Workgroup: connect
Published: 21 April 2023
Authors: O. Terbu, *Spruce Systems, Inc.*; T. Lodderstedt, *yes.com*; K. Yasuda, *Microsoft*; T. Looker, *Mattr*

OpenID for Verifiable Presentations - draft 18

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], 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 RFC 2119 [RFC2119].

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", "Entity", "Request Object", "Request URI" as defined by OpenID Connect Core [OpenID.Core], the term "JSON Web Token (JWT)" defined by JSON Web Token (JWT) [RFC7519], the term "JOSE Header" and the term "Base64url Encoding" 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].

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

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].

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], [ISO.18013-5] (mdoc) and [Hyperledger.Indy] (AnonCreds).

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

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.

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], [ISO.18013-5] (mdoc) and [Hyperledger.Indy] (AnonCreds).

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

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

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

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].

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.

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

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.

Claim-based Holder Binding: Ability of the Holder to prove legitimate possession of a Verifiable Credential by proofing certain claims, e.g., name and date of birth, for example by presenting another Verifiable Credential. Claim-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.

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

VP Token: An artifact defined in this specification that contains a single Verifiable Presentation or an array of Verifiable Presentations as defined in Section 6.1.

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], [ISO.18013-5] (mdoc), 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 A.

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 HTTPS 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 [I-D.ietf-oauth-security-topics], can be implemented on top of this specification.

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.

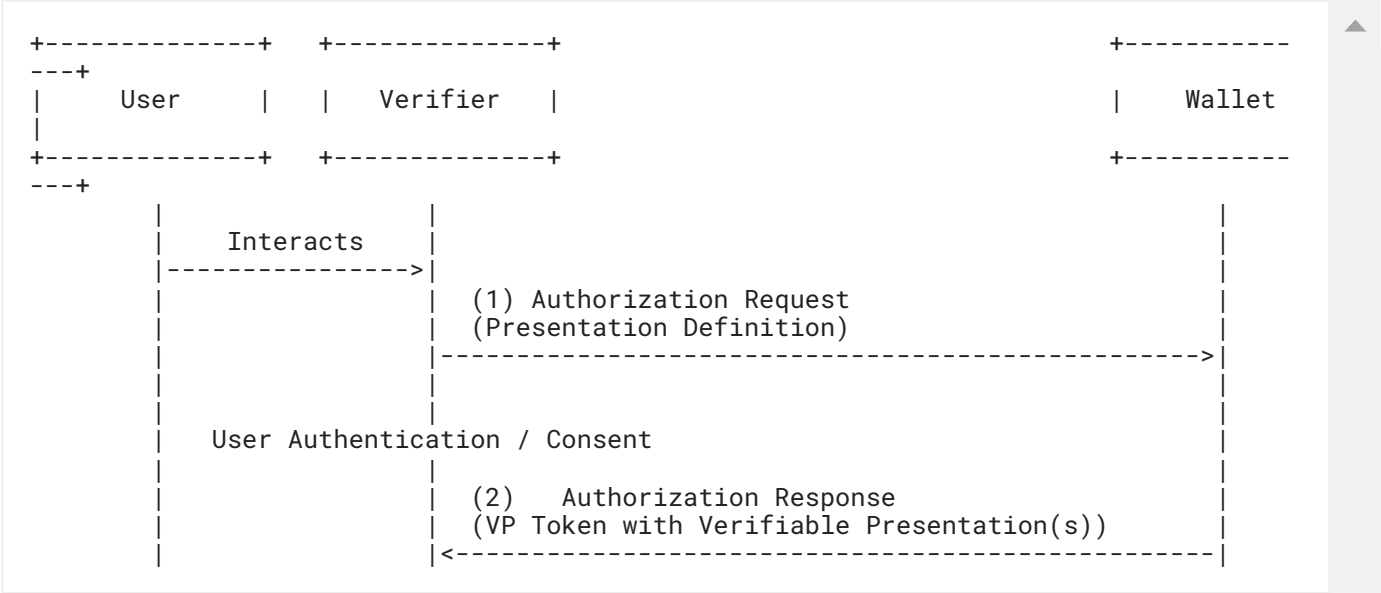


Figure 1: Same Device Flow

(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 User then uses the Wallet to scan the QR Code. The Verifiable Presentations are sent to the Verifier in a direct HTTPS 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.

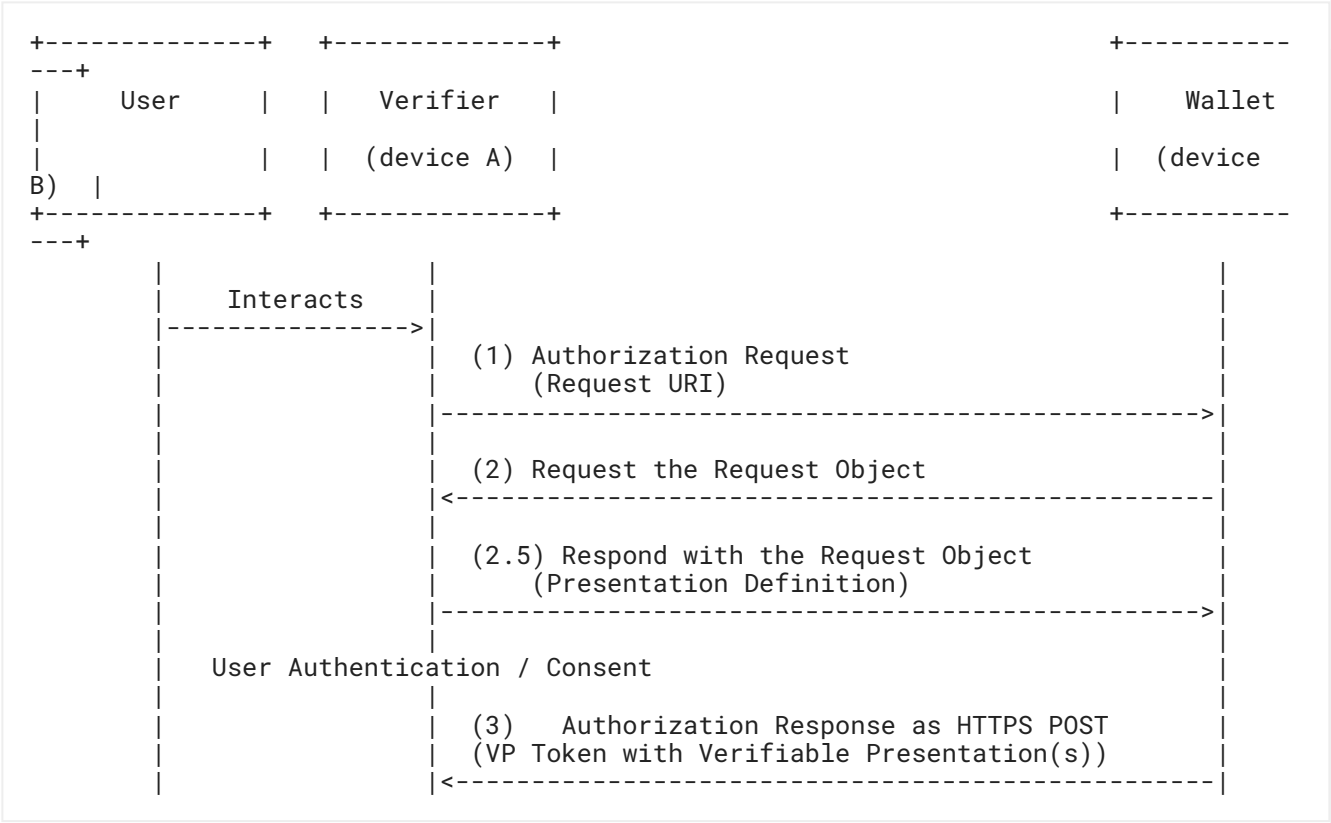


Figure 2: Cross Device Flow

(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 HTTPS GET request to the Request URI to retrieve the Request Object.

(2.5) The HTTPS 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 `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 6 for more details.
- New Response Types `vp_token` and `id_token vp_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 6 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 6.2 for more details.
- The [DIF.PresentationExchange] `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 A are given for Credential formats as specified in [VC_DATA], [ISO.18013-5], and [Hyperledger.Indy].
- A new `client_id_scheme` Authorization Request parameter is defined 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 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].

The Verifier may send an Authorization Request as Request Object by value or by reference as defined in JWT-Secured Authorization Request (JAR) [RFC9101].

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 5 of [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 using `client_id_scheme` parameter. This parameter 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 `client_metadata` and `client_metadata_uri` parameters that contain key value pairs defined in Section 4.3 and Section 2.1 of the OpenID Connect Dynamic Client Registration 1.0 [OpenID.Registration] specification as well as [RFC7591]. The parameter names include a term `client` since the Verifier is acting as an OAuth 2.0 Client.

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

This specification defines the following new parameters:

`presentation_definition`: A string containing a Presentation Definition JSON object. See Section 5.1 for more details. This parameter MUST be present when `presentation_definition_uri` parameter, or a `scope` value representing a Presentation Definition is not present.

`presentation_definition_uri`: A string containing an HTTPS URL pointing to a resource where a Presentation Definition JSON object can be retrieved. This parameter MUST be present when `presentation_definition` parameter, or a `scope` value representing a Presentation Definition is not present. See Section 5.2 for more details.

`client_id_scheme`: OPTIONAL. A string identifying the scheme of the value in the `client_id` Authorization Request parameter (Client Identifier scheme). The `client_id_scheme` parameter namespaces the respective Client Identifier. If an Authorization Request uses the `client_id_scheme` parameter, the Wallet MUST interpret the Client Identifier of the Verifier in the context of the Client Identifier scheme. If the parameter is not present, the Wallet MUST behave as specified in [RFC6749]. See Section 5.7 for the values defined by this specification. If the same Client Identifier is used with different Client Identifier schemes, those occurrences MUST be treated as different Verifiers. 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.

`client_metadata`: OPTIONAL. A JSON object containing the Verifier metadata values. It MUST be UTF-8 encoded. It MUST NOT be present if `client_metadata_uri` parameter is present.

`client_metadata_uri`: OPTIONAL. A string containing an HTTPS URL pointing to a resource where a JSON object with the Verifier metadata can be retrieved. The scheme used in the `client_metadata_uri` value MUST be `https`. The `client_metadata_uri` value MUST be reachable by the Wallet. It MUST NOT be present if `client_metadata` parameter is present.

A public key to be used by the Wallet as an input to the key agreement to encrypt Authorization Response (see Section 6.3). It MAY be passed by the Verifier using the `jwks` or the `jwks_uri` claim within the `client_metadata` or `client_metadata_uri` request parameter.

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 11.1 for details.

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 5.3 for more details.

`response_mode`: OPTIONAL. 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 6.2 for more details). It is also used to request signing and encrypting (see Section 6.3 for more details). If the parameter is not present, the default value is `fragment`.

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

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

5.1. `presentation_definition` Parameter

This parameter contains a Presentation Definition JSON object conforming to the syntax defined in Section 5 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 8).

The formats supported by a Verifier may be set up using the metadata parameter `vp_formats` (see Section 9.1). 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.

