Workgroup: OpenID Digital Credentials Protocols
Published: 16 September 2025
Status: Final
Authors: T. Lodderstedt *SPRIND*, K. Yasuda *SPRIND*, T. Looker *Mattr*, P. Bastian *Bundesdruckerei*

OpenID for Verifiable Credential Issuance 1.0

Abstract

This specification defines an API for the issuance of Verifiable Credentials.

Table of Contents

1. Introduction

This specification defines an OAuth-protected API for the issuance of Verifiable Credentials. Credentials can be of any format, including, but not limited to, IETF SD-JWT VC [I-D.ietf-oauth-sd-jwt-vc], ISO mdoc [ISO.18013-5], and W3C VCDM [VC_DATA].

Verifiable Credentials are very similar to identity assertions, like ID Tokens in OpenID Connect [OpenID.Core], in that they allow a Credential Issuer to assert End-User claims. A Verifiable Credential follows a pre-defined schema (the Credential type) and MAY be bound to a certain Holder, e.g., through Cryptographic Key Binding. Verifiable Credentials can be securely presented for the End-User to the RP, without the involvement of the Credential Issuer.

Access to this API is authorized using OAuth 2.0 [RFC6749], i.e., the Wallet uses OAuth 2.0 to obtain authorization to receive Verifiable Credentials. This way the issuance process can benefit from the proven security, simplicity, and flexibility of OAuth 2.0 and existing OAuth 2.0 deployments and OpenID Connect OPs (see [OpenID.Core]) can be extended to become Credential Issuers.

1.1. Errata revisions

The latest revision of this specification, incorporating any errata updates, is published at openid-4-verifiable-credential-issuance-1_0. The text of the final specification as approved will always be available at openid-4-verifiable-credential-issuance-1_0-final. When referring to this specification from other documents, it is recommended to reference openid-4-verifiable-credential-issuance-1_0.

1.2. Requirements Notation and Conventions

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

2. Terminology

This specification uses the terms "Access Token", "Authorization Endpoint", "Authorization Request", "Authorization Response", "Authorization Code Grant", "Authorization Server", "Client", "Client Authentication", "Client Identifier", "Grant Type", "Refresh Token", "Token Endpoint", "Token Request" and "Token Response" defined by OAuth 2.0 [RFC6749], the terms "End-User", "Entity", and "Request Object" 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" defined by JSON Web Signature (JWS) [RFC7515].

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

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

Credential Dataset: A set of one or more claims about a subject, provided by a Credential Issuer.

Credential (or Verifiable Credential (VC)): An instance of a Credential Configuration with a particular Credential Dataset, that is signed by an Issuer and can be cryptographically verified. An Issuer may provide multiple Credentials as separate instances of the same Credential Configuration and Credential Dataset but with different cryptographic values. In this specification, the term "Verifiable Credential" is also referred to as "Credential". It's important to note that the use of the term "Credential" here differs from its usage in [OpenID.Core] and [RFC6749]. In this context, "Credential" specifically does not encompass other meanings such as passwords used for login credentials.

Credential Format: Data Model used to create and represent Credential information. This format defines how various pieces of data within a Verifiable Credential are organized and encoded, ensuring that the Verifiable Credential can be consistently understood, processed, and verified by different systems. The exact parameters required to use a Credential Format in the context of this specification are defined in the Credential Format Profile. Definitions of Credential Formats is out of scope for this specification. Examples for Credential Formats are IETF SD-JWT VC [I-D.ietf-oauth-sd-jwt-vc], ISO mdoc [ISO.18013-5], and W3C VCDM [VC_DATA].

Credential Format Profile: Set of parameters specific to individual Credential Formats. This specification provides Credential Format Profiles for IETF SD-JWT VC [I-D.ietf-oauth-sd-jwt-vc], ISO mdoc [ISO.18013-5], and W3C VCDM [VC_DATA], which can be found in the section Appendix A. Additionally, other specifications or deployments can define their own Credential Format Profiles by utilizing the extension points defined in this specification.

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

Credential Configuration: Credential Issuer's description of a particular kind of Credential that the Credential Issuer is offering to issue, along with metadata pertaining to the issuance process and the issued Credentials. Each Credential Configuration references a Credential Format and specifies the corresponding parameters given in the Credential Format Profile. It also includes information about how issuance of a described Credential be requested, information on cryptographic methods and algorithms supported for issuance, and display information to be used by the Wallet. A Credential Configuration is identified by a Credential Configuration Identifier string that is unique to an Issuer. Credential Issuer metadata includes one or more Credential Configurations.

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. It can be of any Credential Format.

Credential Issuer (or Issuer): An entity that issues Verifiable Credentials. In the context of this specification, the Credential Issuer acts as an OAuth 2.0 Resource Server (see [RFC6749]). The Credential Issuer might also act as an Authorization Server.

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

Verifier: An entity that requests, receives, and validates Presentations.

Issuer-Holder-Verifier Model: Model that facilitates the exchange of claims, where claims are issued as Verifiable Credentials independently of the process of presenting them to Verifiers in the form of Presentations. An issued Verifiable Credential may be used multiple times, although this is not a requirement.

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

Cryptographic Holder Binding or Cryptographic Key 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. The concrete mechanism depends on the Credential Format. For example, in IETF SD-JWT VC [I-D.ietf-oauth-sd-jwt-vc], the Issuer can enable this binding by including a public key or a reference to a public key that matches to the private key controlled by the Holder.

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

Wallet: An entity used by the Holder to request, receive, store, present, and manage Verifiable Credentials and cryptographic key material. There is no single deployment model of a Wallet: Credentials and keys can be stored and managed either locally, through a remote self-hosted service, or via a remote third-party service. In the context of this specification, the Wallet acts as an OAuth 2.0 Client (see [RFC6749]) and obtains an Access Token to access an OAuth 2.0 Resource Server (Credential Endpoint).

Deferred Credential Issuance: Issuance of Credentials not directly in the response to a Credential issuance request but following a period of time that can be used to perform certain offline business processes.

3. Overview

3.1. Credential Issuer

This specification defines an API for Credential issuance provided by a Credential Issuer. The API is comprised of the following endpoints:

- A mandatory Credential Endpoint from which Credentials can be issued (see Section 8). From this endpoint, one Credential, or multiple Credentials with the same Credential Format and Credential Dataset can be issued in one request.
- An optional Nonce Endpoint from which a fresh `c_nonce` value can be obtained to be used in proof of possession of key material in a subsequent request to the Credential Endpoint (see Section 7).
- An optional Deferred Credential Endpoint to allow for the deferred delivery of Credentials (see Section 9).
- An optional mechanism for the Credential Issuer to make a Credential Offer to the Wallet to encourage the Wallet to start the issuance flow (see Section 4).

- An optional mechanism for the Credential Issuer to receive notification(s) from the Wallet about the status of the Credential(s) that have been issued (see Section 11).
- A mechanism for the Credential Issuer to publish metadata about the Credentials it is capable of issuing (see Section 12.2).

The Credential Endpoint may bind an issued Credential to specific cryptographic key material, in which case Credential requests include proof(s) of possession or attestations for the key material. Multiple key proof types are supported.

3.2. OAuth 2.0

According to the OAuth 2.0 framework, each Credential Issuer acts as a Resource Server that is protected by an Access Token issued by an Authorization Server, as defined in OAuth 2.0 [RFC6749]. The same Authorization Server can protect one or more Credential Issuers. Wallets identify the Authorization Server for a Credential Issuer by referring to the Credential Issuer's metadata (see Section 12.2).

All OAuth 2.0 Grant Types and extensions mechanisms can be used in conjunction with the Credential issuance API. Aspects not defined in this specification are expected to follow [RFC6749].

Existing OAuth 2.0 mechanisms are extended as follows:

- A new Grant Type "Pre-Authorized Code" is defined to facilitate flows where the preparation of the Credential issuance is conducted before the actual OAuth flow starts (see Section 3.5).
- A new authorization details [RFC9396] type `openid_credential` is defined to convey the details about the Credentials (including Credential Dataset, Credential Formats, and Credential types) the Wallet wants to obtain (see Section 5.1.1).
- Client metadata is used to convey the Wallet's metadata. The new Client metadata parameter `credential_offer_endpoint` is added to allow a Wallet (acting as OAuth 2.0 client) to publish its Credential Offer Endpoint (see Section 12.1).
- Authorization Endpoint: The additional parameter `issuer_state` is added to convey state in the context of processing an issuer-initiated Credential Offer (see Section 5.1).

3.3. Core Concepts

In the context of this specification, Credential Datasets, Credential Format and Credential Format Profile are defined in Section 2. While in principle independent of each other, the Credential Dataset and the Credential Format can have a relationship in the sense that an Issuer may only offer certain Credential Formats for certain Credential Datasets.

An End-User typically authorizes the issuance of Credentials with a specific Credential Dataset, but does not usually care about the Credential Format. The same Credential Dataset may even be issued in different Credential Formats or with multiple Credential instances.

3.3.1. Credential Formats and Credential Format Profiles

This specification is Credential Format agnostic and allows implementers to leverage specific capabilities of Credential Formats of their choice. To this end, extension points to add Credential Format specific parameters in the Credential Issuer metadata, Credential Offer, Authorization Request, and Credential Request are defined.

Credential Format Profiles for IETF SD-JWT VC [I-D.ietf-oauth-sd-jwt-vc], ISO mdoc [ISO.18013-5], and W3C VCDM [VC_DATA] are specified in Appendix A. Other specifications or deployments can define their own Credential Format Profiles using the above-mentioned extension points.

3.3.2. Batch Credential Issuance

This specification enables the issuance of Verifiable Credentials through the Credential Endpoint. A single request to this endpoint may request the issuance of a batch of one or more Verifiable Credentials.

Credentials can vary in their format, including Credential Format Profile-specific parameters, in their contents known as the Credential Dataset, and in the cryptographic data such as Issuer signatures, hashes, and keys used for Cryptographic Key Binding. Credentials can therefore vary in the following dimensions:

- Credential Format
- Credential Dataset
- Cryptographic Data

In the context of a single request, the batch of issued Credentials sent in response MUST share the same Credential Format and Credential Dataset, but SHOULD contain different Cryptographic Data. For example to achieve unlinkability between the Credentials, each credential should be bound to different cryptographic keys.

To issue multiple Verifiable Credentials with differing Credential Formats or Credential Datasets, multiple requests MUST be sent to the Credential Endpoint.

3.3.3. Issuance Flow Variations

The issuance can have multiple characteristics that can be combined depending on the use cases:

- Authorization Code Flow or Pre-Authorized Code Flow: The Credential Issuer can obtain End-User information to turn into a Verifiable Credential using End-User authentication and consent at the Credential Issuer's Authorization Endpoint (Authorization Code Flow) or using out-of-band mechanisms outside of the issuance flow (Pre-Authorized Code Flow).
- Wallet initiated or Issuer initiated: The request from the Wallet can be sent to the Credential Issuer without any gesture from the Credential Issuer (Wallet Initiated) or following the communication from the Credential Issuer (Issuer Initiated).
- Same-device or Cross-device Credential Offer: The End-User may receive the Credential Offer from the Credential Issuer either on the same device as the device the Wallet resides on, or through any other means, such as another device or postal mail, so that the Credential Offer can be communicated to the Wallet.
- Immediate or Deferred: The Credential Issuer can issue the Credential directly in response to the Credential Request (immediate) or requires time and needs the Wallet to come back to retrieve Credential (deferred).

