expected request and response parameters.

The value `1` MUST be used for the `<version>` field to indicate the request and response are compatible with this version of the specification. For `<request-type>`, unsigned requests, as defined in Appendix A.3.1, MUST use `unsigned`, and signed requests, as defined in Appendix A.3.2, MUST use `signed`.

The following exchange protocol values are defined by this specification:

- Unsigned requests: `openid4vp-v1-unsigned`
- Signed requests: `openid4vp-v1-signed`

A.2. Request

The Verifier MAY send all the OpenID4VP request parameters to the Digital Credentials API (DC API).

The following is a non-normative example of an unsigned OpenID4VP request (when advanced security features of OpenID4VP are not used) that can be sent over the DC API :

```
{
  response_type: "vp_token",
  response_mode: "dc_api",
  nonce: "n-0S6_WzA2Mj",
  client_metadata: {...},
  dcql_query: {...}
}
```

Out of the Authorization Request parameters defined in [RFC6749] and Section 5, the following are supported with OpenID4VP over the W3C Digital Credentials API:

- `client_id`
- `response_type`
- `response_mode`
- `nonce`

- `presentation_definition`
- `client_metadata`
- `request`
- `transaction_data`
- `dcql_query`

The `client_id` parameter MUST be omitted in unsigned requests defined in Appendix A.3.1. The Wallet MUST ignore any `client_id` parameter that is present in an unsigned request.

Parameters defined by a specific client identifier scheme (such as the `trust_chain` parameter for the OpenID Federation client id scheme) are also supported over the W3C Digital Credentials API.

The `client_id` parameter MUST be present in signed requests defined in Appendix A.3.2, as it communicates to the wallet which Client Identifier Scheme and Client Identifier to use when authenticating the client through verification of the request signature or retrieving client metadata.

The value of the `response_mode` parameter MUST be `dc_api` when the response is neither signed nor encrypted and `dc_api.jwt` when the response is signed and/or encrypted as defined in Section 8.3.

In addition to the above-mentioned parameters, a new parameter is introduced for OpenID4VP over the W3C Digital Credentials API:

- `expected_origins`: REQUIRED when signed requests defined in Appendix A.3.2 are used with the Digital Credentials API (DC API). An array of strings, each string representing an Origin of the Verifier that is making the request. The Wallet can detect replay of the request from a malicious Verifier by comparing values in this parameter to the Origin. This parameter is not for use in unsigned requests and therefore a Wallet MUST ignore this parameter if it is present in an unsigned request.

The transport of the request and Origin to the Wallet is platform-specific and is out of scope of OpenID4VP over the Digital Credentials API.

Additional request parameters MAY be defined and used with OpenID4VP over the DC API.

The Wallet MUST ignore any unrecognized parameters.

A.3. Signed and Unsigned Requests

Any OpenID4VP request compliant to this section of this specification can be used with the Digital Credentials API (DC API). Depending on the mechanism used to identify and authenticate the Verifier, the request can be signed or unsigned. This section defines signed and unsigned OpenID4VP requests for use with the DC API.

A.3.1. Unsigned Request

The Verifier MAY send all the OpenID4VP request parameters as members in the request member passed to the API.

A.3.2. Signed Request

The Verifier MAY send a signed request, for example, when identification and authentication of the Verifier is required.

The signed request allows the Wallet to authenticate the Verifier using one or more trust framework(s) in addition to the Web PKI utilized by the browser. An example of such a trust framework is the Verifier (RP) management infrastructure set up in the context of the eIDAS regulation in the European Union, in which case, the Wallet can

no longer rely only on the web origin of the Verifier. This web origin MAY still be used to further strengthen the security of the flow. The external trust framework could, for example, map the Client Identifier to registered web origins.

The signed Request Object MAY contain all the parameters listed in Appendix A.2, except `request`.

Verifiers SHOULD format signed Requests using JWS Compact Serialization but MAY use JWS JSON Serialization [RFC7515]) to cater for use cases described below.

A.3.2.1. JWS Compact Serialization

When the JWS Compact Serialization is used to send the request, the Verifier can convey only one Trust Framework, i.e., the Verifier should know which trust frameworks the wallet supports. All request parameters are encoded in a request object as defined in Section 5 and the JWS object is used as the value of the `request` claim in the `data` element of the API call.

This is illustrated in the following non-normative example.

```
{ request: "eyJhbGciOiJF..." }
```

This is an example of the payload of a signed OpenID4VP request used with the W3C Digital Credentials API in conjunction with JWS Compact Serialization:

```
{
  "expected_origins": [
    "https://origin1.example.com",
    "https://origin2.example.com"
  ],
  "client_id": "x509_san_dns:rp.example.com",
  "client_metadata": {
    "jwks": {
      "keys": [
        {
          "kty": "EC",
          "crv": "P-256",
          "x": "MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
          "y": "4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM",
          "use": "enc",
          "kid": "1"
        }
      ]
    }
  },
  "response_type": "vp_token",
  "response_mode": "dc_api",
  "nonce": "n-0S6_WzA2Mj",
  "dcql_query": {...}
}
```

A.3.2.2. JWS JSON Serialization

The JWS JSON Serialization [RFC7515]) allows the Verifier to use multiple Client Identifiers and corresponding key material to protect the same request. This serves use cases where the Verifier requests Credentials belonging to different trust frameworks and, therefore, needs to authenticate in the context of those trust frameworks.

In this case, the following request parameters MUST be present in the protected header of the respective `signature` object in the `signatures` array defined in Section 7.2.1 of [RFC7515]:

- `client_id`

All other request parameters MUST be present in the `payload` element of the JWS object.

Below is a non-normative example of such a request:

```
{
  "payload": "eyAiaXNzIjogImh0dHBzOi8...NzY4Mzc4MzYiIF0gfQ",
  "signatures": [
    {
      "protected": "eyJhbGciOiAiRVMyNT..MiLCJraWQiOiAiMSJ9XX19fQ",
      "signature": "PFwem0Ajp2Sag...T2z784h8TQqgTR9tXcif0jw"
    },
    {
      "protected": "eyJhbGciOiAiRVMyNTY...tpZCI6ICIxIn1dfX19",
      "signature": "irgtXbJGwE2wN4Lc...2TvUodsE0vaC-NXpB9G39cMXZ9A"
    }
  ]
}
```

Every object in the `signatures` structure contains the parameters and the signature specific to a particular Client Identifier. The signature is calculated as specified in section 5.1 of [RFC7515].

The following is a non-normative example of a content of a decoded protected header:

```
{
  "alg": "ES256",
  "x5c": [
    "MIICOjCCAeG...djzH7lA==",
    "MIICLTCCAdS...koAmhWVKe"
  ],
  "client_id": "x509_san_dns:rp.example.com"
}
```

The following is a non-normative example of the payload of a signed OpenID4VP request used with the W3C Digital Credentials API in conjunction with JWS JSON Serialization:

```
{
  "expected_origins": [
    "https://origin1.example.com",
    "https://origin2.example.com"
  ],
  "response_type": "vp_token",
  "response_mode": "dc_api",
  "nonce": "n-0S6_WzA2Mj",
  "dcql_query": {...},
  "client_metadata": {
    "jwks": {
      "keys": [
        {
          "kty": "EC",
          "crv": "P-256",
          "x": "MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
          "y": "4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM",
          "use": "enc",
          "kid": "1"
        }
      ]
    }
  }
}
```

A.4. Response

Every OpenID4VP Authorization Request results in a response being provided through the Digital Credentials API (DC API). The response is an instance of the `DigitalCredential` interface, as defined in [W3C.Digital_Credentials_API], and the OpenID4VP Authorization Response parameters as defined for the Response Type are represented as an object within the `data` attribute.

The security properties that are normally provided by the Client Identifier are achieved by binding the response to the Origin it was received from.

The audience for the response (for example, the `aud` value in a Key Binding JWT) MUST be the Origin, prefixed with `origin:`, for example `origin:https://verifier.example.com/`. This is the case even for signed requests. Therefore, when using OpenID4VP over the DC API, the Client Identifier is not used as the audience for the response.

Appendix B. Credential Format Specific Parameters

OpenID for Verifiable Presentations is Credential Format agnostic, i.e., it is designed to allow applications to request and receive Verifiable Presentations and Verifiable Credentials in any Credential Format. This section defines a set of Credential Format specific parameters for some of the known Credential Formats. For the Credential Formats that are not mentioned in this specification, other specifications or deployments can define their own set of Credential Format specific parameters.

B.1. W3C Verifiable Credentials

W3C Verifiable Credentials may use an additional parameter for the `descriptor_map` with the `presentation_submission`: The `path_nested` object inside an Input Descriptor Mapping Object is used to describe how to find a returned Credential within a Verifiable Presentation, and contains a `format` parameter with the Credential format identifier as a value and a `path` parameter with a relative path to the Verifiable Credential. Non-normative examples can be found further in this section.

B.1.1. VC signed as a JWT, not using JSON-LD

This section illustrates presentation of a Credential conformant to [VC_DATA] that is signed using JWS, and does not use JSON-LD.

The Credential format identifiers are `jwt_vc_json` for a W3C Verifiable Credential and `jwt_vp_json` for W3C Verifiable Presentation.

Cipher suites should use algorithm names defined in IANA JOSE Algorithms Registry.