5.2. `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 HTTPS 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 HTTPS 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 HTTPS GET response sent by the Verifier in response to the above HTTPS 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"
                        }
                    }
                ]
            }
        }
    ]
}
```

5.3. 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 Presentation Definition 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 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.

Such definition of a 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 6.1.

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 5.1:

```
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
```

5.4. 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 6 on how the `response_type` value determines the response used to return a VP Token.

5.5. 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 6.2) in conjunction with `request_uri` is RECOMMENDED, since Authorization Request size might be large and might not fit in a QR code.

5.6. 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.

5.7. Verifier Metadata Management

The `client_id_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_id_scheme` is used since the Verifier is acting as an OAuth 2.0 Client.

This specification defines the following values for the `client_id_scheme` parameter, followed by the examples where applicable:

- `pre-registered`: This value represents 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.
- `redirect_uri`: This value indicates that the Verifier's redirect URI is also the value of the Client Identifier. In this case, the Authorization Request MUST NOT be signed, the Verifier MAY omit the `redirect_uri` Authorization Request parameter, and all Verifier metadata parameters MUST be passed using the `client_metadata` or `client_metadata_uri` parameter defined in Section 5.

The following is a non-normative example of a request when `client_id` equals `redirect_uri`.

```
HTTP/1.1 302 Found
Location: https://client.example.org/universal-link?
  response_type=vp_token
  &client_id=https%3A%2F%2Fclient.example.org%2Fcb
  &client_id_scheme=redirect_uri
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
  &presentation_definition=...
  &nonce=n-0S6_WzA2Mj
  &client_metadata=%7B%22vp_formats%22:%7B%22jwt_vp%22:%
  7B%22alg%22:%5B%22EdDSA%22,%22ES256K%22%5D%7D,%22ldp
  _vp%22:%7B%22proof_type%22:%5B%22Ed25519Signature201
  8%22%5D%7D%7D%7D
```

- `entity_id`: This value indicates that the Client Identifier is an Entity Identifier defined in OpenID Connect Federation [OpenID.Federation]. 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 Wallet MUST obtain Verifier metadata only from the Entity Statement(s). The `client_metadata` or `client_metadata_uri` parameter MUST NOT be present in the Authorization Request when this Client Identifier scheme is used.

- `did`: This value indicates that the Client Identifier is a DID defined in [DID-Core]. 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` or the `client_metadata_uri` parameter as defined in Section 5.

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",
   "client_id_scheme": "did",
   "response_types": "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"
            ]
         }
      }
   }
}
```

To use `client_id_scheme` values `entity_id` and `did`, 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 values for the `client_id_scheme` parameter. It is RECOMMENDED to use collision-resistant names for such values.

6. Response

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

VP Token can be returned in the Authorization Response or the Token Response depending on the Response Type used. See Section 5.4 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.

6.1. Response Parameters

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

`vp_token`: REQUIRED. JSON String or JSON object that MUST contain a single Verifiable Presentation or an array of JSON Strings and JSON objects each of them containing a Verifiable Presentations. Each Verifiable Presentation MUST be represented as a JSON string (that is a Base64url encoded value) or a JSON object depending on a format as defined in Annex E of [OpenID.VCI]. When a single Verifiable Presentation is returned, the array syntax MUST NOT be used. If Appendix E of [OpenID.VCI] 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 JSON string.

`presentation_submission`: REQUIRED. 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. 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 the value of the `path` field in it will ultimately depend on the credential format. Non-normative examples can be found further in this section.

Other parameters, such as `state` or `code` (from [RFC6749]), or `id_token` (from [OpenID.Core]), and `iss` (from [RFC9207]) MAY be included in the response as defined in the respective specifications.

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.

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=...
```

The following is a non-normative example of a VP Token containing a single Verifiable Presentation:

```
{
    "@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 a second Presentation Definition example in Section 5.1:

```
{
    "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"
        }
    },
```

```
"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6ImRpZDpleGFtcGxlOjB4YWJjI2tleTEifQ.
eyJpc3MiOiJkaWQ6ZXhhbXBsZTplYmZlYjFmNzEyZWJjNmYxYzI3NmUxMmVjMjEiLCJqdGkiOiJ1cm46
dXVpZDozOTc4MzQ0Zi04NTk2LTRjM2EtYTk3OC04ZmNhYmEzOTAzYzUiLCJhdWQiOiJkaWQ6ZXhhbXBs
ZTo0YTU3NTQ2OTczNDM2ZjZmNmM0YTRhNTc1NzMiLCJuYmYiOjE1NDE0OTM3MjQsImlhdCI6MTU0MTQ5
MzcyNCwiZXhwIjoxNTczMDI5NzIzLCJub25jZSI6IjM0M3MkRlNGRGEtIiwidnAiOnsiQGNvbnRleHQi
OlsiaHR0cHM6Ly93d3cudzMub3JnLzIwMTgvY3JlZGVudGlhbHMvdjEiLCJodHRwczovL3d3dy53My5v
cmcvMjAxOC9jcmVkZW50aWFscy92MSJdLCJ0eXBlIjpbIlZlcmlmaWFibGVQcmVzZW50
YXRpb24iLCJDcmVkZW50aWFsTWFuYWdlclByZXNlbnRhdGlvbiJdLCJ2ZXJpZmlhYmxlQ3JlZGVudGlh
bCI6WyJleUpoYkdjaU9pSlNVekkxTmlJc0luUjVjQ0k2SWtwWFZDSXNJbXRwWkNJNkltUnBaRHBsZUdG
dGNHeGxPbUZpWm1VeE0yWTNNVEl4TWpBME16RmpNamMyWlRFeVpXTmhZaU5yWlhsekxURWlmUS5leUp6
ZFdJaU9pSmthV1E2WlhoaGJYQnNaVHBsWW1abFlqRm1OekV5WldKak5tWXhZekkzTm1VeE1tVmpNakVp
TENKcWRHa2lPaUpvZEhSd09pOHZaWGhoYlhCc1pTNWxaSFV2WTNKbFpHVnVkR2xoYkhNdk16Y3pNaUlz
SW1semN5STZJbWgwZEhCek9pOHZaWGhoYlhCc1pTNWpiMjB2YTJWNWN5OW1iMjh1YW5kcklpd2libUpt
SWpveE5UUXhORGt6TnpJMExDSnBZWFFpT2pFMU5ERTBPVE0zTWpRc0ltVjRjQ0k2TVRVM016QXlPVGN5
TXl3aWJtOXVZMlVpT2lJMk5qQWhOak0wTlVaVFpYSWlMQ0oyWXlJNmV5SkFZMjl1ZEdWNGRDSTZXeUpv
ZEhSd2N6b3ZMM2QzZHk1M015NXZjbWN2TWpBeE9DOWpjbVZrWlc1MGFXRnNjeTkyTVNJc0ltaDBkSEJ6
T2k4dmQzZDNMbmN6TG05eVp5OHlNREU0TDJOeVpXUmxiblJwWVd4ekwyVjRZVzF3YkdWekwzWXhJbDBz
SW5SNWNHVWlPbHNpVm1WeWFXWnBZV0pzWlVOeVpXUmxiblJwWVd3aUxDSlZibWwyWlhKemFYUjVSR1Zu
Y21WbFEzSmxaR1Z1ZEdsaGJDSmRMQ0pqY21Wa1pXNTBhV0ZzVTNWaWFtVmpkQ0k2ZXlKa1pXZHlaV1Vp
T25zaWRIbHdaU0k2SWtKaFkyaGxiRzl5UkdWbmNtVmxJaXdpYm1GdFpTSTZJanh6Y0dGdUlHeGhibWM5
SjJaeUxVTkJKejVDWVdOallXeGhkWExEcVdGMElHVnVJRzExYzJseGRXVnpJRzUxYmNPcGNtbHhkV1Z6
UEM5emNHRnVQaUo5ZlgxOS5LTEpvNUdBeUJORDNMRFRuOUg3RlFva0VzVUVpOGpLd1hoR3ZvTjNKdFJh
NTF4ck5EZ1hEYjBjcTFVVFlCLXJLNEZ0OVlWbVIxTklfWk9GOG9HY183d0FwOFBIYkYySGFXb2RRSW9P
Qnh4VC00V05xQXhmdDdFVDZsa0gtNFM2VXgzclNHQW1jek1vaEVFZjhlQ2VOLWpDOFdla2RQbDZ6S1pR
ajBZUEIxcng2WDAteGxGQnM3Y2w2V3Q4cmZCUF90WjlZZ1ZXclFtVVd5cFNpb2MwTVV5aXBobXlFYkxa
YWdUeVBsVXlmbEdsRWRxclpBdjZlU2U2UnR4Snk2TTEtbEQ3YTVIVHphbllUV0JQQVVIRFpHeUdLWGRK
dy1XX3gwSVdDaEJ6STh0M2twRzI1M2ZnNlYzdFBnSGVLWEU5NGZ6X1FwWWZnLS03a0xzeUJBZlFHYmci
```

```
        XX19.ft_Eq4IniBrr7gtzRfrYj8Vy1aPXuFZU-6_ai0wvaKcsrzI4JkQEKTvbJwdvIeuGuTqy7ipO-
EY

i7V4TvonPuTRdpB7ZHOlYlbZ4wA9WJ6mSVSqDACvYRiFvrOFmie8rgm6GacWatgO4m4NqiFKFko3r58L
        ueFfGw47NK9RcfOkVQeHCq4btaDqksDKeoTrNysF4YS89INa-
prWomrLRAhnwLOo1Etp3E4ESAxg73CR
        2kA5AoMbf5KtFueWnMcSbQkMRdWcGC1VssC0tB0JffVjq7ZV6OTyV4kl1-
UVgiPLXUTpupFfLRhf9Qpq
        MBjYgP62KvhIvW8BbkGUelYMetA"
]
```

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 examples in this specification:

```
{
    "id": "Presentation example 2",
    "definition_id": "Example with multiple VPs",
    "descriptor_map": [
        {
            "id": "ID Card with constraints",
            "format": "ldp_vp",
            "path": "$[0]",
            "path_nested": {
                "format": "ldp_vc",
                "path": "$[0].verifiableCredential[0]"
            }
        },
        {
            "id": "Ontario Health Insurance Plan",
            "format": "jwt_vp_json",
            "path": "$[1]",
            "path_nested": {
                "format": "jwt_vc_json",
                "path": "$[1].vp.verifiableCredential[0]"
            }
        }
    ]
}
```

6.2. 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 HTTPS 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 HTTPS POST request to an endpoint controlled by the Verifier. The Authorization Response parameters are encoded in the body using the `application/x-www-form-urlencoded` content type. The flow can end with an HTTPS POST request from the Wallet to the Verifier, or it can end with a redirect that follows the HTTPS POST request, if the Verifier responds with a redirect URI to the Wallet.

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

`response_uri`: OPTIONAL. The Response URI to which the Wallet MUST send the Authorization Response using an HTTPS 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.

Note: The Verifier's component providing the user interface (Frontend) and the Verifier's component providing the Response URI (Response Endpoint) 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 10.5.

Note: If the Client Identifier scheme `redirect_uri` is used in conjunction with the Response Mode `direct_post`, and the `redirect_uri` parameter is present, the `client_id` value MUST be equal to the `response_uri` value.