The following subsections illustrate some of the authorization flows supported by this specification.

3.3.4. Identifying Credentials Being Issued Throughout the Issuance Flow

Below is the summary of how Credential(s) that are being issued are identified throughout the issuance flow:

- In the Credential Offer, the Credential Issuer identifies offered Credential Configurations using the `credential_configuration_ids` parameter.

- When the Wallet uses Authorization Details in the Authorization Request, the Wallet uses the `credential_configuration_id` parameter to identify the requested Credential Configurations. The Authorization Server returns an `authorization_details` parameter containing the `credential_identifiers` parameter in the Token Response, and the Wallet uses those `credential_identifier` values in the Credential Request.
- When the Wallet uses `scope` parameter in the Authorization Request, the `scope` value(s) are used to identify requested Credential Configurations. In this case, the Authorization Server has two options. If the Authorization Server supports returning `credential_identifiers` parameter in the Token Response, it MAY do so, in which case the Wallet uses those `credential_identifier` values in the Credential Request. If the Authorization Server does not support returning an `authorization_details` parameter containing the `credential_identifiers` parameter in the Token Response, the Wallet uses `credential_configuration_id` parameter in the Credential Request.

3.4. Authorization Code Flow

The Authorization Code Flow uses the grant type `authorization_code` as defined in [RFC6749] to issue Access Tokens.

Figure 1 shows the Authorization Code Flows with the two variations that can be implemented for the issuance of Credentials, as outlined in this specification:

- **Wallet-initiated variation**, described in Appendix H.4 or Appendix H.6;
- **Issuer-initiated variation**, described in Appendix H.1.

Please note that the diagram does not illustrate all the optional features defined in this specification.

```
+----------+    +-----------+         +----------------------+   +--------------------
+
| End-User |    |   Wallet  |         | Authorization Server |   | Credential Issuer
|
+----------+    +-----------+         +----------------------+   +--------------------
+
      |               |                            |                        |
      | (1a) End-User |                            |                        |
      |  selects      |  (1b) Credential Offer     |                        |
      |  Credential-->|  (credential type)         |                        |
      |               |<----------------------------------------------------|
      |               |                            |                        |
      |               |  (2) Obtains Issuer's      |                        |
      |               |      Credential Issuer     |                        |
      |               |      metadata              |                        |
      |               |---------------------------------------------------->|
      |               |                            |                        |
      |               |                            |  (3) Authorization     |
      |               |                            |      Request           |
      |               |                            |      (type(s) of       |
      |               |                            |      Credentials to    |
      |               |                            |      be issued)        |
      |               |--------------------------->|                        |
      |               |                            |                        |
      |  End-User Authentication / Consent         |                        |
      |               |                            |  (4) Authorization     |
      |               |                            |      Response (code)   |
      |               |<---------------------------|                        |
      |               |                            |                        |
      |               |                            |  (5) Token Request     |
      |               |                            |      (code)            |
      |               |--------------------------->|      Token Response    |
      |               |                            |      (Access Token)    |
      |               |<---------------------------|                        |
      |               |                            |                        |
      |               |  (6) Credential Request    |                        |
      |               |      (Access Token,        |                        |
      |               |       proof(s))            |                        |
      |               |---------------------------------------------------->|
      |               |                            |                        |
      |               |      Credential Response   |                        |
      |               |      with Credential(s)    |                        |
      |               |      OR Transaction ID     |                        |
      |               |<----------------------------------------------------|
```

Figure 1: Issuance using Authorization Code Flow

(1a) The Wallet-initiated flow begins as the End-User requests a Credential via the Wallet from the Credential Issuer. The End-User either selects a Credential from a pre-configured list of Credentials ready to be issued, or alternatively, the Wallet gives guidance to the End-User to select a Credential from a Credential Issuer based on the information it received in the presentation request from a Verifier.

(1b) The Issuer-initiated flow begins as the Credential Issuer generates a Credential Offer for certain Credential(s) that it communicates to the Wallet, for example, as a QR code or as a URI. The Credential Offer contains the Credential Issuer's URL and the information about the Credential(s) being offered. This step is defined in Section 4.1.

(2) The Wallet uses the Credential Issuer's URL to fetch the Credential Issuer metadata, as described in Section 12.2. The Wallet needs the metadata to learn the Credential types and formats that the Credential Issuer supports and to determine the Authorization Endpoint (OAuth 2.0 Authorization Server) as well as Credential Endpoint required to start the request. This specification supports configurations where the Credential Endpoint and the Authorization Endpoint are managed by either separate entities or a single entity.

(3) The Wallet sends an Authorization Request to the Authorization Endpoint. The Authorization Endpoint processes the Authorization Request, which typically includes authenticating the End-User and gathering End-User consent. Note: The Authorization Request may be sent as a Pushed Authorization Request.

(4) The Authorization Endpoint returns the Authorization Response with the Authorization Code upon successfully processing the Authorization Request.

Note: Steps (3) and (4) happen in the front channel, by redirecting the End-User via the User Agent. Those steps are defined in Section 5. The Authorization Server and the User Agent may exchange any further messages between the steps if required by the Authorization Server to authenticate the End-User. For example, the Authorization Server may request Credential presentation as a means to authenticate or identify the End-User during the issuance flow, as described in Appendix H.5.

(5) The Wallet sends a Token Request to the Token Endpoint with the Authorization Code obtained in Step (4). The Token Endpoint returns an Access Token in the Token Response upon successfully validating the Authorization Code. This step happens in the back-channel communication (direct communication between two systems using HTTP requests and responses without using redirects through an intermediary such as a browser). This step is defined in Section 6.

(6) The Wallet sends a Credential Request to the Credential Issuer's Credential Endpoint with the Access Token and (optionally) the proof(s) of possession of the private key of a key pair to which the Credential Issuer should bind the issued Credential to. Upon successfully validating Access Token and proof(s), the Credential Issuer returns a Credential in the Credential Response. This step is defined in Section 8.

If the Credential Issuer requires more time to issue a Credential, the Credential Issuer may return a Transaction ID and a time interval in the Credential Response. The Wallet may send a Deferred Credential Request with the Transaction ID to obtain a Credential after the specified time interval has passed, as defined in Section 9.

Note: This flow is based on OAuth 2.0 and the Authorization Code Grant type, but this specification can be used with other OAuth 2.0 grant types as well.

3.5. Pre-Authorized Code Flow

Figure 2 is a diagram of a Credential issuance using the Pre-Authorized Code Flow. In this flow, before initiating the flow with the Wallet, the Credential Issuer first conducts the steps required to prepare for Credential issuance, e.g., End-User authentication and authorization. Consequently, the Pre-Authorized Code is sent by the Credential Issuer to the Wallet. This flow does not use the Authorization Endpoint, and the Wallet exchanges the Pre-Authorized Code for the Access Token directly at the Token Endpoint. The Access Token is then used to request Credential issuance at the Credential Endpoint. See Appendix H.2 for the description of such a use case.

How the End-User provides information required for the issuance of a requested Credential to the Credential Issuer and the business processes conducted by the Credential Issuer to prepare a Credential are out of scope of this specification.

This flow uses the OAuth 2.0 Grant Type `urn:ietf:params:oauth:grant-type:pre-authorized_code`, which is defined in Section 4.1.1.

The following diagram is based on the Credential Issuer-initiated flow, as described in the use case in Appendix H.2. Please note that it does not illustrate all the optional features outlined in this specification.

```
+----------+   +-----------+            +----------------------+   +-----------------
--+
| End-User |   |   Wallet  |            | Authorization Server |   | Credential
Issuer |
+----------+   +-----------+            +----------------------+   +-----------------
--+
      |              |                              |                       |
      |              |  (1) End-User provides       |                       |
      |              |      information required    |                       |
      |              |      for the issuance of     |                       |
      |              |      a certain Credential    |                       |
      |              |----------------------------------------------------->|
      |              |                              |                       |
      |              |  (2) Credential Offer         |                       |
      |              |      (Pre-Authorized Code)   |                       |
      |              |<-----------------------------------------------------|
      |              |  (3) Obtains Issuer's        |                       |
      |              |      Credential Issuer       |                       |
      |              |      metadata                |                       |
      |              |----------------------------------------------------->|
      |   interacts  |                              |                       |
      |------------->|                              |                       |
      |              |                              |                       |
      |              |  (4) Token Request           |                       |
      |              |      (Pre-Authorized Code,   |                       |
      |              |       tx_code)               |                       |
      |              |----------------------------->|                       |
      |              |      Token Response          |                       |
      |              |      (access_token)          |                       |
      |              |<-----------------------------|                       |
      |              |                              |                       |
      |              |  (5) Credential Request      |                       |
      |              |      (access_token, proof(s))|                       |
      |              |----------------------------------------------------->|
      |              |      Credential Response     |                       |
      |              |      (Credential(s))         |                       |
      |              |<-----------------------------------------------------|
```

Figure 2: Issuance using Pre-Authorized Code Flow

(1) The Credential Issuer successfully obtains consent and End-User data required for the issuance of a requested Credential from the End-User using an Issuer-specific business process.

(2) The flow defined in this specification begins as the Credential Issuer generates a Credential Offer for certain Credential(s) and communicates it to the Wallet, for example, as a QR code or as a URI. The Credential Offer contains the Credential Issuer's URL, the information about the Credential(s) being offered, and the Pre-Authorized Code. This step is defined in Section 4.1.

(3) The Wallet uses the Credential Issuer's URL to fetch its metadata, as described in Section 12.2. The Wallet needs the metadata to learn the Credential types and formats that the Credential Issuer supports, and to determine the Token Endpoint (at the OAuth 2.0 Authorization Server) as well as the Credential Endpoint required to start the request.

(4) The Wallet sends the Pre-Authorized Code obtained in Step (2) in the Token Request to the Token Endpoint. The Wallet will additionally send a Transaction Code provided by the End-User, if it was required by the Credential Issuer. This step is defined in Section 6.

(5) This step is the same as Step (6) in the Authorization Code Flow.

It is important to note that anyone who possesses a valid Pre-Authorized Code, without further security measures, would be able to receive a VC from the Credential Issuer. Implementers MUST implement mitigations most suitable to the use case.

One such mechanism defined in this specification is the usage of Transaction Codes. The Credential Issuer indicates the usage of Transaction Codes in the Credential Offer and sends the Transaction Code to the End-User via a second channel different than the issuance flow. After the End-User provides the Transaction Code, the Wallet sends the Transaction Code within the Token Request, and the Authorization Server verifies the Transaction Code.