B.1.1.1. Example Credential

The following is a non-normative example of the payload of a JWT-based W3C Verifiable Credential that will be used throughout this section:

```
{
  "iss": "https://example.gov/issuers/565049",
  "nbf": 1262304000,
  "jti": "http://example.gov/credentials/3732",
  "sub": "did:example:ebfeb1f712ebc6f1c276e12ec21",
  "vc": {
    "@context": [
      "https://www.w3.org/2018/credentials/v1",
      "https://www.w3.org/2018/credentials/examples/v1"
    ],
    "type": [
      "VerifiableCredential",
      "IDCredential"
    ],
    "credentialSubject": {
      "given_name": "Max",
      "family_name": "Mustermann",
      "birthdate": "1998-01-11",
      "address": {
        "street_address": "Sandanger 25",
        "locality": "Musterstadt",
        "postal_code": "123456",
        "country": "DE"
      }
    }
  }
}
```

B.1.1.2. Presentation Request

The following is a non-normative example of an Authorization Request:

```
GET /authorize?
  response_type=vp_token
  &client_id=x509_san_dns%3Aclient.example.org
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
  &dcql_query=%7B...%7D
  &nonce=n-0S6_WzA2Mj HTTP/1.1
Host: wallet.example.com
```

The requirements regarding the Credential to be presented are conveyed in the `presentation_definition` parameter.

The following is a non-normative example of the contents of a `presentation_definition` parameter:

```
{
  "id": "example_jwt_vc",
  "input_descriptors": [
    {
      "id": "id_credential",
      "format": {
        "jwt_vc_json": {
          "proof_type": [
            "JsonWebSignature2020"
          ]
        }
      },
      "constraints": {
        "fields": [
          {
            "path": [
              "$.vc.type"
            ],
            "filter": {
              "type": "array",
              "contains": {
                "const": "IDCredential"
              }
            }
          }
        ]
      }
    }
  ]
}
```

This `presentation_definition` parameter contains a single `input_descriptor` element, which sets the desired format to JWT VC and defines a constraint over the `vc.type` parameter to select Verifiable Credentials of type `IDCredential`.

B.1.1.3. Presentation Response

The following requirements apply to the `nonce` and `aud` claims of the Verifiable Presentation:

- the `nonce` claim MUST be the value of `nonce` from the Authorization Request;
- the `aud` claim MUST be the value of the Client Identifier, except for requests over the DC API where it MUST be the Origin prefixed with `origin:`, as described in Appendix A.4.

The following is a non-normative example of an Authorization Response:

```
HTTP/1.1 302 Found
  Location: https://client.example.org/cb#
    presentation_submission=...
    &vp_token=...
```

The following is a non-normative example of the content of the `presentation_submission` parameter:

```
{
  "definition_id": "example_jwt_vc",
  "id": "example_jwt_vc_presentation_submission",
  "descriptor_map": [
    {
      "id": "id_credential",
      "path": "$",
      "format": "jwt_vp_json",
      "path_nested": {
        "path": "$.vp.verifiableCredential[0]",
        "format": "jwt_vc_json"
      }
    }
  ]
}
```

The following is a non-normative example of the payload of the Verifiable Presentation in the `vp_token` parameter provided in the same response and referred to by the `presentation_submission` above:

```
{
  "iss": "did:example:ebfeb1f712ebc6f1c276e12ec21",
  "jti": "urn:uuid:3978344f-8596-4c3a-a978-8fcaba3903c5",
  "aud": "x509_san_dns:client.example.org",
  "nbf": 1541493724,
  "iat": 1541493724,
  "exp": 1573029723,
  "nonce": "n-0S6_WzA2Mj",
  "vp": {
    "@context": [
      "https://www.w3.org/2018/credentials/v1"
    ],
    "type": [
      "VerifiablePresentation"
    ],
    "verifiableCredential": [
      "eyJhb...ssw5c"
    ]
  }
}
```

B.1.2. LDP VCs

This section illustrates presentation of a Credential conformant to [VC_DATA] that is secured using Data Integrity, using JSON-LD.

The Credential format identifiers are `ldp_vc` for a W3C Verifiable Credential and `ldp_vp` for W3C Verifiable Presentation.

Cipher suites should use signature suites names defined in Linked Data Cryptographic Suite Registry.

B.1.2.1. Example Credential

The following is a non-normative example of the payload of a Verifiable Credential that will be used throughout this section:

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://www.w3.org/2018/credentials/examples/v1"
  ],
  "id": "https://example.com/credentials/1872",
  "type": [
    "VerifiableCredential",
    "IDCredential"
  ],
  "issuer": {
    "id": "did:example:issuer"
  },
  "issuanceDate": "2010-01-01T19:23:24Z",
  "credentialSubject": {
    "given_name": "Max",
    "family_name": "Mustermann",
    "birthdate": "1998-01-11",
    "address": {
      "street_address": "Sandanger 25",
      "locality": "Musterstadt",
      "postal_code": "123456",
      "country": "DE"
    }
  },
  "proof": {
    "type": "Ed25519Signature2018",
    "created": "2021-03-19T15:30:15Z",
    "jws": "eyJhb...JQdBw",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "did:example:issuer#keys-1"
  }
}
```

B.1.2.2. Presentation Request

The following is a non-normative example of an Authorization Request:

```
GET /authorize?
  response_type=vp_token
  &client_id=x509_san_dns%3Aclient.example.org
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
  &dcql_query=%7B...%7D
  &nonce=n-0S6_WzA2Mj HTTP/1.1
Host: wallet.example.com
```

The following is a non-normative example of the contents of a `presentation_definition` parameter that contains the requirements regarding the Credential to be presented:

```
{
  "id": "example_ldp_vc",
  "input_descriptors": [
    {
      "id": "id_credential",
      "format": {
        "ldp_vc": {
          "proof_type": [
            "Ed25519Signature2018"
          ]
        }
      },
      "constraints": {
        "fields": [
          {
            "path": [
              "$.type"
            ],
            "filter": {
              "type": "array",
              "contains": {
                "const": "IDCredential"
              }
            }
          }
        ]
      }
    }
  ]
}
```

This `presentation_definition` parameter contains a single `input_descriptor` element, which sets the desired format to LDP VC and defines a constraint over the `type` parameter to select Verifiable Credentials of type `IDCredential`.

B.1.2.3. Presentation Response

The following requirements apply to the `challenge` and `domain` claims within the `proof` object in the Verifiable Presentation:

- the `challenge` claim MUST be the value of `nonce` from the Authorization Request;
- the `domain` claim MUST be the value of the Client Identifier, except for requests over the DC API where it MUST be the Origin prefixed with `origin:`, as described in Appendix A.4.

The following is a non-normative example of an Authorization Response:

```
HTTP/1.1 302 Found
  Location: https://client.example.org/cb#
    presentation_submission=...
    &vp_token=...
```

The following is a non-normative example of the content of the `presentation_submission` parameter:

```
{
  "definition_id": "example_ldp_vc",
  "id": "example_ldp_vc_presentation_submission",
  "descriptor_map": [
    {
      "id": "id_credential",
      "path": "$",
      "format": "ldp_vp",
      "path_nested": {
        "format": "ldp_vc",
        "path": "$.verifiableCredential[0]"
      }
    }
  ]
}
```

The following is a non-normative example of the Verifiable Presentation in the `vp_token` parameter provided in the same response and referred to by the `presentation_submission` above:

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1"
  ],
  "type": [
    "VerifiablePresentation"
  ],
  "verifiableCredential": [
    {
      "@context": [
        "https://www.w3.org/2018/credentials/v1",
        "https://www.w3.org/2018/credentials/examples/v1"
      ],
      "id": "https://example.com/credentials/1872",
      "type": [
        "VerifiableCredential",
        "IDCredential"
      ],
      "issuer": {
        "id": "did:example:issuer"
      },
      "issuanceDate": "2010-01-01T19:23:24Z",
      "credentialSubject": {
        "given_name": "Max",
        "family_name": "Mustermann",
        "birthdate": "1998-01-11",
        "address": {
          "street_address": "Sandanger 25",
          "locality": "Musterstadt",
          "postal_code": "123456",
          "country": "DE"
        }
      },
      "proof": {
        "type": "Ed25519Signature2018",
        "created": "2021-03-19T15:30:15Z",
        "jws": "eyJhb...JQdBw",
        "proofPurpose": "assertionMethod",
        "verificationMethod": "did:example:issuer#keys-1"
      }
    }
  ],
```

```
  "id": "ebc6f1c2",
  "holder": "did:example:holder",
  "proof": {
    "type": "Ed25519Signature2018",
    "created": "2021-03-19T15:30:15Z",
    "challenge": "n-0S6_WzA2Mj",
    "domain": "https://client.example.org/cb",
    "jws": "eyJhb...IAoDA",
    "proofPurpose": "authentication",
    "verificationMethod": "did:example:holder#key-1"
  }
}
```

B.2. AnonCreds

AnonCreds is a Credential format defined as part of the Hyperledger Indy project [Hyperledger.Indy].

To be able to request AnonCreds, there needs to be a set of identifiers for Verifiable Credentials, Verifiable Presentations ("proofs" in Indy terminology) and crypto schemes.

Credential format identifier is `ac_vc` for a Credential, and `ac_vp` for a Presentation.

Identifier for a CL-signature crypto scheme used in the examples in this section is `CLSignature2019`.