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

```
{
   "client_id": "https://client.example.org/post",
   "client_id_scheme": "redirect_uri",
   "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 deep 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 HTTPS POST request to the Verifier's Response Endpoint:

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

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

If the Response Endpoint has successfully processed the request, it MUST respond with HTTPS status code 200.

The following new parameter is defined for use in the response from the endpoint:

`redirect_uri`: OPTIONAL. When the redirect parameter is used the Wallet MUST send the User Agent to this redirect URI. The redirect URI 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 especially enables the Verifier to prevent session fixation (Section 11.2) attacks.

Note: Response Mode `direct_post` without the `redirect_uri` could be less secure than Response Modes with redirects. For details, see (Section 11.2).

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 number in the URL. This number is used to ensure only the receiver of the redirect can fetch and process the Authorization Response. The number could be added as a path component or a parameter to the URL. It is RECOMMENDED to use a cryptographic random value of 128 bits or more at the time of the writing of this specification. For implementation considerations see Section 10.5.

The following is a non-normative example of the response from the Verifier to the Wallet upon receiving the Authorization Response at the Response Endpoint (using a `response_code` parameter from Section 10.5):

```
  HTTP/1.1 200 OK
  Content-Type: application/json;charset=UTF-8
  Cache-Control: no-store

  {

"redirect_uri":"https://client.example.org/cb#response_code=091535f699ea575c7937fa5f0f4
54aee"
  }
```

If the response does not contain a parameter, the Wallet is not required by this specification 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.

6.3. Signed and Encrypted Responses

This section defines how 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, or sign and encrypt the Authorization Response, implementations MAY use JWT Secured Authorization Response Mode for OAuth 2.0 (JARM) [JARM].

This specification also defines how to encrypt an unsigned Authorization Response by extending 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 11.5.

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.

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": "$",
                "format": "jwt_vp",
                "path_nested": {
                    "path": "$.vp.verifiableCredential[0]",
                    "format": "jwt_vc"
                }
            }
        ]
    }
}
```

The JWT response document MUST include `vp_token` and `presentation_submission` parameters as defined in Section 6.1.

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` or `jwks_uri` 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.

6.3.1. 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 6.2.

The Response Mode `direct_post.jwt` causes the Wallet to send the Authorization Response using an HTTPS POST request instead of redirecting back to the Verifier as defined in Section 6.2. The Wallet adds the `response` parameter containing the JWT as defined in Section 4.1. of [JARM] and Section 6.3 in the body of an HTTPS POST request using the `application/x-www-form-urlencoded` content type.

The following is a non-normative example of a response using the `presentation_submission` and `vp_token` values from Appendix A.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": "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": "$",
                "format": "jwt_vp",
                "path_nested": {
                    "path": "$.vp.verifiableCredential[0]",
                    "format": "jwt_vc"
                }
            }
        ]
    }
}
```

6.4. 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.
- Requested Presentation Definition does not conform to the DIF PEv2 specification [DIF.PresentationExchange].
- The Wallet does not support the `client_id_scheme` value passed in the Authorization Request.

- The Client Identifier passed in the request did not belong to the Client Identifier scheme indicated in the Authorization Request, or requirements of a certain scheme was violated, for example an unsigned request was sent with Client Identifier scheme `entity_id`.

`invalid_client`:

- `client_metadata` or `client_metadata_uri` parameter defined in Section 5 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.

Usage of `client_metadata` or `client_metadata_uri` parameters with `client_id` that the Wallet might be seeing for the first time is mutually exclusive with the registration mechanism where Self-Issued OP assigns `client_id` to the Verifier after receiving Verifier metadata.

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.

6.5. VP Token Validation

Verifiers MUST validate the VP Token in the following manner:

1. 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 11.1 for the checks required to prevent replay of a VP.
3. If applicable, 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 sent in the Presentation Definition in the Authorization Request.
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].

7. Wallet Invocation

The Verifier has the choice of the following mechanisms to invoke a Wallet:

- Custom URL scheme as an `authorization_endpoint` (for example, `openid4vp://` as defined in Section 10.1.2)
- Domain-bound Universal Links/App link as an `authorization_endpoint`
- no specific `authorization_endpoint`, user scanning a QR code with Authorization Request using a manually opened Wallet, instead of an arbitrary camera application on a user-device (neither custom URL scheme nor Universal/App link is used)

8. 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.

8.1. 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 key value pairs, where the key is a string identifying a Credential format supported by the Wallet. Valid Credential format identifier values are defined in Annex E of [OpenID.VCI]. 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`: 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 A.

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 JSON Strings containing the values of the Client Identifier schemes that the Wallet supports. The values defined by this specification are `pre-registered`, `redirect_uri`, `entity_id`, `did`. If omitted, the default value is `pre-registered`. Other values may be used when defined in the profiles of this specification.

8.2. 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 8 for the details.
- Verifier has pre-obtained static set of Wallet's metadata. See Section 10.1.2 for the example.

9. 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.

9.1. Additional Verifier Metadata Parameters

This specification defines the following new 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 A. Deployments can extend the formats supported, provided Issuers, Holders and Verifiers all understand the new format.

`client_id_scheme`: OPTIONAL. JSON String identifying the Client Identifier scheme. The value range defined by this specification is `pre-registered`, `redirect_uri`, `entity_id`, `did`. If omitted, the default value is `pre-registered`.

10. Implementation Considerations

10.1. Static Configuration Values of the Wallets

This document lists profiles that define static configuration values of the Wallets and defines one set of static configuration values that can be used by the Verifier when it is unable to perform Dynamic Discovery and is not using any of the profiles.

10.1.1. Profiles that Define Static Configuration Values

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

- JWT VC Presentation Profile

10.1.2. 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"
  ]
}
```

10.2. Support for Federations/Trust Schemes

Often Verifiers will want to request Verifiable Credentials from a Credential Issuer who is a participant of a federation, or adheres to a known trust scheme, rather than from a specific Credential Issuer, for example, a "BSc Chemistry Degree" Credential from the hypothetical "eduCreds" trust scheme rather than from a specifically named university.

To facilitate this, federations will need to determine how a Credential Issuer can indicate in a Verifiable Credential that they are a member of one or more federations/trust schemes. Once this is done, the Verifier will be able to create a `presentation_definition` that includes this filtering criteria. This will enable the Wallet to select all the Verifiable Credentials that match this criteria and then by some means (for example, by asking the user) determine which matching Verifiable Credential to return to the Verifier. Upon receiving this Verifiable Credential, the Verifier will be able to call its federation API to determine if the Credential Issuer is indeed a member of the federation/trust scheme that it says it is.

Indicating the federations/trust schemes used by a Credential Issuer MAY be achieved by defining a `termsOfUse` property [VC_DATA].

Note: [VC_DATA] describes terms of use as "can be utilized by a Credential Issuer ... to communicate the terms under which a Verifiable Credential ... was issued."

The following is a non-normative example of the terms of use that may be defined:

```
{
   "termsOfUse":[
      {
         "type":"<uri that identifies this type of terms of use>",
         "federations":[
            "<list of federations/trust schemes the Credential Issuer asserts it is a
member of>"
         ]
      }
   ]
}
```

Federations that conform to those specified in [OpenID.Federation] are identified by the `type` `urn:ietf:params:oauth:federation`. Individual federations are identified by the Entity Identifier of the trust anchor. If the federation decides to use trust marks as signs of whether an entity belongs to a federation

or not then the federation is identified by the `type` `urn:ietf:params:oauth:federation_trust_mark` and individual federations are identified by the Entity Identifier of the trust mark issuer.

Trust schemes that conform to the TRAIN [TRAIN] trust scheme are identified by the `type` `https://train.trust-scheme.de/info`. Individual federations are identified by their DNS names.

The following is a non-normative example of a `claims` parameter containing a `presentation_definition` that filters VCs based on their federation memberships:

```
{
    "vp_token": {
        "presentation_definition": {
            "id": "32f54163-7166-48f1",
            "input_descriptors": [
                {
                    "id": "federationExample",
                    "purpose": "To pick a UK university that is a member of the UK
academic federation",
                    "constraints": {
                        "fields": [
                            {
                                "path": [
                                    "$.termsOfUse.type"
                                ],
                                "filter": {
                                    "type": "string",
                                    "const": "https://train.trust-scheme.de/info"
                                }
                            },
                            {
                                "path": [
                                    "$.termsOfUse.federations"
                                ],
                                "filter": {
                                    "type": "string",
                                    "const": "ukuniversities.ac.uk"
                                }
                            }
                        ]
                    }
                }
            ]
        }
    }
}
```

This example will choose a Verifiable Credential that has been issued by a university that is a member of the `ukuniversities.ac.uk` federation and that uses the TRAIN terms of use specification for asserting federation memberships.

10.3. 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.

10.4. 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 [I-D.ietf-oauth-security-topics].

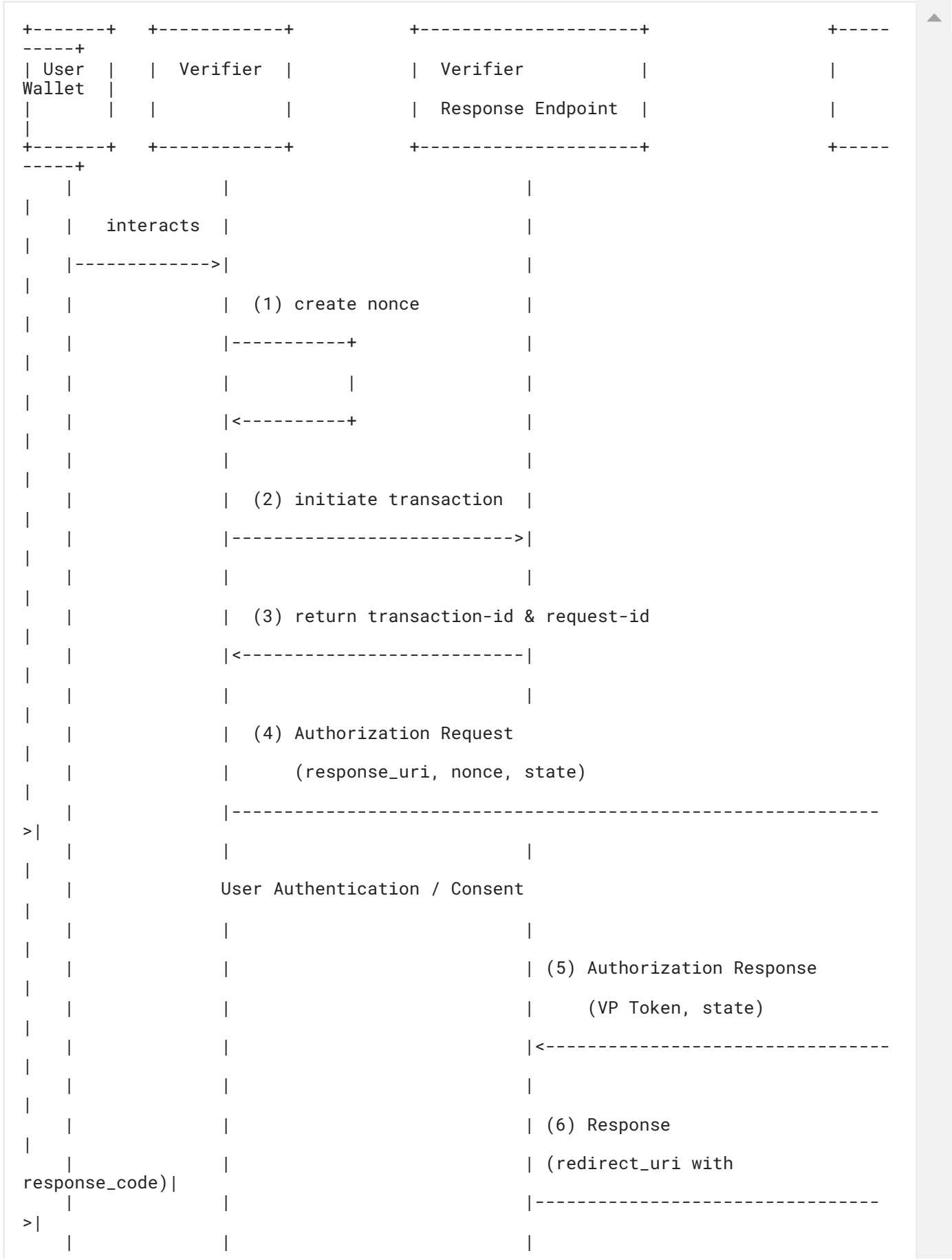
When using Response Mode `direct_post`, also see Section 11.3.