For more details and concrete mitigations, see Section 13.6.

4. Credential Offer Endpoint

This endpoint is used by a Credential Issuer that is already interacting with an End-User who wishes to initiate a Credential issuance. It is used to pass available information relevant for the Credential issuance to ensure a convenient and secure process.

4.1. Credential Offer

The Credential Issuer makes a Credential Offer by allowing the End-User to invoke the Wallet using the Wallet's Credential Offer Endpoint defined in Section 12.1. For example, by clicking a link and/or rendering a QR code containing the Credential Offer that the End-User can scan in a wallet or an arbitrary camera application.

Credential Issuers MAY also communicate Credential Offers directly to a Wallet's backend, but any mechanism for doing so is currently outside the scope of this specification.

The Credential Offer object, which is a JSON-encoded object with the Credential Offer parameters, can be sent by value or by reference.

The Credential Offer contains a single URI query parameter, either `credential_offer` or `credential_offer_uri`:

- `credential_offer`: Object with the Credential Offer parameters. This MUST NOT be present when the `credential_offer_uri` parameter is present.
- `credential_offer_uri`: String that is a URL using the `https` scheme referencing a resource containing a JSON object with the Credential Offer parameters. This MUST NOT be present when the `credential_offer` parameter is present.

For security considerations, see Section 13.5.

4.1.1. Credential Offer Parameters

This specification defines the following parameters for the JSON-encoded Credential Offer object:

- `credential_issuer`: REQUIRED. The URL of the Credential Issuer, as defined in Section 12.2.1, from which the Wallet is requested to obtain one or more Credentials. The Wallet uses it to obtain the Credential Issuer's Metadata following the steps defined in Section 12.2.2.
- `credential_configuration_ids`: REQUIRED. A non-empty array of unique strings that each identify one of the keys in the name/value pairs stored in the `credential_configurations_supported` Credential Issuer metadata. The Wallet uses these string values to obtain the respective object that contains information about the Credential being offered as defined in Section 12.2.4. For example, these string values can be used to obtain `scope` values to be used in the Authorization Request.
- `grants`: OPTIONAL. Object indicating to the Wallet the Grant Types the Credential Issuer's Authorization Server is prepared to process for this Credential Offer. Every grant is represented by a name/value pair. The name is the Grant Type identifier; the value is an object that contains parameters either determining the way the Wallet MUST use the particular grant and/or parameters the Wallet MUST send with the respective request(s). If `grants` is not present or is empty, the Wallet MUST determine the Grant Types the Credential Issuer's Authorization Server supports using the respective metadata. When multiple grants are present, it is at the Wallet's discretion which one to use.

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

The following values are defined by this specification:

- Grant Type `authorization_code`:
  - `issuer_state`: OPTIONAL. String value created by the Credential Issuer and opaque to the Wallet that is used to bind the subsequent Authorization Request with a context set up during previous process steps. If the Wallet decides to use the Authorization Code Flow and received a value for this parameter, it MUST include it in the subsequent Authorization Request to the Authorization Server as the `issuer_state` parameter value.
  - `authorization_server`: OPTIONAL string that the Wallet can use to identify the Authorization Server to use with this grant type when `authorization_servers` parameter in the Credential Issuer metadata has multiple entries. It MUST NOT be used otherwise. The value of this parameter MUST match with one of the values in the `authorization_servers` array obtained from the Credential Issuer metadata.

- Grant Type `urn:ietf:params:oauth:grant-type:pre-authorized_code`:
  - `pre-authorized_code`: REQUIRED. The code representing the Credential Issuer's authorization for the Wallet to obtain Credentials of a certain type. This code MUST be short lived and single use. If the Wallet decides to use the Pre-Authorized Code Flow, this parameter value MUST be included in the subsequent Token Request with the Pre-Authorized Code Flow.
  - `tx_code`: OPTIONAL. Object indicating that a Transaction Code is required if present, even if empty. It describes the requirements for a Transaction Code, which the Authorization Server expects the End-User to present along with the Token Request in a Pre-Authorized Code Flow. If the Authorization Server does not expect a Transaction Code, this object is absent; this is the default. The Transaction Code is intended to bind the Pre-Authorized Code to a certain transaction to prevent replay of this code by an attacker that, for example, scanned the QR code while standing behind the legitimate End-User. It is RECOMMENDED to send the Transaction Code via a separate channel. If the Wallet decides

to use the Pre-Authorized Code Flow, the Transaction Code value MUST be sent in the `tx_code` parameter with the respective Token Request as defined in Section 6.1. If no `length`, `description`, or `input_mode` is given, this object MAY be empty.
    - `input_mode` : OPTIONAL. String specifying the input character set. Possible values are `numeric` (only digits) and `text` (any characters). The default is `numeric`.
    - `length`: OPTIONAL. Integer specifying the length of the Transaction Code. This helps the Wallet to render the input screen and improve the user experience.
    - `description`: OPTIONAL. String containing guidance for the Holder of the Wallet on how to obtain the Transaction Code, e.g., describing over which communication channel it is delivered. The Wallet is RECOMMENDED to display this description next to the Transaction Code input screen to improve the user experience. The length of the string MUST NOT exceed 300 characters. The `description` does not support internationalization, however the Issuer MAY detect the Holder's language by previous communication or an HTTP Accept-Language header within an HTTP GET request for a Credential Offer URI.
  - `authorization_server`: OPTIONAL string that the Wallet can use to identify the Authorization Server to use with this grant type when `authorization_servers` parameter in the Credential Issuer metadata has multiple entries. It MUST NOT be used otherwise. The value of this parameter MUST match with one of the values in the `authorization_servers` array obtained from the Credential Issuer metadata.

The following non-normative example shows a Credential Offer object where the Credential Issuer can offer the issuance of two different Credentials (which may even be of different formats):

```
{
  "credential_issuer": "https://credential-issuer.example.com",
  "credential_configuration_ids": [
    "UniversityDegreeCredential",
    "org.iso.18013.5.1.mDL"
  ],
  "grants": {
    "urn:ietf:params:oauth:grant-type:pre-authorized_code": {
      "pre-authorized_code": "oaKazRN8I0IbtZ0C7JuMn5",
      "tx_code": {
        "length": 4,
        "input_mode": "numeric",
        "description": "Please provide the one-time code that was sent via e-mail"
      }
    }
  }
}
```

4.1.2. Sending Credential Offer by Value Using `credential_offer` Parameter

Below is a non-normative example of a Credential Offer passed by value (with line breaks within values for display purposes only):

```
GET /credential_offer?
  credential_offer=%7B%22credential_issuer%22:%22https://credential-issuer.exam
  ple.com%22,%22credential_configuration_ids%22:%5B%22UniversityDegree_JWT%22,%
  22org.iso.18013.5.1.mDL%22%5D,%22grants%22:%7B%22urn:ietf:params:oauth:grant-
  type:pre-authorized_code%22:%7B%22pre-authorized_code%22:%22oaKazRN8I0IbtZ0C7
  JuMn5%22,%22tx_code%22:%7B%7D%7D%7D%7D
```

The following is a non-normative example of a Credential Offer that can be included in a QR code or a link used to invoke a Wallet deployed as a native app (with line breaks within values for display purposes only):

```
openid-credential-offer://?
  credential_offer=%7B%22credential_issuer%22:%22https://credential-issuer.exam
  ple.com%22,%22credential_configuration_ids%22:%5B%22org.iso.18013.5.1.mDL%22%
  5D,%22grants%22:%7B%22urn:ietf:params:oauth:grant-type:pre-authorized_code%22
  :%7B%22pre-authorized_code%22:%22oaKazRN8I0IbtZ0C7JuMn5%22,%22tx_code%22:%7B%
  22input_mode%22:%22text%22,%22description%22:%22Please%20enter%20the%20serial
  %20number%20of%20your%20physical%20drivers%20license%22%7D%7D%7D%7D
```

4.1.3. Sending Credential Offer by Reference Using `credential_offer_uri` Parameter

Upon receipt of the `credential_offer_uri`, the Wallet MUST send an HTTP GET request to the URI to retrieve the referenced Credential Offer Object, unless it is already cached, and parse it to recreate the Credential Offer parameters.

Note: The Credential Issuer SHOULD use a unique URI for each Credential Offer utilizing distinct parameters, or otherwise prevent the Credential Issuer from caching the `credential_offer_uri`.

Below is a non-normative example of this fetch process:

```
GET /credential_offer HTTP/1.1
Host: server.example.com
```

The response from the Credential Issuer that contains a Credential Offer Object MUST use the media type `application/json`.

This ability to pass the Credential Offer by reference is particularly useful for large Credential Offer objects.

When the Credential Offer is displayed as a QR code, it would usually contain the Credential Offer by reference due to the size limitations of the QR codes. Below is a non-normative example:

```
openid-credential-offer://?
  credential_offer_uri=https%3A%2F%2Fserver%2Eexample%2Ecom%2Fcredential-offer
  %2FGkurKxf5T0Y-mnPFCHqWOMiZi4VS138cQO_V7PZHAdM
```

Below is a non-normative example of a response from the Credential Issuer that contains a Credential Offer Object used to encourage the Wallet to start an Authorization Code Flow:

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

{
  "credential_issuer": "https://credential-issuer.example.com",
  "credential_configuration_ids": [
    "UniversityDegreeCredential"
  ],
  "grants": {
    "authorization_code": {
      "issuer_state": "eyJhbGciOiJSU0Et...FYUaBy"
    }
  }
}
```

Below is a non-normative example of a Credential Offer Object for a Pre-Authorized Code Flow (with a Credential type reference):