B.2.1. Example Credential

The following is a non-normative example of an AnonCred Credential that will be used throughout this section.

```
{
  "schema_id": "3QowxFtwciWceMFr7WbwnM:2:BasicScheme:0.1",
  "cred_def_id": "CsiDLAiFkQb9N4NDJKUagd:3:CL:4687:awesome_cred",
  "rev_reg_id": null,
  "values": {
    "given_name": {
      "raw": "Alice",
      "encoded": "6874ecdbdb214ee888e37c8c983e2f1c9c0ed16907b519704db42bb6"
    },
    "family_name": {
      "raw": "Wonderland",
      "encoded": "f5e16db78511f23bf2bcf0f450f20180951557cd75efe88b276988fd"
    },
    "email": {
      "raw": "alice@example.com",
      "encoded": "0fbaa7f92a47fe3c5201e97f063983c702432e90dd7bf0c723386543"
    }
  },
  "signature": {
    "p_credential": {
      "m_2":
"992195240129977994432208002187600234475371076406214191371856292432784039213 12",
      "a":
"548556525746779881166502363060885163615377345704149093670326722191034441972054896748465450820120127112612497543713104953674756147292096538507200349133984821847572549205370512979369101250236133232553175158239742314935729039916406597411086037153784904088365076431910519861377932688563163336009329150783379200016922350292789311841736926943662236631319436578343493398286189784364029730469999615394443801165813143729065984150145285622073347457740980970005675152122228947713570445005445523723143358948830006141449948567021811410909050334282214036546363249183438081367500409084432124923594857824716362940130622951539970682 52",
      "e":
"259344723055062059907025491480697571938277889515152306249728583105665800713306759149981690559193987143012367913206299323899696942213235956742930239825562861075148170278284639129199",
      "v":
"977423225617965826161030874586673609060253833336339637510512042715627326115520777573240007342290504514760916978895280468392292138385927475847984210013865986559197693721526403273427741674411349176661607661236811589163783458814384047777877615932551403490096873032745927956461585806847228270552979880833410883312450559437179134831763953399331039151162057919911235795917007675379271170053331252291079735284232344593300423804859916403968643214416588459905206153801412671086607579121000658589346508562139550318271086619781712940854619380589332116137235518796299059578133953385153307733479053043801681733360367591070214663597528225374781981078812975105572836893748312136399274883147513923318085314590610847675371323964494354191654012345637136697487470259820179692926115192564354313217049593303511201208208089304991597720916759 7"
    },
    "r_credential": null
  },
  "signature_correctness_proof": {
    "se":
"898650024692810554511924969312048260691399637687533797581722809056977788610012057585144439213217548517680094627672987529874766409998941262324905602278434880865857749175864455659490120359881993653243522595921161754584103681679989216511801516951222991055767048310149902818885131898400173226695593980184304985256958606680344269024838697022632403956195405056760701064662413239237428064066385409205010620382146840365833878840802301415108893130877666939818418022886944971726762448423579646972188928409413153354969210611360234293228835035659134354622782864249464787263344233036121114964943246814333951837182449655506730293 5",
    "c": "935829931409817995984067028413342821000008660012747101652998044986797842155 98"
  },
  "rev_reg": null,
  "witness": null
}
```

The most important parts for the purpose of this section are `scheme_id` parameter and `values` parameter that contains the actual End-User claims.

B.2.2. Presentation Request

B.2.2.1. Request Example

The following is a non-normative example of an Authorization Request:

```
GET /authorize?
  response_type=vp_token
  &client_id=x509_san_dns%3Aclient.example.org
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
  &dcql_query=%7B...%7D
  &nonce=n-0S6_WzA2Mj HTTP/1.1
Host: wallet.example.com
```

The following is a non-normative example of the content of the `presentation_definition` parameter:

```
{
  "id": "example_vc_ac",
  "input_descriptors": [
    {
      "id": "id_credential",
      "format": {
        "ac_vc": {
          "proof_type": [
            "CLSignature2019"
          ]
        }
      },
      "constraints": {
        "fields": [
          {
            "path": [
              "$.schema_id"
            ],
            "filter": {
              "type": "string",
              "const": "did:indy:idu:test:3QowxFtwciWceMFr7WbwnM:2:BasicScheme:0\\.1"
            }
          }
        ]
      }
    }
  ]
}
```

The `format` object in the `input_descrioptor` element uses the format identifier `ac_vc` as defined above and sets the `proof_type` to `CLSignature2019` to denote this descriptor requires a Credential in AnonCreds format signed with a CL signature (Camenisch-Lysyanskaya signature). The rest of the expressions operate on the AnonCreds JSON structure.

The `constraints` object requires the selected Credential to conform with the schema definition `did:indy:idu:test:3QowxFtwciWceMFr7WbwnM:2:BasicScheme:0\\.1`, which is denoted as a constraint over the AnonCred's `schema_id` parameter.

B.2.2.2. Request Example with Selective Release of Claims

The next example leverages the AnonCreds' capabilities for selective release by requesting a subset of the claims in the Credential to be disclosed to the Verifier.

A non-normative example of an Authorization Request would look the same as in Appendix B.2.2.1.

The following is a non-normative example of the difference is in the `presentation_definition` parameter:

```
{
  "id": "example_vc_ac_sd",
  "input_descriptors": [
    {
      "id": "id_credential",
      "format": {
        "ac_vc": {
          "proof_type": [
            "CLSignature2019"
          ]
        }
      },
      "constraints": {
        "limit_disclosure": "required",
        "fields": [
          {
            "path": [
              "$.schema_id"
            ],
            "filter": {
              "type": "string",
              "const": "did:indy:idu:test:3QowxFtwciWceMFr7WbwnM:2:BasicScheme:0\\.1"
            }
          },
          {
            "path": [
              "$.values.given_name"
            ]
          },
          {
            "path": [
              "$.values.family_name"
            ]
          }
        ]
      }
    }
  ]
}
```

This example is identical to the previous one with the following exceptions: It sets the element `limit_disclosure` of the constraint to `require` and adds two more constraints for the individual claims `given_name` and `family_name`. Since such claims are stored underneath a `values` container in an AnonCred, `values` is part of the path to identify the respective claims.

B.2.3. Presentation Response

A non-normative example of the Authorization Response would look the same as in the examples of other Credential formats. It would contain the `presentation_submission` and `vp_token` parameters.

The following is a non-normative example of the content of the `presentation_submission` parameter:

```
{
  "definition_id": "example_vc_ac_sd",
  "id": "example_vc_ac_sd_presentation_submission",
  "descriptor_map": [
    {
      "id": "id_credential",
      "path": "$",
      "format": "ac_vp",
      "path_nested": {
        "path": "$.requested_proof.revealed_attr_groups.id_card_credential",
        "format": "ac_vc"
      }
    }
  ]
}
```

The `descriptor_map` refers to the `input_descriptor` element with an identifier `id_credential` and tells the Verifier that there is a proof of AnonCred Credential (`format` is `ac_vp`) directly in the vp_token (path is the root designated by $). Furthermore, it indicates using `path_nested` parameter that the End-User claims can be found embedded in the proof underneath `requested_proof.revealed_attr_groups.id_card_credential`.

The following is the content of the `vp_token` parameter:

```
{
  "proof": {...},
  "requested_proof": {
    "revealed_attrs": {},
    "revealed_attr_groups": {
      "id_card_credential": {
        "sub_proof_index": 0,
        "values": {
          "family_name": {
            "raw": "Wonderland",
            "encoded": "167908493…94017654562035"
          },
          "given_name": {
            "raw": "Alice",
            "encoded": "270346400…99344178781507"
          }
        }
      }
    },
    …
  },
  "identifiers": [
    {
      "schema_id": "3QowxFtwciWceMFr7WbwnM:2:BasicScheme:0.1",
      "cred_def_id": "CsiDLAiFkQb9N4NDJKUagd:3:CL:4687:awesome_cred",
      "rev_reg_id": null,
      "timestamp": null
    }
  ]
}
```

B.3. Mobile Documents or mdocs (ISO/IEC 18013 and ISO/IEC 23220 series)

ISO/IEC 18013-5:2021 [ISO.18013-5] defines a mobile driving license (mDL) Credential in the mobile document (mdoc) format. Although ISO/IEC 18013-5:2021 [ISO.18013-5] is specific to mobile driving licenses (mDLs), the Credential format can be utilized with any type of Credential (or mdoc document types). The ISO/IEC 23220 series

has extracted components from ISO/IEC 18013-5:2021 [ISO.18013-5] and ISO/IEC TS 18013-7 [ISO.18013-7] that are common across document types to facilitate the profiling of the specification for other document types. The core data structures are shared between ISO/IEC 18013-5:2021 [ISO.18013-5], ISO/IEC 23220-2 [ISO.23220-2], ISO/IEC 23220-4 [ISO.23220-4] which are encoded in CBOR and secured using COSE_Sign1.

The Credential format identifier for Credentials in the mdoc format is `mso_mdoc`.

ISO/IEC TS 18013-7 Annex B [ISO.18013-7] and ISO/IEC 23220-4 [ISO.23220-4] Annex C define a profile of OpenID4VP for requesting and presenting Credentials in the mdoc format.