10.5. Response Mode `direct_post`

The design of the interactions between the different components of the Verifier (especially Frontend and Response Endpoint) 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 11.

The design is illustrated in the following sequence diagram:


User Wallet
Verifier
Verifier Response Endpoint
interacts
(1) create nonce
(2) initiate transaction
(3) return transaction-id & request-id
(4) Authorization Request
(response_uri, nonce, state)
User Authentication / Consent
(5) Authorization Response
(VP Token, state)
(6) Response
(redirect_uri with response_code)

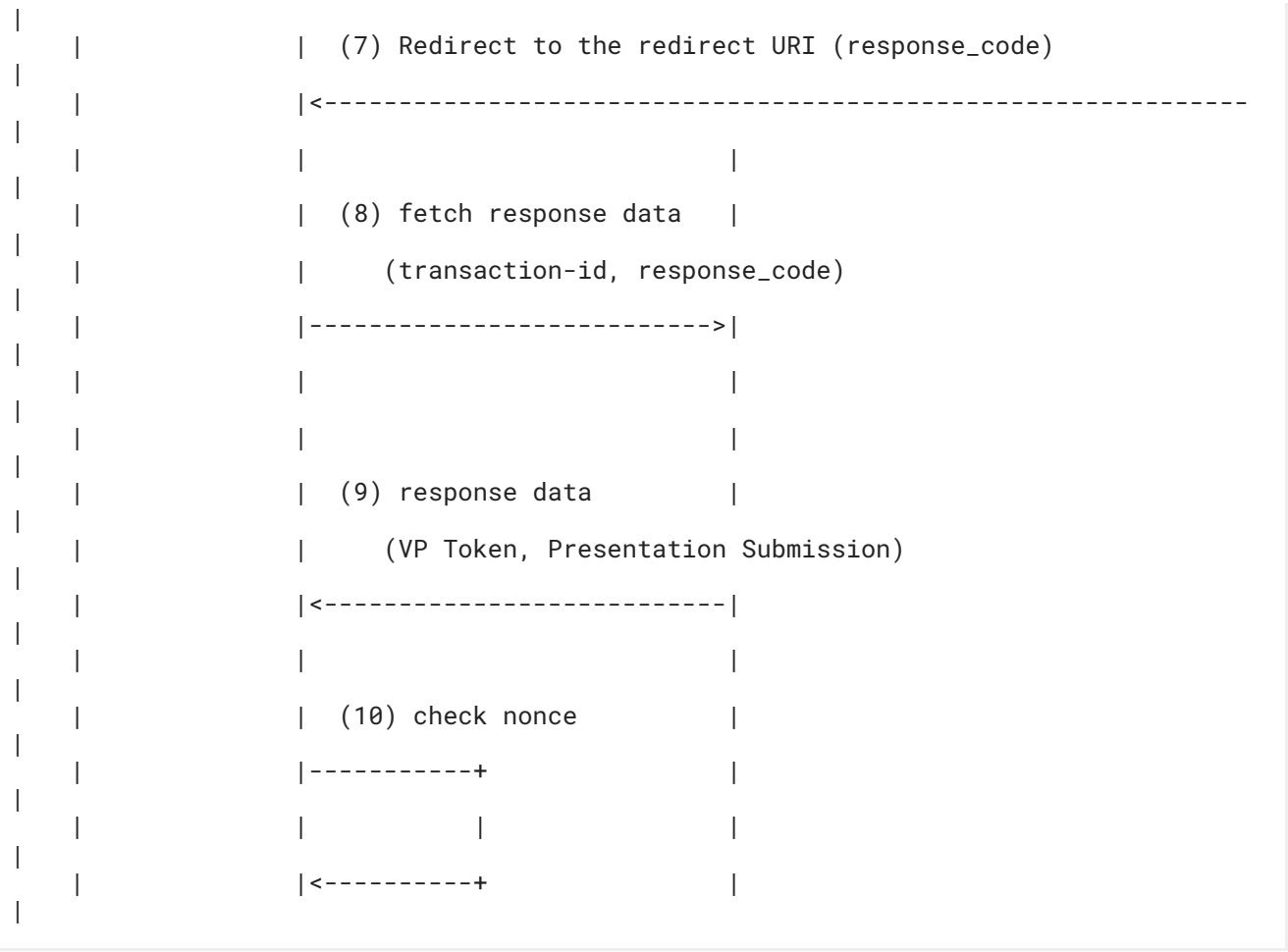


Figure 3: Reference Design for Response Mode `direct_post`

(1) The Verifier selects a `nonce` value as fresh, cryptographically random number with sufficient entropy and associates it with the session.

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

(3) The Response Endpoint 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 her consent to share the request Credentials, the Wallet sends the Authorization Response with the parameters `vp_token`, `presentation_submission` and `state` to the `response_uri` of the Verifier.

(6) The Verifier's Response Endpoint 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 Endpoint 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 Endpoint.

- The Response Endpoint 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 Endpoint checks whether the `response_code` was associated with this Authorization Response in step (6).

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

(9) The Response Endpoint 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.

11. Security Considerations

11.1. 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 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.

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. The format of the respective presentation is determined from the format information in the presentation submission in the Authorization Response.

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.

11.2. 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 Endpoint.

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 Endpoint MUST include a fresh secret (Response Code) into the redirect URI returned to the Wallet and the Verifier's Response Endpoint 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.

See Section 10.5 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].

11.3. Response Mode "direct_post"

11.3.1. 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 [I-D.ietf-oauth-security-topics]. 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.

11.3.2. 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.

11.3.3. Protection of the Authorization Response Data

This specification assumes that the Verifier's Response Endpoint 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 Endpoint 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.

11.4. 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 user.

11.5. 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 11.1.

11.6. DIF Presentation Exchange 2.0.0

11.6.1. 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.

11.6.2. 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].

11.6.3. 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.

12. Normative References

[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 2.0.0", <https://identity.foundation/presentation-exchange/spec/v2.0.0/>.

[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., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1", 8 November 2014, <http://openid.net/specs/openid-connect-core-1_0.html>.

[OpenID.Registration] Sakimura, N., Bradley, J., and M. B. Jones, "OpenID Connect Dynamic Client Registration 1.0 incorporating errata set 1", 8 November 2014, <https://openid.net/specs/openid-connect-registration-1_0.html>.

[OpenID.VCI] Lodderstedt, T., Yasuda, K., and T. Looker, "OpenID for Verifiable Credential Issuance", 3 February 2023, <https://openid.net/specs/openid-4-verifiable-credential-issuance-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>.

[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>.

[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>.

[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>.

[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>.

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

13. Informative References

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

[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-13, 15 April 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-jsonpath-base-13>.

[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-01, 13 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-cross-device-security-01>.

[I-D.ietf-oauth-security-topics] Lodderstedt, T., Bradley, J., Labunets, A., and D. Fett, "OAuth 2.0 Security Best Current Practice", Work in Progress, Internet-Draft, draft-ietf-oauth-security-topics-22, 13 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-security-topics-22>.

[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>.

[OpenID.Federation] Ed., R. H., Jones, M. B., Solberg, A., Gulliksson, S., and J. Bradley, "OpenID Connect Federation 1.0 - draft 28>", 24 March 2023, <https://openid.net/specs/openid-connect-federation-1_0-28.html>.

[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>.

[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>.

[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>.

[TRAIN] Jeyakumar, I. H. J., Chadwick, D. W., and M. Kubach, "A novel approach to establish trust in Verifiable Credential issuers in Self-Sovereign Identity ecosystems using TRAIN", 8 July 2022, <https://oid2022.compute.dtu.dk/index.html>.

[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. Examples with Credentials in Various Formats

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 format, not limited to the formats defined in [VC_DATA]. This section aims to illustrate this with examples utilizing different Credential formats. Customization of OpenID for Verifiable Presentation for Credential formats other than those defined in [VC_DATA] uses extension points of Presentation Exchange [DIF.PresentationExchange].

A.1. W3C Verifiable Credentials

A.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.

A.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"
            }
        }
    }
}
```

A.1.1.2. Presentation Request

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

```
  GET /authorize?
    response_type=vp_token
    &client_id=https%3A%2F%2Fclient.example.org%2Fcb
    &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
    &presentation_definition=...
    &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`.

A.1.1.3. Presentation Response

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": "https://client.example.org/cb",
    "nbf": 1541493724,
    "iat": 1541493724,
    "exp": 1573029723,
    "nonce": "n-0S6_WzA2Mj",
    "vp": {
        "@context": [
            "https://www.w3.org/2018/credentials/v1"
        ],
        "type": [
            "VerifiablePresentation"
        ],
        "verifiableCredential": [
            "eyJhb...ssw5c"
        ]
    }
}
```

Note: The VP's `nonce` claim contains the value of the `nonce` of the presentation request and the `aud` claim contains the Client Identifier of the Verifier. This allows the Verifier to detect replay of a Presentation as recommended in Section 11.1.

A.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.

A.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"
    }
}
```

A.1.2.2. Presentation Request

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

```
  GET /authorize?
    response_type=vp_token
    &client_id=https%3A%2F%2Fclient.example.org%2Fcb
    &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
    &presentation_definition=...
    &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`.

A.1.2.3. Presentation Response

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"
    }
}
```

Note: The VP's `challenge` claim contains the value of the `nonce` of the presentation request and the `domain` claims contains the Client Identifier of the Verifier. This allows the Verifier to detect replay of a presentation as recommended in Section 11.1.

A.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`.

A.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": {
        "first_name": {
            "raw": "Alice",
            "encoded": "6874ecdbdb214ee888e37c8c983e2f1c9c0ed16907b519704db42bb6"
        },
        "last_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":
"977423225617965826161030874586673609060253833336339637510512042715627326115520777573240007342290504514760916978895280468392292138385927475847984210013865986559197693721526403273427741674411349176661607661236811589163783458814384047777877615932551403490096873032745927956461585806847228270552979880833410883312450559437179134831763953399331039151162057199112357959170076753792711700533312522910797352842323445933004238048599164039686432144165884599052061538014126710866075791210006585893465085621395503182710866197817129408546193805893321161372355187962990595781339533851533077334790530438016817333603675910702146635975282253747819810788129751055728368937483121363992748831475139233180853145906108476753713239644943541916540123456371366974874702598201796929261151925643543132170495933035112012082080893049915977209167597"
        },
        "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.