```
{
  "credential_issuer": "https://credential-issuer.example.com",
  "credential_configuration_ids": [
    "UniversityDegree_LDP_VC"
  ],
  "grants": {
    "urn:ietf:params:oauth:grant-type:pre-authorized_code": {
      "pre-authorized_code": "adhjhdjajkdkhjhdj",
      "tx_code": {}
    }
  }
}
```

When retrieving the Credential Offer from the Credential Offer URL, the `application/json` media type MUST be used. The Credential Offer cannot be signed and MUST NOT use `application/jwt` with `"alg": "none"`.

4.2. Credential Offer Response

The Wallet does not create a response. UX control stays with the Wallet after completion of the process.

5. Authorization Endpoint

The Authorization Endpoint is used in the same manner as defined in [RFC6749]. Implementers SHOULD follow the best current practices for OAuth 2.0 Security given in [BCP240].

When the grant type `authorization_code` is used, it is RECOMMENDED to use PKCE [RFC7636] and Pushed Authorization Requests [RFC9126]. PKCE prevents authorization code interception attacks. Pushed Authorization Requests ensure the integrity and authenticity of the authorization request.

5.1. Authorization Request

An Authorization Request is an OAuth 2.0 Authorization Request as defined in Section 4.1.1 of [RFC6749], which requests that access be granted to the Credential Endpoint, as defined in Section 8.

There are two possible methods for requesting the issuance of a specific Credential type in an Authorization Request. The first method involves using the `authorization_details` request parameter, as defined in [RFC9396], containing one or more authorization details of type `openid_credential`, as specified in Section

5.1.1. The second method utilizes scopes, as outlined in Section 5.1.2.

See Section 3.3.4 for the summary of the options how requested Credential(s) are identified throughout the Issuance flow.

5.1.1. Using Authorization Details Parameter

Credential Issuers MAY support requesting authorization to issue a Credential using the `authorization_details` parameter.

The request parameter `authorization_details` defined in Section 2 of [RFC9396] MUST be used to convey the details about the Credentials the Wallet wants to obtain. This specification introduces a new authorization details type `openid_credential` and defines the following parameters to be used with this authorization details type:

- `type`: REQUIRED. String that determines the authorization details type. It MUST be set to `openid_credential` for the purpose of this specification.
- `credential_configuration_id`: REQUIRED. String specifying a unique identifier of the Credential being described in the `credential_configurations_supported` map in the Credential Issuer Metadata as defined in Section 12.2.4. The referenced object in the `credential_configurations_supported` map conveys the details of the requested Credential, such as the format and format-specific parameters like `vct` for SD-JWT VC or `doctype` for ISO mdoc. This specification defines those Credential Format specific Issuer Metadata in Appendix A.
- `claims`: OPTIONAL. A non-empty array of claims description objects as defined in Appendix B.1.

Additional `authorization_details` data fields MAY be defined and used when the `type` value is `openid_credential`. Note that this effectively defines an authorization details type that is never considered invalid due to unknown fields.

The following is a non-normative example of an `authorization_details` object with a `credential_configuration_id`:

```
[
  {
    "type": "openid_credential",
    "credential_configuration_id": "UniversityDegreeCredential"
  }
]
```

If the Credential Issuer metadata contains an `authorization_servers` parameter, the authorization detail's `locations` common data field MUST be set to the Credential Issuer Identifier value. A non-normative example for a deployment where an Authorization Server protects multiple Credential Issuers would look like this:

```
[
  {
    "type": "openid_credential",
    "locations": [
      "https://credential-issuer.example.com"
    ],
    "credential_configuration_id": "UniversityDegreeCredential"
  }
]
```

Below is a non-normative example of an Authorization Request using the `authorization_details` parameter that would be sent by the User Agent to the Authorization Server in response to an HTTP 302 redirect response by the Wallet (with line breaks within values for display purposes only):

```
GET /authorize?
  response_type=code
  &client_id=s6BhdRkqt3
  &code_challenge=E9Melhoa2OwvFrEMTJguCHaoeK1t8URWbuGJSstw-cM
  &code_challenge_method=S256
  &authorization_details=%5B%7B%22type%22%3A%20%22openid_credential%22%2C%20%22
    credential_configuration_id%22%3A%20%22UniversityDegreeCredential%22%7D%5D
  &redirect_uri=https%3A%2F%2Fwallet.example.org%2Fcb

Host: server.example.com
```

This non-normative example requests authorization to issue two different Credentials:

```
[
  {
    "type": "openid_credential",
    "credential_configuration_id": "UniversityDegreeCredential"
  },
  {
    "type": "openid_credential",
    "credential_configuration_id": "org.iso.18013.5.1.mDL"
  }
]
```

Note: Applications MAY combine authorization details of type `openid_credential` with any other authorization details types in an Authorization Request.

5.1.2. Using scope Parameter to Request Issuance of a Credential

Credential Issuers MAY support requesting authorization to issue a Credential using the OAuth 2.0 `scope` parameter.

When the Wallet does not know which scope value to use to request issuance of a certain Credential, it can discover it using the `scope` Credential Issuer metadata parameter defined in Section 12.2.4. When the flow starts with a Credential Offer, the Wallet can use the `credential_configuration_ids` parameter values to identify object(s) in the `credential_configurations_supported` map in the Credential Issuer metadata parameter and use the `scope` parameter value from that object.

The Wallet can discover the scope values using other options such as normative text in a profile of this specification that defines scope values along with a description of their semantics.

The concrete `scope` values are out of scope of this specification.

The Wallet MAY combine scopes discovered from the Credential Issuer metadata with the scopes discovered from the Authorization Server metadata.

It is RECOMMENDED to use collision-resistant scope values.

Credential Issuers MUST interpret each scope value as a request to access the Credential Endpoint as defined in Section 8 for the issuance of a Credential type identified by that scope value. Multiple scope values MAY be present in a single request whereby each occurrence MUST be interpreted individually.

Credential Issuers MUST ignore unknown scope values in a request.

If the Credential Issuer metadata contains an `authorization_servers` property, it is RECOMMENDED to use a `resource` parameter [RFC8707] whose value is the Credential Issuer's identifier value to allow the Authorization Server to differentiate Credential Issuers.

Below is a non-normative example of an Authorization Request provided by the Wallet to the Authorization Server using the scope `UniversityDegreeCredential` and in response to an HTTP 302 redirect (with line breaks within values for display purposes only):

```
GET /authorize?
  response_type=code
  &scope=UniversityDegreeCredential
  &resource=https%3A%2F%2Fcredential-issuer.example.com
  &client_id=s6BhdRkqt3
  &code_challenge=E9Melhoa2OwvFrEMTJguCHaoeK1t8URWbuGJSstw-cM
  &code_challenge_method=S256
  &redirect_uri=https%3A%2F%2Fwallet.example.org%2Fcb
Host: server.example.com
```

If a scope value related to Credential issuance and the `authorization_details` request parameter containing objects of type `openid_credential` are both present in a single request, the Credential Issuer MUST interpret these individually. However, if both request the same Credential type, then the Credential Issuer MUST follow the request as given by the authorization details object.

5.1.3. Additional Request Parameters

This specification defines the following request parameter that can be supplied in an Authorization Request:

- `issuer_state`: OPTIONAL. String value identifying a certain processing context at the Credential Issuer. A value for this parameter is typically passed in a Credential Offer from the Credential Issuer to the Wallet (see Section 4.1). This request parameter is used to pass the `issuer_state` value back to the Credential Issuer.

Note: When processing the Authorization Request, the Credential Issuer MUST take into account that the `issuer_state` is not guaranteed to originate from this Credential Issuer in all circumstances. It could have been injected by an attacker.

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

5.1.4. Pushed Authorization Request

Use of Pushed Authorization Requests is RECOMMENDED to ensure confidentiality, integrity, and authenticity of the request data and to avoid issues caused by large requests sizes.

Below is a non-normative example of a Pushed Authorization Request:

```
POST /par HTTP/1.1
Host: server.example.com
OAuth-Client-Attestation: eyJ...
OAuth-Client-Attestation-PoP: eyJ...
Content-Type: application/x-www-form-urlencoded

response_type=code
&client_id=CLIENT1234
&code_challenge=E9Melhoa2OwvFrEMTJguCHaoeK1t8URWbuGJSstw-cM
&code_challenge_method=S256
&redirect_uri=https%3A%2F%2Fwallet.example.org%2Fcb
&authorization_details=...
```

Below is a non-normative example of a response to a successful request:

```
HTTP/1.1 201 Created
Content-Type: application/json
Cache-Control: no-cache, no-store

{
  "request_uri": "urn:ietf:params:oauth:request_uri:6esc_11ACC5bwc014ltc14eY22c",
  "expires_in": 60
}
```

Below is a non-normative example of the GET request that might subsequently be sent by the Browser:

```
GET /authorize?client_id=s6BhdRkqt3
  &request_uri=urn%3Aietf%3Aparams%3Aoauth%3Arequest_uri%3A6esc_11ACC5bwc014ltc14eY22c
Host: server.example.com
```

5.2. Successful Authorization Response

Authorization Responses MUST be made as defined in [RFC6749].

Below is a non-normative example of a successful Authorization Response:

```
HTTP/1.1 302 Found
Location: https://wallet.example.org/cb?
  code=SplxlOBeZQQYbYS6WxSbIA
```

5.3. Authorization Error Response

The Authorization Error Response MUST be made as defined in [RFC6749].

Below is a non-normative example of an unsuccessful Authorization Response:

```
HTTP/1.1 302 Found
Location: https://wallet.example.org/cb?
  error=invalid_request
  &error_description=Unsupported%20response_type%20value
```

6. Token Endpoint

The Token Endpoint issues an Access Token and, optionally, a Refresh Token in exchange for the Authorization Code that Client obtained in a successful Authorization Response. It is used in the same manner as defined in [RFC6749]. Implementers SHOULD follow the best current practices for OAuth 2.0 Security given in [BCP240].

6.1. Token Request

The Token Request is made as defined in Section 4.1.3 of [RFC6749].

The following are the extension parameters to the Token Request used in the Pre-Authorized Code Flow defined in Section 3.5:

- `pre-authorized_code`: The code representing the authorization to obtain Credentials of a certain type. This parameter MUST be present if the `grant_type` is `urn:ietf:params:oauth:grant-type:pre-authorized_code`.
- `tx_code`: OPTIONAL. String value containing a Transaction Code value itself. This value MUST be present if a `tx_code` object was present in the Credential Offer (including if the object was empty). This parameter MUST only be used if the `grant_type` is `urn:ietf:params:oauth:grant-type:pre-authorized_code`.

Requirements around how the Wallet identifies and, if applicable, authenticates itself with the Authorization Server in the Token Request depend on the Client type defined in Section 2.1 of [RFC6749] and the Client authentication method indicated in the `token_endpoint_auth_method` Client metadata (as defined in [RFC7591]). The requirements specified in Sections 4.1.3 and 3.2.1 of [RFC6749] MUST be followed.

For the Pre-Authorized Code Grant Type, authentication of the Client is OPTIONAL, as described in Section 3.2.1 of OAuth 2.0 [RFC6749], and, consequently, the `client_id` parameter is only needed when a form of Client Authentication that relies on this parameter is used.

If the Token Request contains an `authorization_details` parameter (as defined by [RFC9396]) of type `openid_credential` and the Credential Issuer's metadata contains an `authorization_servers` parameter, the `authorization_details` object MUST contain the Credential Issuer's identifier in the `locations` element.

If the Token Request contains a scope value related to Credential issuance and the Credential Issuer's metadata contains an `authorization_servers` parameter, it is RECOMMENDED to use a `resource` parameter [RFC8707] whose value is the Credential Issuer's identifier value to allow the Authorization Server to differentiate Credential Issuers.

When the Pre-Authorized Grant Type is used, it is RECOMMENDED that the Credential Issuer issues an Access Token valid only for the Credentials indicated in the Credential Offer (see Section 4.1). The Wallet SHOULD obtain a separate Access Token if it wants to request issuance of any Credentials that were not included in the Credential Offer, but were discoverable from the Credential Issuer's `credential_configurations_supported` metadata parameter.

Additional Token Request parameters MAY be defined and used, as described in [RFC6749]. The Authorization Server MUST ignore any unrecognized parameters.

Below is a non-normative example of a Token Request in an Authorization Code Flow:

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code
&code=SplxlOBeZQQYbYS6WxSbIA
&code_verifier=dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
&redirect_uri=https%3A%2F%2Fwallet.example.org%2Fcb
&client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer
&client_assertion=eyJhbGciOiJSU...
```

6.1.1. Request Credential Issuance using `authorization_details` Parameter

Credential Issuers MAY support requesting authorization to issue a Credential using the `authorization_details` parameter. This is particularly useful if the Credential Issuer offered multiple Credential Configurations in the Credential Offer of a Pre-Authorized Code Flow.