[ISO.18013-7] defines the following elements:

- Rules for the `presentation_definition` Authorization Request parameter.
- Rules for the `presentation_submission` Authorization Response parameter.
- Wallet invocation using the `mdoc-openid4vp://` custom URI scheme.
- Required Wallet and Verifier Metadata parameters and their values when OpenID4VP is used with the `mdoc-openid4vp://` custom URI scheme. The `SessionTranscript` and `Handover` CBOR structure when the invocation does not use the DC API. Also see Appendix B.3.5.2.
- Additional restrictions on Authorization Request and Authorization Response parameters to ensure compliance with ISO/IEC TS 18013-7 [ISO.18013-7] and ISO/IEC 23220-4 [ISO.23220-4]. For instance, to comply with ISO/IEC TS 18013-7 [ISO.18013-7], only the same-device flow is supported, the `request_uri` Authorization Request parameter is required, and the Authorization Response has to be encrypted.

B.3.1. Transaction Data

It is RECOMMENDED that each transaction data type defines a data element (`NameSpace`, `DataElementIdentifier`, `DataElementValue`) to be used to return the processed transaction data. Additionally it is RECOMMENDED that it specifies the processing rules, potentially including any hash function to be applied, and the expected resulting structure.

Some document types support some transaction data (Section 8.4) to be protected using mdoc authentication, as part of the `DeviceSigned` data structure [ISO.18013-5]. In those cases, the specifications of these document types include which transaction data types are supported, and the issuer includes the relevant data elements in the `KeyAuthorizations`. If a Wallet receives a request with a `transaction_data` type whose data element is unauthorized, the Wallet MUST reject the request due to an unsupported transaction data type.

B.3.2. DCQL Query and Response

This section defines ISO mdoc specific DCQL Query and Response parameters.

B.3.2.1. Parameters in the `meta` parameter in Credential Query

The following is an ISO mdoc specific parameter in the `meta` parameter in a Credential Query as defined in Section 6.1.

`doctype_value`: OPTIONAL. String that specifies an allowed value for the doctype of the requested Verifiable Credential. It MUST be a valid doctype identifier as defined in [ISO.18013-5].

B.3.2.2. Parameters in the Claims Query

The following are ISO mdoc specific parameters to be used in a Claims Query as defined in Section 6.3.

`intent_to_retain` OPTIONAL. A boolean that is equivalent to `IntentToRetain` variable defined in Section 8.3.2.1.2.1 of [ISO.18013-5].

B.3.2.3. mdoc DCQL Query example

An example DCQL query using the mdoc format is shown in Appendix C. The following is a non-normative example for a VP Token in the response:

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

B.3.3. Presentation Request

See ISO/IEC TS 18013-7 Annex B [ISO.18013-7] and ISO/IEC 23220-4 Annex C [ISO.23220-4] for the latest examples on how to use the `presentation_definition` parameter for requesting Credentials in the mdoc format.

B.3.4. Presentation Response

The VP Token contains the base64url-encoded `DeviceResponse` CBOR structure as defined in ISO/IEC 18013-5 [ISO.18013-5] or ISO/IEC 23220-4 [ISO.23220-4]. Essentially, the `DeviceResponse` CBOR structure contains a signature or MAC over the `SessionTranscript` CBOR structure including the OpenID4VP-specific `Handover` CBOR structure.

See ISO/IEC TS 18013-7 Annex B [ISO.18013-7] and ISO/IEC 23220-4 Annex C [ISO.23220-4] for the latest examples on how to use the `presentation_submission` parameter and how to generate the Authorizaton Response for presenting Credentials in the mdoc format. This includes how the `client_id` and `nonce` are used in the `SessionTranscript`.

B.3.5. `Handover` and `SessionTranscript` Definitions

B.3.5.1. Invocation via the Digital Credentials API

If the presentation request is invoked using the Digital Credentials API, the `SessionTranscript` CBOR structure as defined in Section 9.1.5.1 in [ISO.18013-5] MUST be used with the following changes:

- `DeviceEngagementBytes` MUST be `null`.
- `EReaderKeyBytes` MUST be `null`.
- `Handover` MUST be the `OpenID4VPDCAPIHandover` CBOR structure as defined below.

Note: The following section contains a definition in Concise Data Definition Language (CDDL), a language used to define data structures - see [RFC8610] for more details. `bstr` refers to Byte String, defined as major type 2 in CBOR and `tstr` refers to Text String, defined as major type 3 in CBOR (encoded in utf-8) as defined in section 3.1 of [RFC8949].

```
OpenID4VPDCAPIHandover = [
  "OpenID4VPDCAPIHandover", ; A fixed identifier for this handover type
  OpenID4VPDCAPIHandoverInfoHash ; A cryptographic hash of OpenID4VPDCAPIHandoverInfo
]

; Contains the sha-256 hash of OpenID4VPDCAPIHandoverInfoBytes
OpenID4VPDCAPIHandoverInfoHash = bstr

; Contains the bytes of OpenID4VPDCAPIHandoverInfo encoded as CBOR
OpenID4VPDCAPIHandoverInfoBytes = bstr .cbor OpenID4VPDCAPIHandoverInfo

OpenID4VPDCAPIHandoverInfo = [
  origin,
  nonce,
  jwk_thumbprint
] ; Array containing handover parameters

origin = tstr

nonce = tstr

jwk_thumbprint = bstr
```

The `OpenID4VPDCAPIHandover` structure has the following elements:

- The first element MUST be the string `OpenID4VPDCAPIHandover`. This serves as a unique identifier for the handover structure to prevent misinterpretation or confusion.
- The second element MUST be a Byte String which contains the sha-256 hash of the bytes of `OpenID4VPDCAPIHandoverInfo` when encoded as CBOR.
- The `OpenID4VPDCAPIHandoverInfo` has the following elements:
  - The first element MUST be the string representing the Origin of the request as described in Appendix A.2. It MUST NOT be prefixed with `origin:`.
  - The second element MUST be the value of the `nonce` request parameter.
  - For the Response Mode `dc_api.jwt`, the third element MUST be the JWK SHA-256 Thumbprint as defined in [RFC7638], encoded as a CBOR Byte String, of the Verifier's public key used to encrypt the response. If the Response Mode is `dc_api`, the third element MUST be `null`. For unsigned requests, including the JWK Thumbprint in the `SessionTranscript` allows the Verifier to detect whether the response was re-encrypted by a third party, potentially leading to the leakage of sensitive information. While this does not prevent such an attack, it makes it detectable and helps preserve the confidentiality of the response.

The following is a non-normative example of the input JWK for calculating the JWK Thumbprint in the context of `OpenID4VPDCAPIHandoverInfo`:

```
{
  "kty": "EC",
  "crv": "P-256",
  "x": "DxiH5Q4Yx3UrukE2lWCErq8N8bqC9CHLLrAwLz5BmE0",
  "y": "XtLM4-3h5o3HUH0MHVJV0kyq0iBlrBwlh8qEDMZ4-Pc",
  "use": "enc",
  "alg": "ECDH-ES",
  "kid": "1"
}
```

The following is a non-normative example of the `OpenID4VPDCAPIHandoverInfo` structure:

```
Hex:

837368747470733a2f2f6578616d706c652e636f6d782b6578633767426b786a7831
726463397564527276654b7653734a4971383061766c58654c486847777174415820
4283ec927ae0f208daaa2d026a814f2b22dca52cf85ffa8f3f8626c6bd669047

CBOR diagnostic:

83                                  # array(3)
  73                                #   string(19)
    68747470733a2f2f6578616d706c65  #     "https://example"
    2e636f6d                        #     ".com"
  78 2b                             #   string(43)
    6578633767426b786a783172646339  #     "exc7gBkxjx1rdc9"
    7564527276654b7653734a49713830  #     "udRrveKvSsJIq80"
    61766c58654c48684777717441      #     "avlXeLHhGwqtA"
  58 20                             #   bytes(32)
    4283ec927ae0f208daaa2d026a814f  #     "B\x83i\x92zàò\x08Úª-\x02j\x81O"
    2b22dca52cf85ffa8f3f8626c6bd66  #     "+"Ü¥,ø_ú\x8f?\x86&Æ½f"
    9047                            #     "\x90G"
```

The following is a non-normative example of the `OpenID4VPDCAPIHandover` structure:

```
Hex:

82764f70656e4944345650444341504948616e646f7665725820fbece366f4212f97
62c74cfdbf83b8c69e371d5d68cea09cb4c48ca6daab761a

CBOR diagnostic:

82                                  # array(2)
  76                                #   string(22)
    4f70656e49443456504443415049 48 #     "OpenID4VPDCAPIH"
    616e646f766572                  #     "andover"
  58 20                             #   bytes(32)
    fbece366f4212f9762c74cfdbf83b8  #     "ûìãfô!/\x97bÇLý¿\x83¸"
    c69e371d5d68cea09cb4c48ca6daab  #     "Æ\x9e7\x1d]hÎ\xa0\x9c´Ä\x8c¦Ú«"
    761a                            #     "v\x1a"
```

The following is a non-normative example of the `SessionTranscript` structure:

```
Hex:

83f6f682764f70656e4944345650444341504948616e646f7665725820fbece366f4
212f9762c74cfdbf83b8c69e371d5d68cea09cb4c48ca6daab761a

CBOR diagnostic:

83                                  # array(3)
  f6                                #   null
  f6                                #   null
  82                                #   array(2)
    76                              #     string(22)
      4f70656e49443456504443415049  #       "OpenID4VPDCAPI"
      48616e646f766572              #       "Handover"
    58 20                           #     bytes(32)
      fbece366f4212f9762c74cfdbf83  #       "ûìãfô!/\x97bÇLý¿\x83"
      b8c69e371d5d68cea09cb4c48ca6  #       "¸Æ\x9e7\x1d]hÎ\xa0\x9c´Ä\x8c¦"
      daab761a                      #       "Ú«v\x1a"
```

B.3.5.2. Invocation via other methods

If the presentation request is invoked via other methods, the rules for generating the `SessionTranscript` and `Handover` CBOR structure are specified in ISO/IEC 18013-7 [ISO.18013-7], ISO/IEC 18013-5 [ISO.18013-5] and ISO/IEC 23220-4 [ISO.23220-4].