A.2.2. Presentation Request

A.2.2.1. Request Example

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

```
  GET /authorize?
    response_type=vp_token
    &client_id=https%3A%2F%2Fclient.example.org%2Fcb
    &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
    &presentation_definition=...
    &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.

A.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 A.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.first_name"
                  ]
               },
               {
                  "path": [
                     "$.values.last_name"
                  ]
               }
            ]
         }
      }
   ]
}
```

This example is identic 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.

A.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_definition` 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 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 `presentation_definition` parameter:

```
{
    "proof": {...},
    "requested_proof": {
        "revealed_attrs": {},
        "revealed_attr_groups": {
            "id_card_credential": {
                "sub_proof_index": 0,
                "values": {
                    "last_name": {
                        "raw": "Wonderland",
                        "encoded": "167908493…94017654562035"
                    },
                    "first_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
        }
    ]
}
```

A.3. ISO mobile Driving License (mDL)

This section illustrates how a mobile driving license (mDL) Credential expressed using a data model and data sets defined in [ISO.18013-5] encoded as CBOR can be presented from the End-User's device directly to the Verifier using this specification.

The Credential format identifier is `mso_mdoc`.

Cipher suites should use signature suites names defined in [ISO.18013-5].

A.3.1. Presentation Request

A non-normative example of an Authorization Request would look the same as in the examples of other Credential formats in this Annex. The difference is in the content of the `presentation_definition` parameter.

```
  GET /authorize?
    response_type=vp_token
    &client_id=https%3A%2F%2Fclient.example.org%2Fcb
    &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
    &presentation_definition=...
    &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": "mDL-sample-req",
    "input_descriptors": [
        {
            "id": "mDL",
            "format": {
                "mso_mdoc": {
                    "alg": [
                        "EdDSA",
                        "ES256"
                    ]
                },
                "constraints": {
                    "limit_disclosure": "required",
                    "fields": [
                        {
                            "path": [
                                "$.mdoc.doctype"
                            ],
                            "filter": {
                                "type": "string",
                                "const": "org.iso.18013.5.1.mDL"
                            }
                        },
                        {
                            "path": [
                                "$.mdoc.namespace"
                            ],
                            "filter": {
                                "type": "string",
                                "const": "org.iso.18013.5.1"
                            }
                        },
                        {
                            "path": [
                                "$.mdoc.family_name"
                            ],
                            "intent_to_retain": "false"
                        },
                        {
                            "path": [
                                "$.mdoc.portrait"
                            ],
                            "intent_to_retain": "false"
                        },
                        {
                            "path": [
                                "$.mdoc.driving_privileges"
                            ],
                            "intent_to_retain": "false"
                        }
                    ]
                }
            }
        }
    ]
}
```

To start with, the `format` parameter in the `input_descriptor` element is set to `mso_mdoc`, i.e., it requests presentation of an mDL in CBOR format.

To request user claims in ISO/IEC 18013-5:2021 mDL, a `doctype` and `namespace` of the claim needs to be specified. Moreover, the Verifiers needs to indicate whether it intends to retain obtained user claims or not, using `intent_to_retain` property.

Note: `intent_to_retain` is a property introduced in this example to meet requirements of [ISO.18013-5].

Setting `limit_disclosure` property defined in [DIF.PresentationExchange] to `required` enables selective release by instructing the Wallet to submit only the data parameters specified in the fields array. Selective release of claims is a requirement built into an ISO/IEC 18013-5:2021 mDL data model.

A.3.2. 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 is a non-normative example of the content of the `presentation_definition` parameter:

```
{
    "definition_id": "mDL-sample-req",
    "id": "mDL-sample-res",
    "descriptor_map": [
        {
            "id": "mDL",
            "format": "mso_mdoc",
            "path": "$"
        }
    ]
}
```

The `descriptor_map` refers to the `input_descriptor` element with an identifier `mDL` and tells the Verifier that there is an ISO/IEC 18013-5:2021 mDL (`format` is `mso_mdoc`) in CBOR encoding directly in the `vp_token` (path is the root designated by $).

When ISO/IEC 18013-5:2021 mDL is expressed in CBOR the `path_nested` parameter cannot be used to point to the location of the requested claims. The user claims will always be included in the `issuerSigned` item. `path_nested` parameter can be used, however, when a JSON-encoded ISO/IEC 18013-5:2021 mDL is returned.