The Wallet can use `authorization_details` in the Token Request to request a specific Credential Configuration in both the Authorization Code Flow and the Pre-Authorized Code Flow. The value of the `authorization_details` parameter is defined in Section 5.1.1.

Below is a non-normative example of a Token Request in a Pre-Authorized Code Flow (without Client Authentication):

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=urn:ietf:params:oauth:grant-type:pre-authorized_code
&pre-authorized_code=SplxlOBeZQQYbYS6WxSbIA
&tx_code=493536
&authorization_details=%5B%7B%22type%22%3A%20%22openid_credential%22%2C%20%22
    credential_configuration_id%22%3A%20%22UniversityDegreeCredential%22%7D%5D
```

The Wallet may send the `authorization_details` parameter in the Token Request even when the parameter has been previously sent in the Authorization Request as described in Section 6.1 of [RFC9396]. It allows the AS to make a decision based on whether the Wallet is asking for more or less access than the previous request. With respect to `authorization_details` items using the `credential_configuration_id` introduced in this specification, it is RECOMMENDED that the AS would accept a request from the Wallet containing a subset of `credential_configuration_id` parameters received in the original Authorization Request and issue a token for the reduced set.

6.2. Successful Token Response

Token Responses are made as defined in [RFC6749].

The Authorization Server might decide to authorize issuance of multiple instances for each Credential requested in the Authorization Request. Each Credential instance is described using the same entry in the `credential_configurations_supported` Credential Issuer metadata, but contains different claim values or different subset of claims within the claims set identified by the `credential_configuration_id`.

In addition to the response parameters defined in [RFC6749], the Authorization Server MAY return the following parameters:

- `authorization_details`: REQUIRED when the `authorization_details` parameter, as defined in Section 5.1.1, is used in either the Authorization Request or Token Request. OPTIONAL when `scope` parameter was used to request issuance of a Credential of a certain Credential Configuration. It is a non-empty array of objects, as defined in Section 7 of [RFC9396]. In addition to the parameters defined in Section 5.1.1, this specification defines the following parameter to be used with the authorization details type `openid_credential` in the Token Response:
    - `credential_identifiers`: REQUIRED. A non-empty array of strings, each uniquely identifying a Credential Dataset that can be issued using the Access Token returned in this response. Each of these Credential Datasets corresponds to the Credential Configuration referenced in the `credential_configuration_id` parameter. The Wallet MUST use these identifiers together with an Access Token in subsequent Credential Requests. See Section 3.3.4 for the summary of the options how requested Credential(s) are identified throughout the Issuance flow.

Additional Token Response parameters MAY be defined and used, as described in [RFC6749]. The Wallet MUST ignore any unrecognized parameters in the Token Response. An included `authorization_details` parameter MAY also have additional data fields defined and used when the `type` value is `openid_credential`. The Wallet MUST ignore any unrecognized data fields in the `authorization_details` present in the Token Response.

Below is a non-normative example of a Token Response when the `authorization_details` parameter was used to request issuance of a certain Credential type:

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

{
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6Ikp..sHQ",
  "token_type": "Bearer",
  "expires_in": 86400,
  "authorization_details": [
    {
      "type": "openid_credential",
      "credential_configuration_id": "UniversityDegreeCredential",
      "credential_identifiers": [
        "CivilEngineeringDegree-2023",
        "ElectricalEngineeringDegree-2023"
      ]
    }
  ]
}
```

6.3. Token Error Response

If the Token Request is invalid or unauthorized, the Authorization Server constructs the error response as defined as in Section 5.2 of OAuth 2.0 [RFC6749].

The following additional clarifications are provided for some of the error codes already defined in [RFC6749]:

`invalid_request`:

- The Authorization Server does not expect a Transaction Code in the Pre-Authorized Code Flow but the Client provides a Transaction Code.
- The Authorization Server expects a Transaction Code in the Pre-Authorized Code Flow but the Client does not provide a Transaction Code.

`invalid_grant`:

- The Authorization Server expects a Transaction Code in the Pre-Authorized Code Flow but the Client provides the wrong Transaction Code.
- The End-User provides the wrong Pre-Authorized Code or the Pre-Authorized Code has expired.

`invalid_client`:

- The Client tried to send a Token Request with a Pre-Authorized Code without a Client ID but the Authorization Server does not support anonymous access.

Below is a non-normative example of a Token Error Response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store

{
  "error": "invalid_request"
}
```

7. Nonce Endpoint

This endpoint allows a Client to acquire a fresh `c_nonce` value. A Credential Issuer that requires `c_nonce` values to be incorporated into proofs in the Credential Request (see Section 8.2) MUST offer a Nonce Endpoint.

The `nonce_endpoint` Credential Issuer Metadata parameter, as defined in Section 12.2.4, contains the URL of the Credential Issuer's Nonce Endpoint.

7.1. Nonce Request

A request for a nonce is made by sending an HTTP POST request to the URL provided in the `nonce_endpoint` Credential Issuer Metadata parameter. The Nonce Endpoint is not a protected resource, meaning the Wallet does not need to supply an access token to access it.

Below is a non-normative example of a Nonce Request:

```
POST /nonce HTTP/1.1
Host: credential-issuer.example.com
Content-Length: 0
```

7.2. Nonce Response

The Credential Issuer provides a nonce value in an HTTP response with a 2xx status code and the following parameters included as top-level members in the message body of the HTTP response using the application/json media type:

- `c_nonce`: REQUIRED. String containing a challenge to be used when creating a proof of possession of the key (see Section 8.2). It is at the discretion of the Credential Issuer when to return a new challenge value as opposed to the one returned in the previous request. New challenge values MUST be unpredictable.

Due to the temporal nature of the `c_nonce` value, the Credential Issuer MUST make the response uncacheable by adding a `Cache-Control` header field including the value `no-store`.

The Credential Issuer MAY provide a DPoP nonce in an HTTP header as defined in Section 8.2 of [RFC9449]. In this case, the Wallet uses the new nonce value in the DPoP proof when presenting an access token at the Credential Endpoint.

Below is a non-normative example of a Nonce Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
DPoP-Nonce: eyJ7S_zG.eyJH0-Z.HX4w-7v

{
  "c_nonce": "wKI4LT17ac15ES9bw8ac4"
}
```

8. Credential Endpoint

The Credential Endpoint issues one or more Credentials of the same Credential Configuration and Credential Dataset (as approved by the End-User) upon presentation of a valid Access Token representing this approval. Support for this endpoint is REQUIRED.

Communication with the Credential Endpoint MUST utilize TLS.

The Client sends a Credential Request to obtain:

- one Credential; or
- multiple Credential instances of the same Credential Configuration and Credential Dataset, each with distinct cryptographic material.

If the Issuer supports the issuance of multiple Credentials, the Client can send several consecutive Credential Requests to obtain multiple Credentials in a chosen sequence.

8.1. Binding the Issued Credential to the Identifier of the End-User Possessing that Credential

The issued Credential SHOULD be cryptographically bound to the identifier of the End-User who possesses the Credential. Cryptographic Key Binding allows the Verifier to verify during the presentation of a Credential that the End-User presenting a Credential is the same End-User to whom that Credential was issued. For non-cryptographic types of binding and Credentials issued without any binding, see the Implementation Considerations in Section 14.1 and Section 14.2.

Note: Claims in the Credential are about the subject of the Credential, which is often the End-User who possesses it.

For Cryptographic Key Binding, the Client has different options to provide Cryptographic Key Binding material for a requested Credential within a proof of a certain proof type. A proof type may provide the cryptographic public key(s) either with corresponding proof(s) of possession of the private key(s) or with key attestation(s). Proof types are defined in Appendix F.

8.2. Credential Request

A Client makes a Credential Request to the Credential Endpoint by sending the following parameters in the entity-body of an HTTP POST request. The Credential Request MAY be encrypted (on top of TLS) using the `credential_request_encryption` parameter in Section 12.2 as specified in Section 10.

- `credential_identifier`: REQUIRED when an Authorization Details of type `openid_credential` was returned from the Token Response. It MUST NOT be used otherwise. A string that identifies a Credential Dataset that is requested for issuance. When this parameter is used, the `credential_configuration_id` MUST NOT be present.
- `credential_configuration_id`: REQUIRED if a `credential_identifiers` parameter was not returned from the Token Response as part of the `authorization_details` parameter. It MUST NOT be used otherwise. String that uniquely identifies one of the keys in the name/value pairs stored in the `credential_configurations_supported` Credential Issuer metadata. The corresponding object in the `credential_configurations_supported` map MUST contain one of the value(s) used in the `scope` parameter in the Authorization Request. When this parameter is used, the `credential_identifier` MUST NOT be present.
- `proofs`: OPTIONAL. Object providing one or more proof of possessions of the cryptographic key material to which the issued Credential instances will be bound to. The `proofs` parameter contains exactly one parameter named as the proof type in Appendix F, the value set for this parameter is a non-empty array containing parameters as defined by the corresponding proof type.
- `credential_response_encryption`: OPTIONAL. Object containing information for encrypting the Credential Response. If this request element is not present, the corresponding credential response returned is not encrypted.
  - `jwk`: REQUIRED. Object containing a single public key as a JWK used for encrypting the Credential Response.
  - `enc`: REQUIRED. JWE [RFC7516] `enc` algorithm [RFC7518] for encrypting Credential Responses.
  - `zip`: OPTIONAL. JWE [RFC7516] `zip` algorithm [RFC7518] for compressing Credential Responses prior to encryption. If absent then compression MUST not be used.

See Section 3.3.4 for the summary of the options how requested Credential(s) are identified throughout the Issuance flow.

The proof type contained in the `proofs` parameter is an extension point that enables the use of different types of proofs for different cryptographic schemes.

The proof(s) in the `proofs` parameter MUST incorporate the Credential Issuer Identifier (audience) and, if the Credential Issuer has a Nonce Endpoint, a `c_nonce` value to allow the Credential Issuer to detect freshness. The way that data is incorporated depends on the key proof type. In a JWT, for example, the `c_nonce` value is conveyed in the `nonce` claim, whereas the audience is conveyed in the `aud` claim. In a Linked Data proof, for example, the `c_nonce` is included as the `challenge` element in the key proof object and the Credential Issuer (the intended audience) is included as the `domain` element.