B.4. IETF SD-JWT VC

This section defines how Credentials complying with [I-D.ietf-oauth-sd-jwt-vc] can be presented to the Verifier using this specification.

B.4.1. Format Identifier

The Credential format identifier is `dc+sd-jwt`.

B.4.1.1. Example Credential

The following is a non-normative example of the unsecured payload of an IETF SD-JWT VC that will be used throughout this section:

```
{
  "vct": "https://credentials.example.com/identity_credential",
  "given_name": "John",
  "family_name": "Doe",
  "birthdate": "1940-01-01"
}
```

The following is a non-normative example of an IETF SD-JWT VC using the unsecured payload above, containing claims that are selectively disclosable.

```
{
  "_sd": [
    "3oUCnaKt7wqDKuyh-LgQozzfhgb8gO5Ni-RCWsWW2vA",
    "8z8z9X9jUtb99gjejCwFAGz4aqlHf-sCqQ6eM_qmpUQ",
    "Cxq4872UXXngGULT_kl8fdwVFkyK6AJfPZLy7L5_0kI",
    "TGf4oLbgwd5JQaHyKVQZU9UdGE0w5rtDsrZzfUaomLo",
    "jsu9yVulwQQlhFlM_3JlzMaSFzglhQG0DpfayQwLUK4",
    "sFcViHN-JG3eTUyBmU4fkwusy5I1SLBhe1jNvKxP5xM",
    "tiTngp9_jhC389UP8_k67MXqoSfiHq3iK6o9un4we_Y",
    "xsKkGJXD1-e3I9zj0YyKNv-lU5YqhsEAF9NhOr8xga4"
  ],
  "iss": "https://example.com/issuer",
  "iat": 1683000000,
  "exp": 1883000000,
  "vct": "https://credentials.example.com/identity_credential",
  "_sd_alg": "sha-256",
  "cnf": {
    "jwk": {
      "kty": "EC",
      "crv": "P-256",
      "x": "TCAER19Zvu3OHF4j4W4vfSVoHIP1ILilDls7vCeGemc",
      "y": "ZxjiWWbZMQGHVWKVQ4hbSIirsVfuecCE6t4jT9F2HZQ"
    }
  }
}
```

The following are disclosures belonging to the claims from the example above.

Claim `given_name`:

- SHA-256 Hash: `jsu9yVulwQQlhFlM_3JlzMaSFzglhQG0DpfayQwLUK4`
- Disclosure:
  `WyIyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgImdpdmVuX25hbWUiLCAiSm9o
  biJd`
- Contents: `["2GLC42sKQveCfGfryNRN9w", "given_name", "John"]`

Claim `family_name`:

- SHA-256 Hash: `TGf4oLbgwd5JQaHyKVQZU9UdGE0w5rtDsrZzfUaomLo`
- Disclosure:
  `WyJlbHVWNU9nM2dTTklJOEVZbnN4QV9BIiwgImZhbWlseV9uYW1lIiwgIkRv
  ZSJd`
- Contents: `["eluV5Og3gSNII8EYnsxA_A", "family_name", "Doe"]`

Claim `birthdate`:

- SHA-256 Hash: `tiTngp9_jhC389UP8_k67MXqoSfiHq3iK6o9un4we_Y`
- Disclosure:
  `WyI2SWo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgImJpcnRoZGF0ZSIsICIxOTQw
  LTAxLTAxIl0`
- Contents: `["6Ij7tM-a5iVPGboS5tmvVA", "birthdate", "1940-01-01"]`

B.4.2. Transaction Data

It is RECOMMENDED that each transaction data type defines a top level claim parameter to be used in the Key Binding JWT to return the processed transaction data. Additionally it is RECOMMENDED that it specifies the processing rules, potentially including any hash function to be applied, and the expected resulting structure.

Note: When following this recommendation, the transaction data mechanism requires use of an SD-JWT VC with Cryptographic Holder Binding.

B.4.2.1. A Profile of Transaction Data in SD-JWT VC

The following is one profile that can be included in a transaction data type specification:

- The `transaction_data` request parameter includes the following parameter, in addition to `type` and `credential_ids` from Section 5.1:
  - `transaction_data_hashes_alg`: OPTIONAL. Array of strings each representing a hash algorithm identifier, one of which MUST be used to calculate hashes in `transaction_data_hashes` response parameter. The value of the identifier MUST be a hash algorithm value from the "Hash Name String" column in the IANA "Named Information Hash Algorithm" registry [IANA.Hash.Algorithms] or a value defined in another specification and/or profile of this specification. If this parameter is not present, a default value of `sha-256` MUST be used. To promote interoperability, implementations MUST support the sha-256 hash algorithm.

- The Key Binding JWT in the response includes the following top level parameters:
  - `transaction_data_hashes`: Array of hashes, where each hash is calculated using a hash function over the data in the strings received in the `transaction_data` request parameter. Each hash value ensures the integrity of, and maps to, the respective transaction data object. If `transaction_data_hashes_alg` was specified in the request, the hash function MUST be one of its values. If `transaction_data_hashes_alg` was not specified in the request, the hash function MUST be `sha-256`.
  - `transaction_data_hashes_alg`: REQUIRED when this parameter was present in the `transaction_data` request parameter. String representing the hash algorithm identifier used to calculate hashes in `transaction_data_hashes` response parameter.

B.4.3. Verifier Metadata

The `format` value in the `vp_formats` parameter of the Verifier metadata MUST have the key `dc+sd-jwt`, and the value is an object consisting of the following name/value pairs:

- `sd-jwt_alg_values`: OPTIONAL. A JSON array containing identifiers of cryptographic algorithms the Verifier supports for signing of an Issuer-signed JWT of an SD-JWT. If present, the `alg` JOSE header (as defined in [RFC7515]) of the Issuer-signed JWT of the presented SD-JWT MUST match one of the array values.
- `kb-jwt_alg_values`: OPTIONAL. A JSON array containing identifiers of cryptographic algorithms the Verifier supports for signing of a Key Binding JWT (KB-JWT). If present, the `alg` JOSE header (as defined in [RFC7515]) of the presented KB-JWT MUST match one of the array values.

The following is a non-normative example of `client_metadata` request parameter value in a request to present an IETF SD-JWT VC.

```
{
  "vp_formats": {
    "dc+sd-jwt": {
      "sd-jwt_alg_values": ["ES256", "ES384"],
      "kb-jwt_alg_values": ["ES256", "ES384"]
    }
  }
}
```

B.4.4. DCQL Query and Response

This section defines SD-JWT VC specific DCQL Query and Response parameters.

B.4.4.1. Parameters in the `meta` parameter in Credential Query

The following is an SD-JWT VC specific parameter in the `meta` parameter in a Credential Query as defined in Section 6.1.

`vct_values`: OPTIONAL. An array of strings that specifies allowed values for the type of the requested Verifiable Credential. All elements in the array MUST be valid type identifiers as defined in [I-D.ietf-oauth-sd-jwt-vc]. The Wallet MAY return credentials that inherit from any of the specified types, following the inheritance logic defined in [I-D.ietf-oauth-sd-jwt-vc].

B.4.4.2. SD-JWT VC DCQL Query example

A non-normative example DCQL query using the SD-JWT VC format is shown in Section 7.4. The respective response is shown in Section 8.1.1.

Additional examples are shown in Appendix C.

B.4.5. Presentation Request

The following is a non-normative example of an Authorization Request:

```
GET /authorize?
  response_type=vp_token
  &client_id=x509_san_dns%3Aclient.example.org
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
  &dcql_query=%7B...%7D
  &nonce=n-0S6_WzA2Mj HTTP/1.1
Host: wallet.example.com
```

The following is a non-normative example of the contents of a `presentation_definition` parameter that contains the requirements regarding the Credential to be presented:

```
{
  "id": "example_sd_jwt_vc_request",
  "input_descriptors": [
    {
      "id": "identity_credential",
      "format": {
        "dc+sd-jwt": {
          "sd-jwt_alg_values": ["ES256", "ES384"],
          "kb-jwt_alg_values": ["ES256", "ES384"]
        }
      },
      "constraints": {
        "limit_disclosure": "required",
        "fields": [
          {
            "path": ["$.vct"],
            "filter": {
              "type": "string",
              "const": "https://credentials.example.com/identity_credential"
            }
          },
          {
            "path": ["$.family_name"]
          },
          {
            "path": ["$.given_name"]
          }
        ]
      }
    }
  ]
}
```

The presentation of an IETF SD-JWT VC is requested by adding an object named `dc+sd-jwt` to the `format` object of an `input_descriptor`. The `input_descriptor` value is applied to the unsecured payload of the IETF SD-JWT VC which correspond to the disclosures of the presented SD-JWT VC.