The following is a non-normative example of an ISO/IEC 18013-5:2021 mDL encoded as CBOR in diagnostic notation (line wraps within values are for display purposes only) as conveyed in the `vp_token` parameter.

```
{
    "status": 0,
    "version": "1.0",
    "documents": [
        {
            "docType": "org.iso.18013.5.1.mDL",
            "deviceSigned": {
                "deviceAuth": {
                    "deviceMac": [
                        << {1: 5} >>,
                        {},
                        null,
h'A574C64F18902BFE18B742F17C581218F88EA279AA96D0F5888123843461A3B6'
                    ]
                },
                "nameSpaces": 24(h'A0')
            },
            "issuerSigned": {
```

```
                "issuerAuth": [
                  << {1: -7} >>,
                    {
                        33:
h'30820215308201BCA003020102021404AD06A30C1A6DC6E93BE0E2E8F78DCAFA7907C2300A06082A8648C
E3D040302305B310B30090603550406130253A45312E302C060355040A0C25465053204D6F62696C69747920
616E64205472616E73706F7274206F66205A65746F706961311C301A06035504030C1349414341205A65746
573436F6E666964656E73301E170D323130393239303333034355A170D323231313033303333034345A30
50311A30180603550403C114453205A65746573436F6E666964656E7331253023060355040A0C1C5A65746
F706961204369747920446574206F662054726166666963310B300906035504061302534530130607
2A8648CE3D020106082A8648CE3D030107034200047C5545E9A0B15F4FF3CE5015121E8AD3257C28D541C1C
D0D604FC9D1E352CCC38ADEF5F7902D44B7A6FC1F99F06EEDF7B0018FD9DA716AEC2F1FFAC173356C7DA369
3067301F0603551D23041830168014BBA2A53201700D3C97542EF42889556D15B7AC4630150603551D25010
1FF040B3009060728818C5D050102301D0603551D0E04160414CE5FD758A8E88563E625CF056BFE9F692F42
96FD300E0603551D0F0101FF040403020780300A06082A8648CE3D040302034700304402201 2B06A3813FFE
C5679F3B8CDDB51EAA4B95B0CBB1786B09405E2000E9C46618C02202C1F778AD252285ED05D9B55469F1CB7
8D773671F30FE7AB815371942328317C'
                    },
                    <<
                      24(<<
                          {
                            "docType": "org.iso.18013.5.1.mDL",
                            "version": "1.0",
                            "validityInfo": {
                                "signed": 0("2022-04-15T06:23:56Z"),
                                "validFrom": 0("2022-04-15T06:23:56Z"),
                                "validUntil": 0("2027-01-02T00:00:00Z")
                            },
                            "valueDigests": {
                                "org.iso.18013.5.1": {
                                    1:
h'0F1571A97FFB799CC8FCDF2BA4FC29099290AAD37AE37ACE3C3BAE85C6379AD5',
                                    2:
h'0CDFE077400432C055A2B69596C90AAA47277C9678BFFC32BBC7F0CF82713B8E',
                                    3:
h'E2382149255AE8E955AF9B8984395315F3A38427C267F910A637D3FC81F25BB4',
                                    4:
h'BBC77E6CCA981A3AD0C3E544EDF8B68F19F4DACF1AF2AA0E6436401B4539ABA2',
                                    6:
h'BB6E6C68D1B4B4EC5A2AE9206F5F976A32061FA878BD5B44476F96D35462F6B2',
                                    8:
h'F8A5966E6DAC9970E0334D8F75E24DC63832E73A56AEF21C0D4B91487FC6AB03',
                                    9:
h'EAD5E8B5E543BD31F3BE57DE4ED6CCF7BB635221725F80538165DA7DC0BF92BB',
                                    11:
h'38CE9A09DC0121E1A9C2EF3EE2456530C2183AA8326FBE13B7D19A17DF77E980',
                                    12:
h'DEFDF1AA746718016EF1B94BFE5FB7774B8665F57D48ADAB83ABE0B28C22DB59',
                                    13:
h'A8868DF71AA4FB7D0AD3459C2E75E63767FE477B5A8FDF45537E936AFAB59C44',
                                    15:
h'95B651F1BA60EF5867E63E8DB1B0328464E5B66775E213B743A1E31F8EFBD9CB',
                                    16:
h'364E3C65D46D06FEDEB0E7293A86BA45FDFA99AA1A6DA3C3289B6E073B589922',
                                    17:
h'B584E5D5EF4CFC93FDB1E4EE8F3996090EF0B1E8FD2AC594D7D8793093BB328F',
                                    18:
h'677468F3E28CAAB521337E0FEF7FFEB067D2A2704F88B5D50D84CAF17209BA25',
                                    19:
h'95501E3E769230DC945CFBDC707C45218459F1129EFD5088BDA672CEF5991598',
                                    20:
h'677FACBBCA2EB9306BD649227B9AD66DF4A9AF6A5AB7D073F1BAAC23254B6D78',
                                    22:
h'BCCFB15CB36125BF1ECBFDE32FF908BD3BAA2DC0BA949B673E96CBA26902059F',
```

```
                                    23:
h'F9EE4D36F67DBD75E23311AC1C294C563992463A30B47039D25E03B6C6EFFCA3',
                                    25:
h'AFC5A127BE44753172844B13491D880C3768691A4C9916E5257CEFA4BAA74654',
                                    26:
h'1E1DA854356D3CFB7D983B8105F8A081057D4D01E910F263143BDC9AF1EF363C'
                                }
                            },
                            "deviceKeyInfo": {
                                "deviceKey": {
                                    1: 2,
                                    -1: 1,
                                    -2:
h'B820963964E53AF064686DD9218303494A5B23A175C34CC54AD1D9244EFD0BA5',
                                    -3:
h'0A6DA0AF437E2943F1836F31C678D89298E93D7E95057FD4D04E8E3EC2BBA935'
                                }
                            },
                            "digestAlgorithm": "SHA-256"
                        }
                    >>)
                >>,

h'1AD0D6A7313EFDC38FCD765852FA2BD43DEBF48BF5A516960A685162B2B8242935861329ECB54F68234FC
88A0228EC5DF22CB9689EC5053C80EDC59CC99EE80D'
            ],
            "nameSpaces": {
                "org.iso.18013.5.1": [
                    24(<<
                        {
                          "digestID": 1,
                          "random": h'E0B70BCEFBD43686F345C9ED429343AA',
                          "elementIdentifier": "expiry_date",
                          "elementValue": 1004("2030-02-22")
                        }
                    >>),
                    24(<<
                        {
                          "digestID": 6,
                          "random": h'AE84834F389EE69888665B90A3E4FCCE',
                          "elementIdentifier": "given_name",
                          "elementValue": "Doe"
                        }
                    >>),
                    24(<<
                        {
                          "digestID": 11,
                          "random": h'960CB15A2EA9B68E5233CE902807AA95',
                          "elementIdentifier": "issuing_country",
                          "elementValue": "UT"
                        }
                    >>),
                    24(<<
                        {
                          "digestID": 13,
                          "random": h'9D3774BD5994CCFED248674B32A4F76A',
                          "elementIdentifier": "document_number",
                          "elementValue": "ES24689"
                        }
                    >>),
                    24(<<
                        {
                          "digestID": 15,
                          "random": h'EB12193DC66C6174530CDC29B274381F',
```

```
                            "elementIdentifier": "portrait",
                            "elementValue":
h'89504E470D0A1A0A0000000D4948445200000001F3000001F20803000000C88DD7DC000000197445587453 6
F667477617265004164 6F626520496D6167652656164797 1C9653C00000180504C5445336D82D0B293417E
9A4848484684A24A8CAE306A7ED1C8D144809D357085407D993F7B96B3B3B0275D718687864765734886A54
A8BAC3E7A954A8AAB38738ACFA8CE4686A53C7993FEE6CF4787A74785A44584A33D7A94A18B765555554381
9E3A768F498AACE4D6E339748D4B8CADAC67AA71706F74878436718845565D4545454A5F68FEF0E3395965E
2C5A545748A2E5A69FDDCBCB0AA9737515A515151F1CAA33B77904C4C4C3E7B96939B986D929EB79C824783
A044494BD1BDA35B899BA0A5A2466D7D4C839AFDD2A84F879F424D524C8DAF515B5FFFFFFF467E9AA458A25
357594888A840778F42809C4989A94582A04583A1417F9B40748B4482A043819D4483A14888A7407C973C78
913A758D3E79934889AA3B7891427F9C42809D4889A9498AAA3A758E417E994988A843809D39748CBA82B9E
9E6E98E7C6CF3F3F3E6BD9E50555773685D5E5E5FFDD5AE5C5651504D4BF9F4F9F3EAF3FEFCF9675F57FEF8
F14170845C7A7E38748E6A4E69487F9B467B94925591816E80534A530E2CDF5E00001E3B4944415478DAECD
DEB6313C77600F031A212B13D1675E340C15EA7BAB588A41819635FD72832288A6AC55CA879A55052726FD3
5BD960630C01737BDDF65FAF64F9A1C7AE761E6766CE91E77C49F20556FBCB397366667796FD33C2F89BC8F
8C793F8DB88F887BF0B8D3FFDE9E79F7FFEE31FFFA533FED015FFF9877F6DC6DFF7C5E5767C751447FF7AAD
372E5E3F8D8B47F1E8347E3A8AA7EDD83C89E6BFCF76C7B3A3F8F3A54B977EF9E56167BC3A89C7EDB8D015B
F8E76C6F3EE18EB89C9C9C9E9664C4E324F7EDEC8319A7B72B3E4D3CC939F37727CE69EDC34393A734F6E9C
1C9BB927374F8ECCDC935B20C765EEC96D90A332F7E456C831997B723BE45798273F6FE42F99273F6FE468C
C3DB935722CE69EDC1E3912734F6E911C87B927B7498EA26FF7E456C931987B72BBE408CC3DB96572F7E69E
DC36B973734F6E9DDCB5B927B74FEED8DC933B20776BEEC95D903B35F7E44EC85D9A7B7237E40ECD3DB923F
217CC939F377267E69EDC19B92B734FEE8EDC91B9277748EEC6DC93BB247762EEC99D92BB30F7E46EC91D98
7B72C7E4F6CD3DB96B72EBE69EDC39F9BF334F7EDEC82D9B7B7204E476CD3D390672ABE69E1C05B94D734F8
E83FCF7CC939F37727BE69E1C0BB935734F8E86DC96B927C7436EC9DC932322B763EEC931915B31F7E4A8C8
6D987B725CE416CC3D393272F3E69E1C1BB971734F8E8EDCB4B927C7476ED8DC932324FF9A79F2F3466ED4D
C93A3243769EEC971921B34F7E448C9CD997B72ACE4C6CC3D395A7253E69E1C2FB921734F8E98DC8CB927C7
4C6EC4DC93A3263761EEC971931B30F7E4C8C9FF839D1FF2C362259DCE35A3963F895AEB3FD3E974B1788EC
8C1CD31926F36B173E57C4CD49AF8C54BE7801CDA1C1DF961A523AD05A29C4B1F1C0E37F9BFB12126FFB112
9FDD51F0C5E1258735C7447E98AEE5B5A2E93E9CE4A0E678C8B5C14FDD0F878F1CD21C0BF9771518F0E3D62
E5D1C3272407324E43FA6CB79E0A8158AC3440E678E83FCC742DE48D45A457E48C8C1CC51909B126F8FED07
43420E658E817CD3A4783BD97F1D067220730CE49572DE7C140EE993C39823203FACE5ED44AE489DFC776C3
8C8D3797BD154274D0E61EE9EDC5A929FAA5326FF2F3604E45646F21EF543BAE4FAE6EEC90B79175138A44A
AE6DEE9C7CB3967713E50A51725D73D7E4C58A2BF256817F40925CD3DC297931EDD0BB1D698AE47AE60EC98
B85721E419CA63A21722D7367E4DF556A792451AE9023D7317745FE5DBA9C4714056AE41AE6AEC82BA8C45B
F5FD392D72757347E4C55A1E5DD41E902257367743FE5D3A8F31CA4F2891AB9ABB213FACE5F378D1A9907FC
9089157F268A3894E865CCDDC09F977853CE2288F9121573277435ECBA38EDA181572157327E47F454EDE9C
B25121573077427E58CEA38F34117279734F1E194F68904B9BFBB17CC0904E835CD6DC930FACEE24C825CDD
DCCCB73792AF18402B99CB91BF23419F27C8E02B994B95F7D8B8DF931FCE432E66EC8FF5AA6645EE663E8C9
25CC1D6D9ED6F2A4E2361FC34E2E6EEE883C9D27169C8F2127173677447E488DBC99E8FC9F70938B9ABB7A1
02A47CEBC99E8FCF7A8C905CD9D3DFB468FBC95E827E838C9C5CC9D3DE15A23685EE627E848C985CC9D3DC7
9ECEE789267A0B1D2BB988B9BB5717CA24CD6BFC081D2DB980B9BB179468A6793EBF7084FE2556F2787377E
444D3FCB8B873FE0D52F2587387AF2156F25483F7A2A3228F3377F9B2718DACF9420F3A2EF2187397E445B2
E4C75DDC293A32F26F18DA23050A74CD4F8A3BE70C21F94073A7E49B65C2E6B74FD193F8C80799BB3D2BA64
298FCACB837D1D1910F30777C3C508EB2F95971EF4547401E6DEEFA4428D2E4A79D7B2F3A06F24873E78780
D136BFCD3BD1AFA0228F32777ED45F9AB6799977C5154CE411E6EE0FF4ACE587A7B89FA023210F37774FBE4
99CBCBBB8B7D1B190879A23388F5D6538DFDF456CCEAFA0210F33C7F0A10DC9E17C3F5B1AA956AB25B4037A
335E60210F3147F10525A9D9F976A97A1C25B4037A33FE8284BCDF1CC777D2C4175EF76796AB67B18DB7B8B
7D05190F799E320DF5413C7841E62CE7F8782BCD71CC90730455BB8ED1E7144E8651E868E81BCC71CCB676E
C53658DE8D544302CB98CE07A03B25EF3647F331EBB4629263425F0845FFC63D7997399EEF978B3C2F51AA4
6C5C83ED601BD195F3827EF34C7432E3055DB1FA946C79B8F78CDF917AEC93BCC1191C79B0F24AF56971174
72351E85EE98FCCC1C13F9E59A1E796B50DFC7D9C41DA3BB243F3547457E599BBC39A87FC4D9C4B5CBBB4BF
22F1846F23873017204F53DDA9C275D921F9B2323BFACDAB1F7A4FA3B944D5C27BA0BF2B63936F2C1E6DB55
D1709BEA83CC8FD19D901F99A3231F68BE5B950897A37A8DC7A1BB216F99E3231F64BEFFA62A15EE1AF8328
F417744DE344748FE15C0607E56E067D04DD68ED1BF7643FE05C348FE1550653F5996DB4669CEF9D74EC807
983B24FF0AACB23B555F1043B74D1E6DEE923CDA7CA6AA182ED4E3CD5BE8D6C923CD9D92479ABF5BAE56D5D
5F7314DD64ED0ED934799BB25FF0AAC81EBE9E6DEA1333F42B74B1E61EE98FC72C42390EFAABA51CA6233E7
7FB14D1E6EEE9AFC72CE449A1F97F812C832CDFEEECCCC4CCC68511332E75F5A260F35774E1E61FEAE0A132
3DB9A35FE63E96497A7F44EDFBC13DD067998B97BF208F352152C46D4B3FDDDCC1BC125FD329745B7421E62
8E803CDC7C7FB90A19CB2585747FB73D22F170359744B743DE6F8E81FC5A4E6F3F4D6270DF96C8F7DD52E8C
6FDB6BE791BDD12799F390AF270F391AA9918296DC7BFD1BABB5D8A2C33DBFAE62D745BE4BDE638C8AF853D
DFFEB16A32DE8C9466B2BB6149BF9B9D29C53C71F9517D21EEEC24396BE43DE648C843CD4B552BF166A415C
D8958A9F54FB1E5FD1100F34E74B3E4DDE658C843CDDF54F1C60C80F919BA61F22E7334E461E61FAB98E31D
80F909BA69F24E733CE461E625D4E62508F336BA71F20E7344E461E66FAAF4125DD6BC856E9EFCCC1C13F9B
5A2B17557AB23BAB4394F5A203F3547451E62BE8DDCFC0D8C7913DD38F989392EF210F31272F3EA4718739E
```

```
344E7E6C8E8C3CC47C19BB7909C8FC14DD1879DB1C1B79BFF92E76F2D0E2AE647E8C6E8EFCC81C1DF9B5436
AC37978E7AE667E846E90BC658E8FFCDA3572C379E84E8BA27913DD2479D31C23F93562B3F388015DD59CF3
2F0D927FC150925F24363B8FD8685137EF4407276728C9AF936BE19A016A7E860E4FCE50925F2F03BDBEE2B
689D3313F413740CE50925FCF916BE1AAD55D58F336BA09728692BCD77C8482F936B0790BDD08394349DE6B
4E813C649B45D39C7F69869CA124EF31FF48C2BCA4F30C64C4A16246C8194AF2EB057A6D7BC8648D1B42D72