The `proofs` parameter MUST be present if the `proof_types_supported` parameter is present in the `credential_configurations_supported` parameter of the Issuer metadata for the requested Credential.

The `c_nonce` value is retrieved from the Nonce Endpoint as defined in Section 7.

Additional Credential Request parameters MAY be defined and used. The Credential Issuer MUST ignore any unrecognized parameters.

The Credential Issuer indicates support for encrypted requests by including the `credential_request_encryption` parameter in the Credential Issuer Metadata. The Client MAY encrypt the request when `encryption_required` is `false` and MUST do so when `encryption_required` is `true`.

When performing Credential Request encryption, the Client MUST encode the information in the Credential Request in a JWT as specified by Section 10, using the parameters from the `credential_request_encryption` object in the Credential Issuer Metadata.

If the Credential Request is not encrypted, the media type of the request MUST be set to `application/json`.

Below is a non-normative example of a Credential Request for a Credential in [ISO.18013-5] format using the Credential configuration identifier and a key proof type `jwt` (with line breaks within values for display purposes only):

```
POST /credential HTTP/1.1
Host: server.example.com
Content-Type: application/json
Authorization: Bearer czZCaGRSa3F0MzpnWDFmQmF0M2JW

{
  "credential_configuration_id": "org.iso.18013.5.1.mDL",
  "proofs": {
    "jwt": [
      "eyJraWQiOiJkaWQ6ZXhhbXBsZTplYmZlYjFmNzEyZWJjNmYxYzI3NmUxMmVjMjEva2V5cy8x
       IiwiYWxnIjoiRVMyNTYiLCJ0eXAiOiJKV1QifQ"
    ]
  }
}
```

Below is a non-normative example of a Credential Request for two Credential instances in an IETF SD-JWT VC [I-D.ietf-oauth-sd-jwt-vc] format using a Credential identifier from the Token Response and key proof type `jwt` (with line breaks within values for display purposes only):

```
POST /credential HTTP/1.1
Host: server.example.com
Content-Type: application/json
Authorization: Bearer czZCaGRSa3F0MzpnWDFmQmF0M2JW

{
  "credential_identifier": "CivilEngineeringDegree-2023",
  "proofs": {
    "jwt": [
      "eyJ0eXAiOiJvcGVuaWQ0dmNpLXByb29mK2p3dCIsImFsZyI6IkVTMjU2IiwiandrIjp7Imt0
       eSI6IkVDIiwiY3J2IjoiUC0yNTYiLCJ4IjoiblVXQW9BdjNYWml0aDhFN2kxOU9kYXhPTFlG
       T3dNLVoyRXVNMDJUaXJUNCIsInkiOiJIc2tIVThCalVpMVU5WHFpN1N3bWo4Z3dBS18weGtj
       RGpFV183MVNvc0VZIn19",
      "eyJraWQiOiJkaWQ6ZXhhbXBsZTplYmZlYjFmNzEyZWJjNmYxYzI3NmUxMmVjMjEva2V5cy8x
       IiwiYWxnIjoiRVMyNTYiLCJ0eXAiOiJKV1QifQ"
    ]
  }
}
```

Below is a non-normative example of a Credential Request for one Credential in W3C VCDM format using a Credential identifier from the Token Response and key proof type `di_vp` (with line breaks within values for display purposes only):

```
POST /credential HTTP/1.1
Host: server.example.com
Content-Type: application/json
Authorization: BEARER czZCaGRSa3F0MzpnWDFmQmF0M2JW

{
  "credential_identifier": "CivilEngineeringDegree-2023",
  "proofs": {
    "di_vp": [
      {
        "@context": [
          "https://www.w3.org/ns/credentials/v2",
          "https://www.w3.org/ns/credentials/examples/v2"
        ],
        "type": [
          "VerifiablePresentation"
        ],
        "holder": "did:key:z6MkvrFpBNCoYewiaeBLgjUDvLxUtnK5R6mqh5XPvLsrPsro",
        "proof": [
          {
            "type": "DataIntegrityProof",
            "cryptosuite": "eddsa-2022",
            "proofPurpose": "authentication",
            "verificationMethod": "did:key:z6MkvrFpBNCoYewiaeBLgjUDvLxUtnK5R6mq
             h5XPvLsrPsro#z6MkvrFpBNCoYewiaeBLgjUDvLxUtnK5R6mqh5XPvLsrPsro",
            "created": "2023-03-01T14:56:29.280619Z",
            "challenge": "82d4cb36-11f6-4273-b9c6-df1ac0ff17e9",
            "domain": "did:web:audience.company.com",
            "proofValue": "z5hrbHzZiqXHNpLq6i7zePEUcUzEbZKmWfNQzXcUXUrqF7bykQ7A
             CiWFyZdT2HcptF1zd1t7NhfQSdqrbPEjZceg7"
          }
        ]
      }
    ]
  }
}
```

The Credential Issuer indicates support for encrypted responses by including the `credential_response_encryption` parameter in the Credential Issuer Metadata. The Client MAY request encrypted responses by providing its encryption parameters in the Credential Request when `encryption_required` is `false` and MUST do so when `encryption_required` is `true`. Credential Request encryption MUST be used if the `credential_response_encryption` parameter is included, to prevent it being substituted by an attacker.

8.3. Credential Response

The Credential Response can either be returned immediately or in a deferred manner. The response can contain one or more Credentials with the same Credential Configuration and Credential Dataset depending on the Credential Request:

- If the Credential Issuer is able to immediately issue the requested Credentials, it MUST respond with the HTTP status code 200 (see Section 15.3.3 of [RFC9110]).

- If the Credential Issuer is not able to immediately issue the requested credentials (e.g., due to a manual review process being required or the data used to issue the credential is not ready yet), the Credential Issuer MUST return a response with a `transaction_id` parameter. In this case, the Credential Issuer MUST also use the HTTP status code 202 for the response. The `transaction_id` MAY be used by the Client at a later time at the Deferred Credential endpoint.

If the Client requested an encrypted response by including the `credential_response_encryption` object in the request, the Credential Issuer MUST encode the information in the Credential Response as specified by Section 10, using the parameters from the `credential_response_encryption` object. Note that this is done regardless of the content.

If the Credential Response is not encrypted, the media type of the response MUST be set to `application/json`.

The following parameters are used in the JSON-encoded Credential Response body:

- `credentials`: OPTIONAL. Contains an array of one or more issued Credentials. It MUST NOT be used if the `transaction_id` parameter is present. The elements of the array MUST be objects. The number of elements in the `credentials` array matches the number of keys that the Wallet has provided via the `proofs` parameter of the Credential Request, unless the Issuer decides to issue fewer Credentials. Each key provided by the Wallet is used to bind to, at most, one Credential. This specification defines the following parameters to be used inside this object:
    - `credential`: REQUIRED. Contains one issued Credential. The encoding of the Credential depends on the Credential Format and MAY be a string or an object. Credential Formats expressed as binary data MUST be base64url-encoded and returned as a string. More details are defined in the Credential Format Profiles in Appendix A.
- `transaction_id`: OPTIONAL. String identifying a Deferred Issuance transaction. This parameter is contained in the response if the Credential Issuer cannot immediately issue the Credential. The value is subsequently used to obtain the respective Credential with the Deferred Credential Endpoint (see Section 9). It MUST not be used if the `credentials` parameter is present. It MUST be invalidated after the Credential for which it was meant has been obtained by the Wallet.
- `interval`: REQUIRED if `transaction_id` is present. Contains a positive number that represents the minimum amount of time in seconds that the Wallet SHOULD wait after receiving the response before sending a new request to the Deferred Credential Endpoint. It MUST NOT be used if the `credentials` parameter is present.
- `notification_id`: OPTIONAL. String identifying one or more Credentials issued in one Credential Response. It MUST be included in the Notification Request as defined in Section 11.1. It MUST not be used if the `credentials` parameter is not present.

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

Below is a non-normative example of a Credential Response in an immediate issuance flow for a Credential in JWT VC format (JSON encoded):

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

{
  "credentials": [
    {
      "credential": "LUpixVCWJk0eOt4CXQe1NXK....WZwmhmn9OQp6YxX0a2L"
    }
  ]
}
```

Below is a non-normative example of a Credential Response in an immediate issuance flow for multiple Credential instances in JWT VC format (JSON encoded) with an additional `notification_id` parameter:

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

{
  "credentials": [
    {
      "credential": "LUpixVCWJk0eOt4CXQe1NXK....WZwmhmn9OQp6YxX0a2L"
    },
    {
      "credential": "YXNkZnNhZGZkamZqZGFza23....29tZTIzMjMyMzIzMjMy"
    }
  ],
  "notification_id": "3fwe98js"
}
```

Below is a non-normative example of a Credential Response in a deferred flow:

```
HTTP/1.1 202 Accepted
Content-Type: application/json
Cache-Control: no-store

{
  "transaction_id": "8xLOxBtZp8",
  "interval" : 3600
}
```

8.3.1. Credential Error Response

When the Credential Request is invalid or unauthorized, the Credential Issuer constructs the error response as defined in this section.

8.3.1.1. Authorization Errors

If the Credential Request does not contain an Access Token that enables issuance of a requested Credential, the Credential Endpoint returns an authorization error response such as defined in Section 3 of [RFC6750].

8.3.1.2. Credential Request Errors

For errors related to the Credential Request's payload, such as issues with `type`, `format`, `proofs`, encryption parameters, or if the request is denied, the specific error codes from this section MUST be used instead of the generic `invalid_request` parameter defined in Section 3.1 of [RFC6750].

If the Wallet is requesting the issuance of a Credential that is not supported by the Credential Endpoint, the HTTP response MUST use the HTTP status code 400 (Bad Request) and set the content type to `application/json` with the following parameters in the JSON-encoded response body:

- `error`: REQUIRED. The `error` parameter SHOULD be a single ASCII [USASCII] error code from the following:
  - `invalid_credential_request`: The Credential Request is missing a required parameter, includes an unsupported parameter or parameter value, repeats the same parameter, or is otherwise malformed.
  - `unknown_credential_configuration`: Requested Credential Configuration is unknown.
  - `unknown_credential_identifier`: Requested Credential identifier is unknown.
  - `invalid_proof`: The `proofs` parameter in the Credential Request is invalid: (1) if the field is missing, or (2) one of the provided key proofs is invalid, or (3) if at least one of the key proofs does not contain a `c_nonce` value (refer to Section 7.2).
  - `invalid_nonce`: The `proofs` parameter in the Credential Request uses an invalid nonce: at least one of the key proofs contains an invalid `c_nonce` value. The wallet should retrieve a new `c_nonce` value (refer to Section 7).
  - `invalid_encryption_parameters`: This error occurs when the encryption parameters in the Credential Request are either invalid or missing. In the latter case, it indicates that the Credential Issuer requires the Credential Response to be sent encrypted, but the Credential Request does not contain the necessary encryption parameters.
  - `credential_request_denied`: The Credential Request has not been accepted by the Credential Issuer. The Wallet SHOULD treat this error as unrecoverable, meaning if received from a Credential Issuer the Credential cannot be issued.
- `error_description`: OPTIONAL. The `error_description` parameter MUST be a human-readable ASCII [USASCII] text, providing any additional information used to assist the Client implementers in understanding the occurred error. The values for the `error_description` parameter MUST NOT include characters outside the set `%x20-21 / %x23-5B / %x5D-7E`.