Setting `limit_disclosure` property defined in [DIF.PresentationExchange] to `required` enables selective release by instructing the Wallet to submit only the disclosures for the matching claims specified in the fields array. The unsecured payload of an IETF SD-JWT VC is used to perform the matching.

B.4.6. Presentation Response

A non-normative example of the Authorization Response would look the same as in the examples of other Credential formats in this Annex.

The following requirements apply to the `nonce` and `aud` claims in the Key Binding JWT:

- the nonce claim MUST be the value of nonce from the Authorization Request;
- the aud claim MUST be the value of the Client Identifier, except for requests over the DC API where it MUST be the Origin prefixed with `origin:`, as described in Appendix A.4.

The following is a non-normative example of the content of the `presentation_submission` parameter:

```
{
  "definition_id": "example_sd_jwt_vc_request",
  "id": "example_sd_jwt_vc_presentation_submission",
  "descriptor_map": [
    {
      "id": "identity_credential",
      "path": "$",
      "format": "dc+sd-jwt"
    }
  ]
}
```

The following is a non-normative example of the `vp_token` parameter provided in the same response and referred to by the `presentation_submission` above:

```
eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImRjK3NkLWp3dCIsICJraWQiOiAiZG9jLXNp
Z25lci0wNS0yNS0yMDIyIn0.eyJfc2QiOiBbIjNvVUNuYUt0N3dxREt1eWgtTGdRb3p6
ZmhnYjhnTzVOaS1SQ1dzV1cydkEiLCAiOHo4ejlYOWpVdGI5OWdqZWpDd0ZBR3o0YXFs
SGYtc0NxUTZlTV9xbXBVUSIsICJDeHE0ODcyVVhYbmdHVUxUX2tsOGZkd1ZGa3lLNkFK
ZlBaTHk3TDVfMGtJIiwgIlRHZjRvTGJnd2Q1SlFhSHlLVlFaVTlVZEdFMHc1cnREc3Ja
emZVYW9tTG8iLCAianN1OXlWdWx3UVFsaEZsTV8zSmx6TWFTRnpnbGhRRzBEcGZheVF3
TFVLNCIsICJzRmNWaUhOLUpHM2VUVXlCbVU0Zmt3dXN5NUkxU0xCaGUxak52S3hQNXhN
IiwgInRpVG5ncDlfamhDMzg5VVA4X2s2N01YcW9TZmlIcTNpSzZvOXVuNHdlX1kiLCAi
eHNLa0dKWEQxLWUzSTl6ajBZeUtOdi1sVTVZcWhzRUFGOU5oT3I4eGdhNCJdLCAiaXNz
IjogImh0dHBzOi8vZXhhbXBsZS5jb20vaXNzdWVyIiwgImlhdCI6IDE2ODMwMDAwMDAs
ICJleHAiOiAxODgzMDAwMDAwLCAidmN0IjogImh0dHBzOi8vY3JlZGVudGlhbHMuZXhh
bXBsZS5jb20vaWRlbnRpdHlfY3JlZGVudGlhbCIsICJfc2RfYWxnIjogInNoYS0yNTYi
LCAiY25mIjogeyJqd2siOiB7Imt0eSI6ICJFQyIsICJjcnYiOiAiUC0yNTYiLCAieCI6
ICJUQ0FFUjE5WnZ1M09IRjRqNFc0dmZTVm9ISVAxSUxpbERsczd2Q2VHZW1jIiwgInki
OiAiWnhqaVdXYlpNUUdIVldLVlE0aGJTSWlyc1ZmdWVjQ0U2dDRqVDlGMkhaUSJ9fX0.
8eHLENOFGlZ7dcHSOCYzTu6BuBN8PqYnJCcPgGUh6XoxF6U6S5NVZq40cuLyvJqHZ56x
DGeQch0lBjLRKvS4Rw~WyJlbHVWNU9nM2dTTklJOEVZbnN4QV9BIiwgImZhbWlseV9uY
W1lIiwgIkRvZSJd~WyIyR0xDNDJzS1F2ZUNmR2ZyeU5STjl3IiwgImdpdmVuX25hbWUi
LCAiSm9obiJd~eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImtiK2p3dCJ9.eyJub25jZS
I6ICJuLTBTNl9XekEyTWoiLCAiYXVkIjogImh0dHBzOi8vZXhhbXBsZS5jb20vdmVyaW
ZpZXIiLCAiaWF0IjogMTcwOTgzODYwNCwgInNkX2hhc2giOiAiRHktUll3WmZhYW9DM2
luSmJMc2xnUHZNcDA5YkgtY2xZUF8zcWJScXRXNCJ9.RmgIhqCHYWerxbDboMuB0lli6
3HPJHI9Vl2ZNOGh20C7_6p7nf3Wkd2wkx5WlmwTwtHKc87MBY2nuRLoeduQMA
```

In this example the `vp_token` contains only the disclosures for the claims specified in the `presentation_submission`, along with a Key Binding JWT.

The following is a non-normative example of the unsecured payload of the Key Binding JWT.

```
{
  "nonce": "n-0S6_WzA2Mj",
  "aud": "x509_san_dns:client.example.org",
  "iat": 1709838604,
  "sd_hash": "Dy-RYwZfaaoC3inJbLslgPvMp09bH-clYP_3qbRqtW4",
  "transaction_data_hashes": [ "fOBUSQvo46yQO-wRwXBcGqvnbKIueISEL961_Sjd4do" ]
}
```

B.5. Combining this specification with SIOPv2

This section shows how SIOP and OpenID for Verifiable Presentations can be combined to present Verifiable Credentials and pseudonymously authenticate an End-User using subject controlled key material.

B.5.1. Request

The following is a non-normative example of a request that combines this specification and [SIOPv2].

```
GET /authorize?
  response_type=vp_token%20id_token
  &scope=openid
  &id_token_type=subject_signed
  &client_id=x509_san_dns%3Aclient.example.org
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
  &presentation_definition=...
  &nonce=n-0S6_WzA2Mj HTTP/1.1
Host: wallet.example.com
```

The differences to the example requests in the previous sections are:

- `response_type` is set to `vp_token id_token`. This means the Wallet returns the `presentation_submission` and `vp_token` parameters in the same response as the `id_token` parameter as described in Section 8.
- The request includes the `scope` parameter with value `openid` making this an OpenID Connect request. Additionally, the request also contains the parameter `id_token_type` with value `subject_signed` requesting a Self-Issuer ID Token, i.e., the request is a SIOP request.

B.5.2. Response

The following is a non-normative example of a response sent upon receiving a request provided in Appendix B.5.1:

```
HTTP/1.1 302 Found
Location: https://client.example.org/cb#
  id_token=
  &presentation_submission=...
  &vp_token=...
```

In addition to the `presentation_submission` and `vp_token`, it also contains an `id_token`.

The following is a non-normative example of the payload of a Self-Issued ID Token [SIOPv2] contained in the above response:

```
{
  "iss": "did:example:NzbLsXh8uDCcd6MNwXF4W7noWXFZAfHkxZsRGC9Xs",
  "sub": "did:example:NzbLsXh8uDCcd6MNwXF4W7noWXFZAfHkxZsRGC9Xs",
  "aud": "x509_san_dns:client.example.org",
  "nonce": "n-0S6_WzA2Mj",
  "exp": 1311281970,
  "iat": 1311280970
}
```

Note: The `nonce` and `aud` are set to the `nonce` of the request and the Client Identifier of the Verifier, respectively, in the same way as for the Verifier, Verifiable Presentations to prevent replay.

Appendix C. Examples for DCQL Queries

The following is a non-normative example of a DCQL query that requests a Verifiable Credential in the format `mso_mdoc` with the claims `vehicle_holder` and `first_name`:

```
{
  "credentials": [
    {
      "id": "my_credential",
      "format": "mso_mdoc",
      "meta": {
        "doctype_value": "org.iso.7367.1.mVRC"
      },
      "claims": [
        {"path": ["org.iso.7367.1", "vehicle_holder"]},
        {"path": ["org.iso.18013.5.1", "first_name"]}
      ]
    }
  ]
}
```

The following is a non-normative example of a DCQL query that requests multiple Verifiable Credentials; all of them must be returned:

```
{
  "credentials": [
    {
      "id": "pid",
      "format": "dc+sd-jwt",
      "meta": {
        "vct_values": ["https://credentials.example.com/identity_credential"]
      },
      "claims": [
        {"path": ["given_name"]},
        {"path": ["family_name"]},
        {"path": ["address", "street_address"]}
      ]
    },
    {
      "id": "mdl",
      "format": "mso_mdoc",
      "meta": {
        "doctype_value": "org.iso.7367.1.mVRC"
      },
      "claims": [
        {"path": ["org.iso.7367.1", "vehicle_holder"]},
        {"path": ["org.iso.18013.5.1", "first_name"]}
      ]
    }
]
}
```