46728C9AFA7A9ADBC1A320F45D7256728C97BCC494CD5FA27E8656E045D9B9CA124EF311FA1695EE326D0F5
C9194AF28B456F7EF6901C34394349DE63BE4CC37C57E9CC01397408728692BCC7BC7ABECD3BD041C8194AF
28B17BD79D7863A2839C3497E91DEF4BCDF7C8103A30391339CE48F6A04CDB7CD991FA14391339CE4773ACF
FEA0623E03BF24D3890E46CE5092AFB61E24DDA5B50CD76D9E2D2D571761D1C1C81956F28E4D8B1972E6BBE
DE7F7CCA32B913384E47797BB77AAC8999F3EE2F1D930BA1A39C347FE68BC677B929AF9D9533D37B9517445
72868FFC87DE56988A79A9FF41AE1B26D155C9193AF247F73AEED9F23B2A4FC39D6EA6765D2DEC88DE8DAE4
CCED0913FBAD97B1F472899F74C32B831747572868EFC51DF420725F3DEC7B83E9B42D72067E8C8EF74DFB4
37B4CC7BAF75959B41D72167D8C8BB86F376A21332EF6B3717B911742D72868DBC63A67692E854CC97438E2
11EE126D0F5C81936F2476B7D5B1754CCAB61330C6E025D8F9C6123EF371F21631EF6FA2C3782CE74C81936
F2EEA95ABB6652312FD932E79C6990336CE43F55872B3E5B478F2567D8C8BDB9267A3C39C346EECDF5D099A
AB943F26133AF70ABE84CD5DC25F94F3787CB9C739BE84CD5DC29F94F6BDE5C199DA99ABB257FEACD95D199
AAB963F2A7E34345BEC6EDA1335573D7E44FEF0D95F908B7862E2A9E64D8C89FAE0E95F9A269F3D3C57761F
25E73F7E44FEF0C95F92AB7842E4EDE638E80FCE90F7E4946015D82BCDB1C03F9D3CDA19AA057B8157419F2
2E731CE49BC334595BE656222943DE698E847C73981AF7118E093DD96B8E857CF38E6FE1CCA0277BCDD1906
F6EFA16CE087AB2D71C11F9300DE89CA3414FF69A63221FA2017D91A3414FF69AA3221FA201FD06C7829EEC
35C745FEF4E9D0CCD02B1C097AB2D71C1BF9D06CADAD718E033DD96B8E8E7C76588AFB04C7819EEC35C7473
E3B3B24C53DC951A0277BCD3192CF0E47E7BEC83906F464BF3942F2D9E1D85BFBCC31A027FBCD3192CFCE0E
431777937304E8C9507384E4B377FDE41C063D196A8E917C7676CDA739047A32D41C27F9104CD726B87BF46
484394A72FA89EE30CD4FD19311E648C9C98FE837B873F4A4883922F2D9678BFE01193DF4A488392AF2673F
905E8CAB70E7E822E6B8C89F3DA3DCC6AD728E179DA1257F46B8BAAF718E189DE125FFF325AAD57DB9C231A
333C4E4977E58A669FE9973CCE80C31F9A55FEE9044BFC1396A748699FC978714D1119187A333D4E40F1FDE
A136A62FA3220F4567B8C91F3EFC9E56F73E52E11C3B3A434EDE8CBBE3640AFCE2678E2F42CCB1933F2453E
12B1C67F49953207FF88A42815FE39C063A2341FE8AC23AEC2A2782CE4890BFFADEAFC4C0A13312E4AF5EE1
7F8462997322E88C06F92BFC27882D722AE88C06F92BFC8FCDDCE054D0190DF257AFD0CFD6929C0A3A2342F
E18FB6C6D8D732AE88C08F963ECB3B5094E069D11217FFC187971AF7032E88C0A39F2E2BEC639197446851C
79719FE09C8C3AA342FEF8C24D5FDA61D01919F20B8BBEB403BDED4086FCC25D5FDA61829121BF80B9B8274
999D321BF8077CD7D91533327428EB8B8DFA0664E85FCC205AC1BAACB9C98391D72B4C57D91983921F20B17
903E005BA16E8E98FC02CE23C4D6387173CCE448BBB81BC4CD5193FF3A8A718ABE9CA46D8E9C7C74D57770D
0E6D8C94747977D07076B8E9F7C14DF89CF239CB23901F2D1BBBE838334A7403E3A8A6DBA76931336A7413E
8AED719909C2E644C84747D7FC440DC89C0C39B2E9DA2A276B4E877CF4F94D3F518330A744FE1C53A22F72A
AE6A4C89F634AF40A557362E488129D649AB7CCA9913FFF7ED9A7B99E3939F2E7CFEFF934D73227488E26D1
2BA4CD49916319D189A6F9B13931F2E7777C9AEB9A5323C7614E35CD8FCCC991E330AF1036A747FE7CCCAFB
46B99532447604E7143EDC49C243902F3094EDC9C1AB97BF39B9CB83939F231E70F4E7C266E4E8FDCBD39A7
6D4E90DC9BEB995324F7E6FA6BAFD4C8BDB9B63939F2B17BDE5CCF9C1EB937D7342748EECDF5CC29927B732
D7392E4DE1CD49C04F9D81D6F0E674E83DC9B039A13219FF4E660E654C8BD39983919726F0E654E877CDA9B
C39813229FBEEBCD21CC29914F4F7B7300735AE493FED1287D7362E4AECD4786C09C1AB937D73627473EEDC
D35CDE9917B734D7382E4DE5CCF9C22F9F4B237D73027493EBDE6CDD5CD69927B731D739AE4DE1CC49C14B9
3787302745FEFD1DC76741AEDD4DD237A7447E2755AF5F756B7EB55E4FDD256E4E88FCE06DBD8EC1BCBE555
BA06C4E883C9DB8D732FFD6BD797DA27C9BAE391DF2835A22F11A8B7926912099EA8C14793A914864EB58CC
EB6F13098AA9CE08913F692679A25DDA71984FB4AEA746D09C0C79BADCBAC589141EF3F1A30B2A53ABEF8C0
AF9EF7389766CE1314F1D5FD26D8AE6F8C90F6AC7F73751C763BE75724DB45A3946833C7D7277136FDBE675
B7E69FDA17717A55E57962E6E8C94FEBFA69DBEE7851E656BDC79C547D6704C80FCA897E73A7C5FDDBE38BC
8765C189DFACEF093A7138910F3FA2D77E47BF510733AA9CEB0933FC925C2CD3FB94FF31E732A5375869CBC
524E4498BB1BD16FD523CC894CD5196AF207854422D2FC3747D57DEFB748731AF59D61263F9B948799D73FE
DB9ADEC61E6145A3986983C5D4E0C3477D3BB7790B73659FA027F7D6768C91FE41289187317E8573BFFFED0
2B445FDF1956F24A39116F6E1FBD8B3CC21C7BAA339CE411497EB6DEEE08BD9BBC1E7991B7919B23240F6BD
EC2CDEDA2F79067A2AF12732BC7109287CCD03AE2752FFA9E93F6AD732F9558AA3384599E1B742B8F9F99A8
DB9FB2EDF5920F36C78BCE10B66F92E69616676E7DEAFB8BEF0DBCD01A6A7364F3F2817732315EEF0F0BCBB
0B77EAB0F9339B635F6C1E6F742CC8D0FEA7B57C3FED62C5973743B6983CD27C2EEBEE1FA1E52D7E3CD1378
CD91EF97C72CCA74D4F73D5B53B4F8E939727364E44FD2EB4B31B73202A0FEE996D524AFD75FC75C68909B5
F406C8E635E5E29DC0F8220CEFC7514BA91517DEF6AE45F978AB9D0A5E68FD9C92D203547405E49AF07EDD8
8FB995A94884FA6F57AD25F9E92B0D83CD8FD891A53B4340FEE0205D08CE6225213F593B4BF55B6697613A6
355CC1C9D3B734CFEE02CBF45CD57EB0303AEC00F28EB226D7B62BFFB77A171670EC99BE97D3FE88F38F3EC
608966818751BFFA5BCC5F14C899A381676EC80F2A85F520221271ED703D2E3E5D353B908BB5ED8995A85F9
89B770ACFAC930FE216323F7E4BD1A8FAAD6FE3FF8E94B2F971C2BB826716C95F16D331DC62E6A97ADDACBA
8878DC6A7B9CF971C6BB28F5CC0E79B35513E10E04A6E7112BEE70EA62E2C7270E689A3B8167C6C98B95F05
64DDD7CA25E37A77EF593E89F9E8DBD50895F7DD4DC2DD8343743FEB258114D6E29F3B775E190ECE1F7C4C5
6357DB25CD2D0EF2CC0CF95831ADC02DB40C3770F5556B95662F767626D5C22998DBA9F5CC00F958A510A88
78079AA2E159F84925D7418175D79ED5E8853C87893E6C0E463E9402B5612504D5C47898F5B9C934B71B116
```

```
4ECBBCC5BE60CC1C98BCB81E18379FA8CBC720F5BD6F15FEC0B7A6CD9B45DEDC7A3B24B966928B990775958
8ACF0F2392EB20A07601EEC983207241F2B68930702F75264254EF8A18ABD4F4A7F98400B17BAE08EA0BE33
50F2F5C08E79AAAE1657C59E67055985033137 82CEB0910B99AFD6A1D0AFAAFE495981CB5C0950A2336CE44
B22E6D93A10BA6A96C76FA442991B4067C8C8C5CC13CAE6DDC74DED29936712B6CCE11B3906D6B1C3900B9A
A794CD3F0D7AE910744506CA1C1C9DE199A4092FC3A9ACCA8456F75BEA7FCA84D06502DD939C31732DF24A6
0D55C7940FF90C9BC398EE5EA2775F3B736CD837943E67AAB6F50BF4E6449467E40DFCA4C4D351A731B7DD1
684C4D653EC8936F25AC9A070B46CCF5D6D8EFDB36171ED03F34B537E2A2319531309C039AEF9830D7DB492
B04B6CDC74575E2C1DB3137B5053E9CEB2FBE1A19D21904F9011C7920763385B759DE6F88C7FB0F5BB0C339
A039E490CE20F6CBEFDB3717DD66C948983736E63E800EE790E63BC0E69A4FC500567661F34406B6B49FB0B
F871CCE21CD01AB3BD32787ACEC824B32C203FAD4866434208773 5073B8DE9D699303ADB9CA9A4F4057F6E3
98021CCE2136D60C5477A6FDB8633A70622E3243FF30276FBEB105379CC39A83B5714C97FC41E0C83C053E9
81F4FDADE830DE7C0E63BA0E61ACFB11760CDF785CDEFC10FE6A2D55D743887DA64016EE39826793170659E
3552D95B113B25081C9903B5714CF36D15E034175D861379B3A1A1481E5BDD330957E63928731D72E834973
14F19A9EC02D57DDC99394CA233BD77D2D61D9AAF9AA9ECF1D53DEBCE1C24D19916792570689E3553D9E357
66C42F3111604C74A6F5E629789A0712F773E053EE990DAD98D27CB2DD98790EC65C9D1C3ECDA5CC072DBFC
EE9990FDA6C5975690E91E84CE74881825BF309230D5C6CA2679D9A03243AD3202F066ECD03330D5C4C1BB7
95706A0E90E84CE3E0100369BE247343A3F753B5D37C401B37EED85C3FD1993AB989349733BF67A6816B474
A77E1157C33F53860CDE58E072A3837CF9A98A7C5ADC605AECDE721CDE5C81F04CECD23965F41D23CAA8DCB
245C9BEF009A4B1E02963661BE2F673E6E629E3638D1EF3937D74E74A67CEEDB7D04E6A1B3B5D4C686C144C
FBA37DF81329725AF0408CCC3666BEFE7A0CC373EA81D2962DA5C77BAC6540FF42C60300F9BAD4D8191CFA5
34676AA6CC7320E6D2E445233F46668B2562B60696E68DD47BDD999A29F300C25CFED8DE020EF3ACA1346FA
43E00CCD48C99CFEB9BCB93BFBC8FC3BC7F6F6D4A7B723EE885C5540285F98EB6B9C2E1DC663A3805F3B0D9
5A66AAA158E0E7E25E505D95BCBC7D43F76941D35CE53CF60216F3C8BDB54C2AFC8DF3E899594AE075A8B74
8CC737AE64A47F00758CCE3DE55DCCA6404ABBDC8EB895B0924E65A5D1C53FAD0461A8DB9C8AB0D1F84CA3A
E4BB89E6CDE721CCA5BEADB26EEAA748930B1D0F38A5F79084F24CCDA0F90E80B9147931C0632EF2D1868CC
6D6A9C6229C49739D2E8EA97C41A980C85CE845F406CC703E8EC87C5ED75CF2A359F731998BBC88BE05F3DE
F90422F31D4D7349F2830093B9D051715320C3B9FCE52D05088B3B53F81A620195B9D861EE71E82223442A8
1C93CA7632E4BFE32C0652E74CA4806C05C6E116E65DF20B8567167F2DF3CAD203307396544A46D7F8B855B
B38B63F29FB92D203317FADCDA94FE789EC1C3AD57DC9934B9C9D2AE669E02306F00CCD45696AC796B15772
6FD316B93A53D58DA5F3153DCA7F4975EB331E081ED98D73197FA7E79C1F86F596ACAAF74DAAF74FFA7CA99
90B1AB325AFB2B2B818BC869984B913F70F2F396564C1F2195D12AED4ED077D4CDA5C8CD96762571A17D968
6F67AFB84E52325CC157726496EA1B4CB8A0BEDB368EFABC5EFAFAC5029EE4C927CEC3E3A71A17D16ED0577
91FD15DB7DDC0E84793CB9E5D22EF8FE5A6C717FAFFDCC84D8FE8A65F5057DF378F2B1B4CD9FB402B6CF22B
0830EB5BFB28FBEB83339F2B1755C455D749F45C03C03B6BF622FD97774CD45C88B18C105F65952BA0FCA48
6E9DAF602EEE4C8ADCD2702EBF1697D5DD3F8F6BDC516D9D6BCED69814B985D2BEB4AFB4E6BEA56DDE00DE3
AB7C29ED33117237F80ACA40B17F786E6C3CEABAAD7657A6CD7301723375DDA95C563F759445E6B807C7FC5
5A1F3FAF6C2E486E7A116E49FDCEC66CA28B9867204EEFB65DE073AAE6A2E4A617E174CC0717F739BD27DC5
7352E0CDF6C8DC9901B9FA9254C1577CDE33EDFA2355799AD310972F38B702BA68ABBDE01DE3AA5DDF44C7D
5EC55C9CDCFC9E9AA9E29ED17B4B7115ED70AE54DC9904B985C7250C1577B1530229967695D91A1327B7B10
8973053DCC5CC33044BBB4A7167E2E436F6D456CC1477B183855260AF26E25E8A63E2E49316F6D496CC1477
31F3298043FAED9BEFE898C7913FB1B169A053DC75CD1B40078A581DCE15666B4C987CD2CA9E9A4E714F697
E84690EBEB4DBD8539D57368F259FB4F2F4A399E22E78661CC5D2AE30A03361F2693B8FC8E8D451DDAF3065
289676F9019D09935B19CEF506F494CE325C54E38EBDB4CB0FE84C947CDAD213AFFB268ABBBD03DB1D3C2B2
33DA03351F2694B2F332C1928EE5B82E60D8AA55D7E4067A2E4D3B65E663050DC453FD03247B2B44B2FBF32
51724BC3B9DE6C6D42F3DB5B244BBBF480CE04C927ADBDC062A0B80B9FE99EA158DAA5077426483E69EFDD4
4F8E22E6C9E2259DA65077426483E69EF0516F8E22E6C3E45B2B4CB0EE84C90DCDA706EA2B80B7FBAA141B2
B4CB0EE84C8C7CDAE6FBA8E0C55DD87C8E6469971DD09918F9B4CDA306C0D7DCC53FD142B3B44B0EE84C8C7
C7A3D2092E8AFD597E1FA1B7722A55D72406762E4764F0E827E5A46F903D8444ABBE480CE84C8A7ED1E2F01
5CDC3F889B4FC13DFC68F5C4897915F3C1E4D3568F97802EEE12DFC66E803DFC68B5B4CB0DE84C887CDAF26
951B0C55DC27C0EECB9F67DAB376C47DE3C8E7CDA2E39707197F99CE67B32CFB5AB0FE84C88FCC0F22F802D
EE32E61932CFB5AB0FE84C84DCF6700EFC12938CF9149157967406742 6427EC5FEE18F90C5BDA1684EA8B44
B0DE84C84FCE5BA7573C8375465CC1B344BBBD480CE44C89F58FF05A0C55DC67C8EC241039A033A13207FE9
E22C678D1B9E555E86EB6ADCDF524A7399019D0990BF4C07B4127D4BC33CA3773C989B0E4EC73C9CFC6521A
095E8E392E7FB86367113A44ABBCC360B13207FE9E227E8747159D565B84E738DFFE9F65DDCAF0525F328F2
0327E660C55DCEBC0150DA9DDCAF7915F328F22B1527BF01ACB8A7A4CCE7F44B3BF62FF2B078F22BE9805AA
2BF555D863B6BDC03521D9CD4AA0C8B27BFB21E904BF48CBA798666699768E2583CF91547BF41E775C55575
F329DDD2BEE4E8762D489A0F223F70651EC014F78682F96B7A692EDEC4B1587257C3B9DE742DA36CDED07C1
06EDFD5DDCA49990F24BF52080826FAAAF421135D8DFB04BD34176FE2582CB9B3164E2BD103C5A5D776E3FE
9ADA444D6A4067B1E457DCFD089DE95A4AD93CA355DA1DDEAD7961F338F20387BF42A3B84FA82DC3B59BB82
CC53417378F2377D8C269257AA061BE456FA226D3C4B13872972D9C56A2A754CD1B1AA5DD659A0B37712C8E
DCB1F99276719F92359FD328ED4B4E6F966013C7E2C85F0401D1447FAD68BE9122D9C1890FE82C8EFCC0B1B
9FA02ECB8AAF9FF124D7309F381E42F2A01D5449F505A866BC6FF114D73D1268EC590BF480764137D4BCDFC
7FA8A6B96813C762C85FAC0764137D5C65E97563E3BFA9A6B96813F7FF020C008EA9ADF1AC1529080000000
049454E44AE426082'
```

```
                        }
                    >>)),
                    24(<<
                        {
                          "digestID": 19,
                          "random": h'DB143143538F3C8D41DC024F9CB25C9D',
                          "elementIdentifier": "birth_date",
                          "elementValue": 1004("1980-02-05")
                        }
                    >>),
                    24(<<
                        {
                          "digestID": 20,
                          "random": h'6059FF1CE27B4997B4ADE1DE7B01DC60',
                          "elementIdentifier": "family_name",
                          "elementValue": "John"
                        }
                    >>),
                    24(<<
                        {
                          "digestID": 22,
                          "random": h'1E69C89C81B21A1BA56ACA3E026A2A3F',
                          "elementIdentifier": "issue_date",
                          "elementValue": 1004("2020-02-23")
                        }
                    >>),
                    24(<<
                        {
                          "digestID": 25,
                          "random": h'CAD1F6A38F603451F1FA653F81FF309D',
                          "elementIdentifier": "driving_privileges",
                          "elementValue": [
                              {
                                  "issue_date": 1004("2018-08-09"),
                                  "expiry_date": 1004("2024-10-20"),
                                  "vehicle_category_code": "A"
                              },
                              {
                                  "issue_date": 1004("2017-02-20"),
                                  "expiry_date": 1004("2024-10-20"),
                                  "vehicle_category_code": "B"
                              }
                          ]
                        }
                    >>),
                    24(<<
                        {
                          "digestID": 26,
                          "random": h'53C15C57B3B076E788795829190220B4',
                          "elementIdentifier": "issuing_authority",
                          "elementValue": "UTOPIA"
                        }
                    >>)
                ]
            }
        }
    ]
}
```