The usage of these parameters takes precedence over the `invalid_request` parameter defined in Section 8.3.1.1, since they provide more details about the errors.

Note that Credential Error Responses are never encrypted, even if a valid Credential Response would have been.

The following is a non-normative example of a Credential Error Response where an unsupported Credential Format was requested:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store

{
  "error": "unknown_credential_configuration"
}
```

9. Deferred Credential Endpoint

This endpoint is used to issue one or more Credentials previously requested at the Credential Endpoint in cases where the Credential Issuer was not able to immediately issue this Credential. Support for this endpoint is OPTIONAL.

The Wallet MUST present to the Deferred Endpoint an Access Token that is valid for the issuance of the Credential(s) previously requested at the Credential Endpoint.

Communication with the Deferred Credential Endpoint MUST utilize TLS.

9.1. Deferred Credential Request

The Deferred Credential Request is an HTTP POST request. The Deferred Credential Request MAY be encrypted (on top of TLS) using the `credential_request_encryption` parameter in Section 12.2 as specified in Section 10.

The following parameters are used in the Deferred Credential Request:

- `transaction_id`: REQUIRED. String identifying a Deferred Issuance transaction.
- `credential_response_encryption`: OPTIONAL. as defined in Section 8.2.

The Credential Issuer MUST invalidate the `transaction_id` after the Credential for which it was meant has been obtained by the Wallet.

Additional Deferred Credential Request parameters MAY be defined and used. The Credential Issuer MUST ignore any unrecognized parameters.

The Credential Issuer indicates support for encrypted requests by including the `credential_request_encryption` parameter in the Credential Issuer Metadata. The Client MAY encrypt the request when `encryption_required` is `false` and MUST do so when `encryption_required` is `true`.

When performing Deferred Credential Request encryption, the Client MUST encode the information in the Deferred Credential Request in a JWT as specified by Section 10, using the parameters from the `credential_request_encryption` object in the Credential Issuer Metadata.

If the Deferred Credential Request is not encrypted, the media type of the request MUST be set to `application/json`.

The following is a non-normative example of a Deferred Credential Request:

```
POST /deferred_credential HTTP/1.1
Host: server.example.com
Content-Type: application/json
Authorization: Bearer czZCaGRSa3F0MzpnWDFmQmF0M2JW

{
  "transaction_id": "8xLOxBtZp8"
}
```

The Credential Issuer indicates support for encrypted responses by including the `credential_response_encryption` parameter in the Credential Issuer Metadata. The Client MAY request encrypted responses by providing its encryption parameters in the Deferred Credential Request when `encryption_required` is `false` and MUST do so when `encryption_required` is `true`. Note that this object will be used for encrypting the response, regardless of what was sent in the initial Credential Request. If it is not included encryption will not be performed. Deferred Credential Request encryption MUST but used if the `credential_response_encryption` parameter is included, to prevent it being substituted by an attacker.

9.2. Deferred Credential Response

A Deferred Credential Response may either contain the requested Credentials or further defer the issuance:

- If the Credential Issuer is able to issue the requested Credentials, the Deferred Credential Response MUST use the `credentials` parameter as defined in Section 8.3 and MUST respond with the HTTP status code 200 (see Section 15.3.3 of [RFC9110]).
- If the Credential Issuer still requires more time, the Deferred Credential Response MUST use the `interval` and `transaction_id` parameters as defined in Section 8.3 and it MUST respond with the HTTP status code 202 (see Section 15.3.3 of [RFC9110]). The value of `transaction_id` MUST be same as the value of `transaction_id` in the Deferred Credential Request.

The Deferred Credential Response MAY use the `notification_id` parameter as defined in Section 8.3.

Additional Deferred Credential Response parameters MAY be defined and used. The Wallet MUST ignore any unrecognized parameters.

If the Client requested an encrypted response by including the `credential_response_encryption` object in the request, the Credential Issuer MUST encode the information in the Deferred Credential Response as specified by Section 10, using the parameters from the `credential_response_encryption` object. Note that this is done regardless of the content. The `credential_response_encryption` object may be different from the one included in the initial Credential Request so the Credential Issuer MUST use the newly provided one. This is to simplify key management in the case of longer deferred issuance.

If the Deferred Credential Response is not encrypted, the media type of the response MUST be set to `application/json`.

The following is a non-normative example of a Deferred Credential Response containing Credentials:

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

{
  "credentials": [
    {
      "credential": "LUpixVCWJk0eOt4CXQe1NXK....WZwmhmn9OQp6YxX0a2L"
    },
    {
      "credential": "YXNkZnNhZGZkamZqZGFza23....29tZTIzMjMyMzIzMjMy"
    }
  ],
  "notification_id": "3fwe98js"
}
```

The following is a non-normative example of a Deferred Credential Response, where the Credential Issuer still requires more time:

```
HTTP/1.1 202 OK
Content-Type: application/json

{
  "transaction_id": "8xLOxBtZp8",
  "interval": 86400
}
```

9.3. Deferred Credential Error Response

When the Deferred Credential Request is invalid, the Credential Issuer constructs the error response as defined in Section 8.3.1.

The following additional error code is specified in addition to those already defined in Section 8.3.1.2:

- `invalid_transaction_id`: The Deferred Credential Request contains an invalid `transaction_id`. This error occurs when the `transaction_id` was not issued by the respective Credential Issuer or it was already used to obtain a Credential.

This is a non-normative example of a Deferred Credential Error Response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store

{
  "error": "invalid_transaction_id"
}
```

In the event the Credential Issuer can no longer issue the credential(s), the `credential_request_denied` error code as defined in Section 8.3.1.2 should be used in response to a request. A wallet upon receiving this error SHOULD stop making requests to the deferred credential endpoint for the given `transaction_id`.

10. Encrypted Credential Requests and Responses

Encryption of requests and responses for the Credential and Deferred Credential Endpoints is performed as follows:

The contents of the message MUST be encoded as a JWT as described in [RFC7519]. The media type MUST be set to `application/jwt`.

The Public Key used to encrypt the message is selected based on the context. In the case where multiple public keys are available, any may be selected based on the information about each key, such as the `kty` (Key Type), `use` (Public Key Use), `alg` (Algorithm), and other JWK parameters. The `alg` parameter MUST be present. The JWE `alg` algorithm used MUST be equal to the `alg` value of the chosen JWK. If the selected public key contains a `kid` parameter, the JWE MUST include the same value in the `kid` JWE Header Parameter (as defined in Section 4.1.6) of the encrypted message. This enables the easy identification of the specific public key that was used to encrypt the message. The JWE `enc` content encryption algorithm used is obtained based on context.

If a `zip` (Compression Algorithm) value is specified, then compression is performed before encryption, as specified in [RFC7516]. If absent, no compression is performed.

When encryption of a message was required but the received message is unencrypted, it SHOULD be rejected.

For security considerations see Section 13.11

11. Notification Endpoint

This endpoint is used by the Wallet to notify the Credential Issuer of certain events for issued Credentials. These events enable the Credential Issuer to take subsequent actions after issuance. The Credential Issuer needs to return one `notification_id` parameter per Credential Response or Deferred Credential Response for the Wallet to be able to use this endpoint. Support for this endpoint is OPTIONAL. The Issuer cannot assume that a notification will be sent for every issued Credential since the use of this Endpoint is not mandatory for the Wallet.

The Wallet MUST present to the Notification Endpoint a valid Access Token issued at the Token Endpoint as defined in Section 6.

A Credential Issuer that requires a request to the Notification Endpoint MUST ensure the Access Token issued by the Authorization Server is valid at the Notification Endpoint.

The notification from the Wallet is idempotent. When the Credential Issuer receives multiple identical calls from the Wallet for the same `notification_id`, it returns success. Due to the network errors, there are no guarantees that a Credential Issuer will receive a notification within a certain time period or at all.

Communication with the Notification Endpoint MUST utilize TLS.

11.1. Notification Request

The Wallet sends an HTTP POST request to the Notification Endpoint with the following parameters in the entity-body and using the `application/json` media type. If the Wallet supports the Notification Endpoint, the Wallet MAY send one or more Notification Requests per `notification_id` value received.

- notification_id: REQUIRED. String received in the Credential Response or Deferred Credential Response identifying an issuance flow that contained one or more Credentials with the same Credential Configuration and Credential Dataset.
- event: REQUIRED. Type of the notification event. It MUST be a case sensitive string whose value is either credential_accepted, credential_failure, or credential_deleted. credential_accepted is to be used when the Credentials were successfully stored in the Wallet, with or without user action. credential_deleted is to be used when the unsuccessful Credential issuance was caused by a user action. In all other unsuccessful cases, credential_failure is to be used. Partial errors during issuance of multiple Credentials as a batch (e.g., one of the Credentials could not be stored) MUST be treated as the overall issuance flow failing.
- event_description: OPTIONAL. Human-readable ASCII [USASCII] text providing additional information, used to assist the Credential Issuer developer in understanding the event that occurred. Values for the event_description parameter MUST NOT include characters outside the set %x20-21 / %x23-5B / %x5D-7E.

Additional Notification Request parameters MAY be defined and used. The Credential Issuer MUST ignore any unrecognized parameters.

Below is a non-normative example of a Notification Request when a credential was successfully accepted by the End-User:

```
POST /notification HTTP/1.1
Host: server.example.com
Content-Type: application/json
Authorization: Bearer czZCaGRSa3F0MzpnWDFmQmF0M2JW

{
  "notification_id": "3fwe98js",
  "event": "credential_accepted"
}
```

Below is a non-normative example of a Notification Request when a Credential was deleted by the End-User:

```
POST /notification HTTP/1.1
Host: server.example.com
Content-Type: application/json
Authorization: Bearer czZCaGRSa3F0MzpnWDFmQmF0M2JW

{
  "notification_id": "3fwe98js",
  "event": "credential_failure",
  "event_description": "Could not store the Credential. Out of storage."
}
```

11.2. Successful Notification Response

When the Credential Issuer has successfully received the Notification Request from the Wallet, it MUST respond with an HTTP status code in the 2xx range. Use of the HTTP status code 204 (No Content) is RECOMMENDED.

Below is a non-normative example of a response to a successful Notification Request:

```
HTTP/1.1 204 No Content
```

11.3. Notification Error Response

If the Notification Request does not contain an Access Token or contains an invalid Access Token, the Notification Endpoint returns an Authorization Error Response, such as defined in Section 3 of [RFC6750].

When the `notification_id` value is invalid, the HTTP response MUST use the HTTP status code 400 (Bad Request) and set the content type to `application/json` with the following parameters in the JSON-encoded response body:

- `error`: REQUIRED. The value of the `error` parameter SHOULD be one of the following ASCII [USASCII] error codes:
    - `invalid_notification_id`: The `notification_id` in the Notification Request was invalid.
    - `invalid_notification_request`: The Notification Request is missing a required parameter, includes an unsupported parameter or parameter value, repeats the same parameter, or is otherwise malformed.

It is at the discretion of the Issuer to decide how to proceed after returning an error response.

The following is a non-normative example of a Notification Error Response when an invalid `notification_id` value was used:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store

{
  "error": "invalid_notification_id"
}
```

12. Metadata

12.1. Client Metadata

This specification defines the following new Client Metadata parameter in addition to those defined by [RFC7591] for Wallets acting as OAuth 2.0 Client:

- `credential_offer_endpoint`: OPTIONAL. Credential Offer Endpoint of a Wallet.