The following shows a complex query where the Wallet is requested to deliver the `pid` credential, or the `other_pid` credential, or both `pid_reduced_cred_1` and `pid_reduced_cred_2`. Additionally, the `nice_to_have` credential may optionally be delivered.

```
{
  "credentials": [
    {
      "id": "pid",
      "format": "dc+sd-jwt",
      "meta": {
        "vct_values": ["https://credentials.example.com/identity_credential"]
      },
      "claims": [
        {"path": ["given_name"]},
        {"path": ["family_name"]},
        {"path": ["address", "street_address"]}
      ]
    },
    {
      "id": "other_pid",
      "format": "dc+sd-jwt",
      "meta": {
        "vct_values": ["https://othercredentials.example/pid"]
      },
      "claims": [
        {"path": ["given_name"]},
        {"path": ["family_name"]},
        {"path": ["address", "street_address"]}
      ]
    },
    {
      "id": "pid_reduced_cred_1",
      "format": "dc+sd-jwt",
      "meta": {
        "vct_values": ["https://credentials.example.com/reduced_identity_credential"]
      },
      "claims": [
        {"path": ["family_name"]},
        {"path": ["given_name"]}
      ]
    },
    {
      "id": "pid_reduced_cred_2",
      "format": "dc+sd-jwt",
      "meta": {
        "vct_values": ["https://cred.example/residence_credential"]
      },
      "claims": [
        {"path": ["postal_code"]},
        {"path": ["locality"]},
        {"path": ["region"]}
      ]
    },
    {
      "id": "nice_to_have",
      "format": "dc+sd-jwt",
      "meta": {
        "vct_values": ["https://company.example/company_rewards"]
      },
      "claims": [
        {"path": ["rewards_number"]}
      ]
    }
  ],
  "credential_sets": [
    {
      "purpose": "Identification",
      "options": [
```

```
        [ "pid" ],
        [ "other_pid" ],
        [ "pid_reduced_cred_1", "pid_reduced_cred_2" ]
      ]
    },
    {
      "purpose": "Show your rewards card",
      "required": false,
      "options": [
        [ "nice_to_have" ]
      ]
    }
  ]
}
```

The following shows a query where an ID and an address are requested; either can come from an mDL or a photoid Credential.

```
{
  "credentials": [
    {
      "id": "mdl-id",
      "format": "mso_mdoc",
      "meta": {
        "doctype_value": "org.iso.18013.5.1.mDL"
      },
      "claims": [
        {
          "id": "given_name",
          "path": ["org.iso.18013.5.1", "given_name"]
        },
        {
          "id": "family_name",
          "path": ["org.iso.18013.5.1", "family_name"]
        },
        {
          "id": "portrait",
          "path": ["org.iso.18013.5.1", "portrait"]
        }
      ]
    },
    {
      "id": "mdl-address",
      "format": "mso_mdoc",
      "meta": {
        "doctype_value": "org.iso.18013.5.1.mDL"
      },
      "claims": [
        {
          "id": "resident_address",
          "path": ["org.iso.18013.5.1", "resident_address"]
        },
        {
          "id": "resident_country",
          "path": ["org.iso.18013.5.1", "resident_country"]
        }
      ]
    },
    {
      "id": "photo_card-id",
      "format": "mso_mdoc",
      "meta": {
```

```
        "doctype_value": "org.iso.23220.photoid.1"
      },
      "claims": [
        {
          "id": "given_name",
          "path": ["org.iso.18013.5.1", "given_name"]
        },
        {
          "id": "family_name",
          "path": ["org.iso.18013.5.1", "family_name"]
        },
        {
          "id": "portrait",
          "path": ["org.iso.18013.5.1", "portrait"]
        }
      ]
    },
    {
      "id": "photo_card-address",
      "format": "mso_mdoc",
      "meta": {
        "doctype_value": "org.iso.23220.photoid.1"
      },
      "claims": [
        {
          "id": "resident_address",
          "path": ["org.iso.18013.5.1", "resident_address"]
        },
        {
          "id": "resident_country",
          "path": ["org.iso.18013.5.1", "resident_country"]
        }
      ]
    }
  ],
  "credential_sets": [
    {
      "purpose": "Identification",
      "options": [
        [ "mdl-id" ],
        [ "photo_card-id" ]
      ]
    },
    {
      "purpose": "Proof of address",
      "required": false,
      "options": [
        [ "mdl-address" ],
        [ "photo_card-address" ]
      ]
    }
  ]
}
```

The following is a non-normative example of a DCQL query that requests

- the mandatory claims `last_name` and `date_of_birth`, and
- either the claim `postal_code`, or, if that is not available, both of the claims `locality` and `region`.

```
{
  "credentials": [
    {
      "id": "pid",
      "format": "dc+sd-jwt",
      "meta": {
        "vct_values": [ "https://credentials.example.com/identity_credential" ]
      },
      "claims": [
        {"id": "a", "path": ["last_name"]},
        {"id": "b", "path": ["postal_code"]},
        {"id": "c", "path": ["locality"]},
        {"id": "d", "path": ["region"]},
        {"id": "e", "path": ["date_of_birth"]}
      ],
      "claim_sets": [
        ["a", "c", "d", "e"],
        ["a", "b", "e"]
      ]
    }
  ]
}
```

The following example shows a query that uses the `values` constraints to request a credential with specific values for the `last_name` and `postal_code` claims:

```
{
  "credentials": [
    {
      "id": "my_credential",
      "format": "dc+sd-jwt",
      "meta": {
        "vct_values": [ "https://credentials.example.com/identity_credential" ]
      },
      "claims": [
          {
            "path": ["last_name"],
            "values": ["Doe"]
          },
          {"path": ["first_name"]},
          {"path": ["address", "street_address"]},
          {
            "path": ["postal_code"],
            "values": ["90210", "90211"]
          }
      ]
    }
  ]
}
```

Appendix D. IANA Considerations

D.1. OAuth Authorization Endpoint Response Types Registry

This specification registers the following `response_type` values in the IANA "OAuth Authorization Endpoint Response Types" registry [IANA.OAuth.Parameters] established by [RFC6749].

D.1.1. vp_token

- Response Type Name: `vp_token`
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Specification Document(s): Section 8 of this specification

D.1.2. vp_token id_token

- Response Type Name: `vp_token id_token`
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Specification Document(s): Section 8 of this specification

D.2. OAuth Parameters Registry

This specification registers the following OAuth parameters in the IANA "OAuth Parameters" registry [IANA.OAuth.Parameters] established by [RFC6749].

D.2.1. presentation_definition

- Name: `presentation_definition`
- Parameter Usage Location: authorization request
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 5.1 of this specification

D.2.2. presentation_definition_uri

- Name: `presentation_definition_uri`
- Parameter Usage Location: authorization request
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 5.1 of this specification

D.2.3. dcql_query

- Name: `dcql_query`
- Parameter Usage Location: authorization request
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 5.1 of this specification

D.2.4. client_metadata

- Name: `client_metadata`
- Parameter Usage Location: authorization request
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 5.1 of this specification

D.2.5. request_uri_method

- Name: `request_uri_method`

- Parameter Usage Location: authorization request
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 5.1 of this specification

D.2.6. transaction_data

- Name: `transaction_data`
- Parameter Usage Location: authorization request
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 5.1 of this specification

D.2.7. wallet_nonce

- Name: `wallet_nonce`
- Parameter Usage Location: authorization request, token response
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 5.11 of this specification

D.2.8. response_uri

- Name: `response_uri`
- Parameter Usage Location: authorization request
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 8.2 of this specification

D.2.9. vp_token

- Name: `vp_token`
- Parameter Usage Location: authorization response, token response
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 8.1 of this specification

D.2.10. presentation_submission

- Name: `presentation_submission`
- Parameter Usage Location: authorization response, token response
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 8.1 of this specification

D.2.11. expected_origins

- Name: `expected_origins`
- Parameter Usage Location: authorization request
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Appendix A.2 of this specification

D.3. OAuth Extensions Error Registry

This specification registers the following errors in the IANA "OAuth Extensions Error" registry [IANA.OAuth.Parameters] established by [RFC6749].

D.3.1. vp_formats_not_supported

- Name: `vp_formats_not_supported`
- Usage Location: authorization endpoint, token endpoint
- Protocol Extension: OpenID for Verifiable Presentations
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 8.5 of this specification

D.3.2. invalid_presentation_definition_uri

- Name: `invalid_presentation_definition_uri`
- Usage Location: authorization endpoint, token endpoint
- Protocol Extension: OpenID for Verifiable Presentations
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 8.5 of this specification

D.3.3. invalid_presentation_definition_reference

- Name: `invalid_presentation_definition_reference`
- Usage Location: authorization endpoint, token endpoint
- Protocol Extension: OpenID for Verifiable Presentations
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 8.5 of this specification

D.3.4. invalid_request_uri_method

- Name: `invalid_request_uri_method`
- Usage Location: authorization endpoint
- Protocol Extension: OpenID for Verifiable Presentations
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 8.5 of this specification

D.3.5. wallet_unavailable

- Name: `wallet_unavailable`
- Usage Location: authorization endpoint, token endpoint
- Protocol Extension: OpenID for Verifiable Presentations
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 8.5 of this specification

D.4. OAuth Authorization Server Metadata Registry

This specification registers the following authorization server metadata parameters in the IANA "OAuth Authorization Server Metadata" registry [IANA.OAuth.Parameters] established by [RFC8414].