In the `deviceSigned` item, the `deviceAuth` item includes a signature by the deviceKey that belongs to the End-User. It is used to prove legitimate possession of the Credential, since the Issuer has signed over the deviceKey during the issuance of the Credential.

Note: The deviceKey does not have to be HW-bound.

In the `issueSigned` item, `issuerAuth` item includes Issuer's signature over the hashes of the user claims, and `namespaces` items include user claims within each namespace that the End-User agreed to reveal to the Verifier in that transaction.

Note: The user claims in the `deviceSigned` item correspond to self-attested claims inside a Self-Issued ID Token [SIOPv2] (none in the example below), and user claims in the `issuerSigned` item correspond to the user claims included in a VP Token signed by a trusted third party.

Note: The reason hashes of the user claims are included in the `issuerAuth` item lies in the selective release mechanism. Selective release of the user claims in an ISO/IEC 18013-5:2021 mDL is performed by the Issuer signing over the hashes of all the user claims during the issuance, and only the actual values of the claims that the End-User has agreed to reveal to the Verifier being included during the presentation.

The example in this section is also applicable to the electronic identification Verifiable Credentials expressed using data models defined in ISO/IEC TR 23220-2.

A.4. 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.

A.4.1. Request

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

```
GET /authorize?
  response_type=id_token
  &scope=openid
  &id_token_type=subject_signed
  &client_id=https%3A%2F%2Fclient.example.org%2Fcb
  &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 `id_token`. If the request also includes a `presentation_definition` parameter, the Wallet is supposed to return the `presentation_submission` and `vp_token` parameters in the same response as the `id_token` parameter.
- 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.

A.4.2. Response

The following is a non-normative example of a response sent upon receiving a request provided in Appendix A.4.1:

```
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": "https://client.example.org/cb",
  "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 B. IANA Considerations

- Response Type Name: `vp_token`
- Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
- Specification Document(s): https://openid.net/specs/openid-4-verifiable-presentations-1_0.html
- Response Type Name: `vp_token id_token`
- Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
- Specification Document(s): https://openid.net/specs/openid-4-verifiable-presentations-1_0.html

Note: Plan to register the following Response Types in the OAuth Authorization Endpoint Response Types IANA Registry.

Appendix C. Acknowledgements

We would like to thank John Bradley, Brian Campbell, David Chadwick, Giuseppe De Marco, Daniel Fett, George Fletcher, Fabian Hauck, Joseph Heenan, Alen Horvat, Andrew Hughes, Edmund Jay, Michael B. Jones, Gaurav Khot, Ronald Koenig, Mark Haine, Adam Lemmon, Kenichi Nakamura, Nat Sakimura, Arjen van Veen, and Jacob Ward for their valuable feedback and contributions that helped to evolve this specification.

Appendix D. Notices

developing specifications, and (ii) implementing Implementers Drafts and Final Specifications based on such documents, provided that attribution be made to the OIDF as the source of the material, but that such attribution does not indicate an endorsement by the OIDF.

The technology described in this specification was made available from contributions from various sources, including members of the OpenID Foundation and others. Although the OpenID Foundation has taken steps to help ensure that the technology is available for distribution, it takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this specification or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any independent effort to identify any such rights. The OpenID Foundation and the contributors to this specification make no (and hereby expressly disclaim any) warranties (express, implied, or otherwise), including implied warranties of merchantability, non-infringement, fitness for a particular purpose, or title, related to this specification, and the entire risk as to implementing this specification is assumed by the implementer. The OpenID Intellectual Property Rights policy requires contributors to offer a patent promise not to assert certain patent claims against other contributors and against implementers. The OpenID Foundation invites any interested party to bring to its attention any copyrights, patents, patent applications, or other proprietary rights that may cover technology that may be required to practice this specification.

Appendix E. Document History

[[To be removed from the final specification]]

-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
Spruce Systems, Inc.
Email: oliver.terbu@spruceid.com

Torsten Lodderstedt
yes.com
Email: torsten@lodderstedt.net

Kristina Yasuda
Microsoft
Email: kristina.yasuda@microsoft.com

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