D.4.1. presentation_definition_uri_supported

- Metadata Name: `presentation_definition_uri_supported`
- Metadata Description: Boolean value specifying whether the Wallet supports the transfer of presentation_definition by reference
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 10 of this specification

D.4.2. vp_formats_supported

- Metadata Name: `vp_formats_supported`
- Metadata Description: An object containing a list of name/value pairs, where the name is a string identifying a Credential format supported by the Wallet
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 10 of this specification

D.5. OAuth Dynamic Client Registration Metadata Registry

This specification registers the following client metadata parameters in the IANA "OAuth Dynamic Client Registration Metadata" registry [IANA.OAuth.Parameters] established by [RFC7591].

D.5.1. vp_formats

- Client Metadata Name: `vp_formats`
- Client Metadata Description: An object defining the formats and proof types of Verifiable Presentations and Verifiable Credentials that a Verifier supports
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 11.1 of this specification

D.6. Media Types Registry

This section registers the following media type [RFC2046] in the IANA "Media Types" registry <xref target="IANA.MediaTypes"/> in the manner described in [RFC6838].

D.6.1. application/verifier-attestation+jwt

The media type for a Verifier Attestation JWT is `application/verifier-attestation+jwt`.

- Type name: `application`
- Subtype name: `verifier-attestation+jwt`
- Required parameters: n/a
- Optional parameters: n/a
- Encoding considerations: Uses JWS Compact Serialization as defined in [RFC7515].
- Security considerations: See Security Considerations in in [RFC7519].

- Interoperability considerations: n/a
- Published specification: Section 12 of this specification
- Applications that use this media type: Applications that issue, present, verify verifier attestation VCs
- Additional information:
  - Magic number(s): n/a
  - File extension(s): n/a
  - Macintosh file type code(s): n/a
- Person & email address to contact for further information: TBD
- Intended usage: COMMON
- Restrictions on usage: none
- Author: Oliver Terbu, oliver.terbu@mattr.global
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net

D.7. JSON Web Signature and Encryption Header Parameters Registry

This specification registers the following JWS header parameter in the IANA "JSON Web Signature and Encryption Header Parameters" registry [IANA.JOSE] established by [RFC7515].

D.7.1. jwt

- Header Parameter Name: `jwt`
- Header Parameter Description: This header contains a JWT. Processing rules MAY depend on the `typ` header value of the respective JWT.
- Header Parameter Usage Location: JWS
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Specification Document(s): Section 12 of this specification

D.7.2. client_id

- Header Parameter Name: `client_id`
- Header Parameter Description: This header contains a Client Identifier. A Client Identifier is used in OAuth to identify a certain client. It is defined in [RFC6749], section 2.2.
- Header Parameter Usage Location: JWS
- Change Controller: IETF
- Specification Document(s): [RFC6749]

D.8. Uniform Resource Identifier (URI) Schemes Registry

This specification registers the following URI scheme in the IANA "Uniform Resource Identifier (URI) Schemes" registry [IANA.URI.Schemes].

D.8.1. openid4vp

- URI Scheme: `openid4vp`
- Description: Custom scheme used for wallet invocation
- Status: Provisional
- Well-Known URI Support: -

- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 13.1.2 of this specification

Appendix E. Acknowledgements

We would like to thank Richard Barnes, Paul Bastian, Vittorio Bertocci, Christian Bormann, John Bradley, Marcos Caceres, Brian Campbell, Lee Campbell, Tim Cappalli, Gabe Cohen, David Chadwick, Andrii Deinega, Rajvardhan Deshmukh, Giuseppe De Marco, Mark Dobrinic, Daniel Fett, Pedro Felix, George Fletcher, Ryan Galluzzo, Timo Glasta, Sam Goto, Mark Haine, Martijn Haring, Fabian Hauck, Roland Hedberg, Joseph Heenan, Bjorn Hjelm, Alen Horvat, Andrew Hughes, Jacob Ideskog, Łukasz Jaromin, Edmund Jay, Michael B. Jones, Tom Jones, Judith Kahrer, Takahiko Kawasaki, Gaurav Khot, Niels Klomp, Ronald Koenig, Markus Kreusch, Adam Lemmon, Hicham Lozi, Daniel McGrogan, Jeremie Miller, Kenichi Nakamura, Gareth Oliver, Andreea Prian, Rolson Quadras, Javier Ruiz, Nat Sakimura, Arjen van Veen, Steve Venema, Jan Vereecken, David Waite, Jacob Ward, David Zeuthen for their valuable feedback and contributions to this specification.

Appendix F. Notices

Appendix G. Document History

[[To be removed from the final specification]]

-25

- clarify value matching in DCQL
- clarify why requests using redirect_uri scheme cannot be signed

- add `trusted_authorities` to DCQL
- add note introducing cbor and cddl
- clarify DCQL case of `claims` and `claim_sets` being absent
- add language on client ID and nonce binding for ISO mdocs and W3C VCs
- for DC API, always use Origin for binding the response (e.g. in Key Binding JWT aud and sessionTranscript in mdoc)
- clarify the behavior is not to sign when authorization_signed_response_alg is omitted
- add a note on the use of apu/apv in the JWE header of encrypted responses
- add x509_hash client identifier scheme
- remove x509_san_uri client identifier scheme
- clarify that `dcql_query` and `presentation_definition` are passed as JSON objects (not strings) in request objects
- support returning multiple presentations for a single dcql credential query when requested using `multiple`
- Added support for multiple Client Identifiers and corresponding Request Signature to the DC API profile

-24

- add mdoc specific `intent_to_retain` mechanism, using the definition from 18013-5
- require `typ` value in request object to be `oauth-authz-req+jwt`
- add `SessionTranscript` requirements
- use claims path pointer for mdoc based credentials

-23

- fixed percent-encoding of URI examples
- fixed an example that used 'client' where 'wallet' is more appropriate
- make SIOP example request/response consistent with each other
- make example request and example SD-JWT key binding JWT consistent
- add note that there are a choice of encryption JWE algorithms available, including the HPKE draft
- add `transaction_data` & `dcql_query` to list of allowed parameters in W3C Digital Credentials API appendix
- change credential format identifier `vc+sd-jwt` to `dc+sd-jwt` to align with the media type in draft -06 of [I-D.ietf-oauth-sd-jwt-vc] and update `typ` accordingly in examples
- remove references to the openid4vci credential format section
- clarified what profiling OID4VP means
- moved credential format specific DCQL parameters to the annex
- generalized W3C Digital Credentials API references
- changed response mode value for the OID4VP over the DC API
- updated to PE ver 2.1.1 (used to be 2.0.0)

-22

- Introduced the Digital Credentials Query Language
- add transaction data mechanism
- remove `client_id_scheme` and turn it into a prefix of the `client_id`; this addresses a security issue with the previous solution
- Clarified what can go in the `client_metadata` parameter
- Fixed #227: Enabled non-breaking extensibility.

- Fixed #383: Completed IANA Considerations section.

-21

- removed `client_metadata_uri` authorization parameter
- added how OpenID4VP request/response can be used over the browser API
- remove path_nested description from Response Parameters section and move it into W3C VC Annex
- fix indentation of examples
- added references to ISO/IEC 23220 and 18013 documents
- added `post` request method for Request URI
- Added IETF SD-JWT VC profile
- Added `wallet_unavailable` error

-20

- added "verifier_attestation" client id scheme value

-19

- added "x509_san_uri" and "x509_san_dns" client id scheme value

-18

- editorial update based on the 45 days review period prior to the Vote for proposed Second Implementer's Draft

-17

- direct_post response mode uses state to identify response
- Added sequence diagrams for same and cross device flows to overview section

-16

- Added `client_id_scheme` parameter
- Defined that single VP Tokens must not use the array syntax for single Verifiable Presentations

-15

- Added definition of VP Token
- Editorial improvements for better readability (restructured request and response section, consistent terminology, and casing)

-14

- added support for signed and encrypted authorization responses based on JARM
- clarified response encoding for authorization responses
- moved invocation/just-in-time client metadata exchange/AS Discovery sections from siopv2 to openid4vp

-13

- added scope support

-12

- add Cross-Device flow (using SIOP v2 text)
- Added Client Metadata Section (based on SIOP v2 text)

-11

- changed base protocol to OAuth 2.0
- consolidated the examples

-10

- Added AnonCreds example
- Added ISO mobile Driving License (mDL) example

-09

- added support for passing presentation_definition by reference
- added description how to requset credential issued by a member of a federation

-08

- reflected editorial comments received during pre-implementer's draft review period

-07

- added text on other credential formats
- fixed inconsistency in security consideration regarding nonce

-06

- added additional security considerations
- removed support for embedding Verifiable Presentations in ID Token or UserInfo response
- migrated to Presentation Exchange 2.0

-05

- moved presentation submission parameters outside of Verifiable Presentations (ID Token or UserInfo)

-04

- added presentation submission support
- cleaned up examples to use `nonce` & `client_id` instead of `vp_hash` for replay detection
- fixed further nits in examples
- added and reworked references to other specifications

-03

- aligned with SIOP v2 spec

-02

- added `presentation_definition` as sub parameter of `verifiable_presentation` and VP Token

-01

- adopted DIF Presentation Exchange request syntax

- added security considerations regarding replay detection for Verifiable Credentials

-00

- initial revision

Authors' Addresses

Oliver Terbu
Mattr
Email: oliver.terbu@mattr.global

Torsten Lodderstedt
SPRIND
Email: torsten@lodderstedt.net

Kristina Yasuda
SPRIND
Email: kristina.yasuda@sprind.org

Tobias Looker
Mattr
Email: tobias.looker@mattr.global