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

12.1.1. Client Metadata Retrieval

How to obtain Client Metadata is out of scope of this specification. Profiles of this specification MAY also define static sets of Client Metadata values to be used.

If the Credential Issuer is unable to perform discovery of the Wallet's Credential Offer Endpoint, the following custom URL scheme is used: `openid-credential-offer://`.

12.2. Credential Issuer Metadata

The Credential Issuer Metadata contains information on the Credential Issuer's technical capabilities, supported Credentials, and (internationalized) display information.

12.2.1. Credential Issuer Identifier

A Credential Issuer is identified by a case sensitive URL using the `https` scheme that contains scheme, host and, optionally, port number and path components, but no query or fragment components.

12.2.2. Credential Issuer Metadata Retrieval

The Credential Issuer's configuration can be retrieved using the Credential Issuer Identifier.

Credential Issuers publishing metadata MUST make a JSON document available at the path formed by inserting the string `/.well-known/openid-credential-issuer` into the Credential Issuer Identifier between the host component and the path component, if any.

For example, the metadata for the Credential Issuer Identifier `https://issuer.example.com/tenant` would be retrieved from `https://issuer.example.com/.well-known/openid-credential-issuer/tenant`. The metadata for the Credential Issuer Identifier `https://tenant.issuer.example.com` would be retrieved from `https://tenant.issuer.example.com/.well-known/openid-credential-issuer`.

Communication with the Credential Issuer Metadata Endpoint MUST utilize TLS.

To fetch the Credential Issuer Metadata, the Wallet MUST send an HTTP request using the GET method and the path formed following the steps above. The Wallet is RECOMMENDED to send an `Accept` header in the HTTP GET request to indicate the Content Type(s) it supports, and by doing so, signaling whether it supports signed metadata.

The Credential Issuer MUST respond with HTTP Status Code 200 and return the Credential Issuer Metadata containing the parameters defined in Section 12.2.4 as either

- an unsigned JSON document using the media type `application/json`, or
- a signed JSON Web Token (JWT) containing the Credential Issuer Metadata in its payload using the media type `application/jwt`.

The Credential Issuer MUST support returning metadata in an unsigned form 'application/json' and MAY support returning it in a signed form 'application/jwt'. In all cases the Credential Issuer MUST indicate the media type of the returned Metadata using the HTTP `Content-Type` header. It is RECOMMENDED for Credential Issuers to respond with a `Content-Type` matching to the Wallet's requested `Accept` header when the requested content type is supported.

The Wallet is RECOMMENDED to send an `Accept-Language` header in the HTTP GET request to indicate the language(s) preferred for display. It is up to the Credential Issuer whether to:

- send a subset the metadata containing internationalized display data for one or all of the requested languages and indicate returned languages using the HTTP `Content-Language` Header, or
- ignore the `Accept-Language` Header and send all supported languages or any chosen subset.

The language(s) in HTTP `Accept-Language` and `Content-Language` Headers MUST use the values defined in [RFC3066].

Below is a non-normative example of a Credential Issuer Metadata request:

```
GET /.well-known/openid-credential-issuer HTTP/1.1
Host: server.example.com
Accept-Language: fr-ch, fr;q=0.9, en;q=0.8, de;q=0.7, *;q=0.5
```

12.2.3. Signed Metadata

The signed metadata MUST be secured using a JSON Web Signature (JWS) [RFC7515] and contain the following elements:

- in the JOSE header,
  - `alg`: REQUIRED. A digital signature algorithm identifier such as per IANA "JSON Web Signature and Encryption Algorithms" registry [IANA.JOSE]. It MUST NOT be none or an identifier for a symmetric algorithm (MAC).
  - `typ`: REQUIRED. MUST be `openidvci-issuer-metadata+jwt`, which explicitly types the key proof JWT as recommended in Section 3.11 of [RFC8725].
- in the JWS payload,
  - `iss`: OPTIONAL. String denoting the party attesting to the claims in the signed metadata
  - `sub`: REQUIRED. String matching the Credential Issuer Identifier
  - `iat`: REQUIRED. Integer for the time at which the Credential Issuer Metadata was issued using the syntax defined in [RFC7519].
  - `exp`: OPTIONAL. Integer for the time at which the Credential Issuer Metadata is expiring, using the syntax defined in [RFC7519].

All metadata parameters used by the Credential Issuer MUST be added as top-level claims in the JWS payload.

When requesting signed metadata, the Wallet MUST establish trust in the signer of the metadata. Otherwise, the Wallet MUST reject the signed metadata. When validating the signature, the Wallet obtains the keys to validate the signature before processing the metadata, e.g., using JOSE header parameters like `x5c`, `kid` or `trust_chain` to convey the public key. The concrete mechanisms for doing this are out of scope of this specification.

See Appendix I.1 for a non-normative example of signed Credential Issuer Metadata.

12.2.4. Credential Issuer Metadata Parameters

This specification defines the following Credential Issuer Metadata parameters:

- `credential_issuer`: REQUIRED. The Credential Issuer's identifier, as defined in Section 12.2.1. The value MUST be identical to the Credential Issuer's identifier value into which the well-known URI string was inserted to create the URL used to retrieve the metadata. If these values are not identical (when compared using a simple string comparison with no normalization), the data contained in the response MUST NOT be used.
- `authorization_servers`: OPTIONAL. A non-empty array of strings, where each string is an identifier of the OAuth 2.0 Authorization Server (as defined in [RFC8414]) the Credential Issuer relies on for authorization. If this parameter is omitted, the entity providing the Credential Issuer is also acting as the Authorization Server, i.e., the Credential Issuer's identifier is used to obtain the Authorization Server metadata. The actual OAuth 2.0 Authorization Server metadata is obtained from the `oauth-authorization-server` well-known location as defined in Section 3 of [RFC8414]. When there are

multiple entries in the array, the Wallet may be able to determine which Authorization Server to use by querying the metadata; for example, by examining the `grant_types_supported` values, the Wallet can filter the server to use based on the grant type it plans to use. When the Wallet is using `authorization_server` parameter in the Credential Offer as a hint to determine which Authorization Server to use out of multiple, the Wallet MUST NOT proceed with the flow if the `authorization_server` Credential Offer parameter value does not match any of the entries in the `authorization_servers` array.

- `credential_endpoint`: REQUIRED. URL of the Credential Issuer's Credential Endpoint, as defined in Section 8.2. This URL MUST use the `https` scheme and MAY contain port, path, and query parameter components.
- `nonce_endpoint`: OPTIONAL. URL of the Credential Issuer's Nonce Endpoint, as defined in Section 7. This URL MUST use the `https` scheme and MAY contain port, path, and query parameter components. If omitted, the Credential Issuer does not require the use of `c_nonce`.
- `deferred_credential_endpoint`: OPTIONAL. URL of the Credential Issuer's Deferred Credential Endpoint, as defined in Section 9. This URL MUST use the `https` scheme and MAY contain port, path, and query parameter components. If omitted, the Credential Issuer does not support the Deferred Credential Endpoint.
- `notification_endpoint`: OPTIONAL. URL of the Credential Issuer's Notification Endpoint, as defined in Section 11. This URL MUST use the `https` scheme and MAY contain port, path, and query parameter components. If omitted, the Credential Issuer does not support the Notification Endpoint.
- `credential_request_encryption`: OPTIONAL. Object containing information about whether the Credential Issuer supports encryption of the Credential Request on top of TLS.
  - `jwks`: REQUIRED. A JSON Web Key Set, as defined in [RFC7591], that contains one or more public keys, to be used by the Wallet as an input to a key agreement for encryption of the Credential Request. Each JWK in the set MUST have a kid (Key ID) parameter that uniquely identifies the key.
  - `enc_values_supported`: REQUIRED. A non-empty array containing a list of the JWE [RFC7516] encryption algorithms (`enc` values) [RFC7518] supported by the Credential Endpoint to decode the Credential Request from a JWT [RFC7519].
  - `zip_values_supported`: OPTIONAL. A non-empty array containing a list of the JWE [RFC7516] compression algorithms (`zip` values) [RFC7518] supported by the Credential Endpoint to uncompress the Credential Request after decryption. If absent then no compression algorithms are supported. The Wallet may use any of the supported compression algorithm to compress the Credential Request prior to encryption.
  - `encryption_required`: REQUIRED. Boolean value specifying whether the Credential Issuer requires the additional encryption on top of TLS for the Credential Requests. If the value is `true`, the Credential Issuer requires encryption for every Credential Request. If the value is `false`, the Wallet MAY choose whether it encrypts the request or not.
- `credential_response_encryption`: OPTIONAL. Object containing information about whether the Credential Issuer supports encryption of the Credential Response on top of TLS.
  - `alg_values_supported`: REQUIRED. A non-empty array containing a list of the JWE [RFC7516] encryption algorithms (`alg` values) [RFC7518] supported by the Credential Endpoint to encode the Credential Response in a JWT [RFC7519].
  - `enc_values_supported`: REQUIRED. A non-empty array containing a list of the JWE [RFC7516] encryption algorithms (`enc` values) [RFC7518] supported by the Credential Endpoint to encode the Credential Response in a JWT [RFC7519].
  - `zip_values_supported`: OPTIONAL. A non-empty array containing a list of the JWE [RFC7516] compression algorithms (`zip` values) [RFC7518] supported by the Credential Endpoint to compress the Credential Response prior to encryption. If absent then compression is not supported.

  - `encryption_required`: REQUIRED. Boolean value specifying whether the Credential Issuer requires the additional encryption on top of TLS for the Credential Response. If the value is `true`, the Credential Issuer requires encryption for every Credential Response and therefore the Wallet MUST provide encryption keys in the Credential Request. If the value is `false`, the Wallet MAY choose whether it provides encryption keys or not.
- `batch_credential_issuance`: OPTIONAL. Object containing information about the Credential Issuer's support for issuance of multiple Credentials in a batch in the Credential Endpoint. The presence of this parameter means that the issuer supports more than one key proof in the `proofs` parameter in the Credential Request so can issue more than one Verifiable Credential for the same Credential Dataset in a single request/response.
  - `batch_size`: REQUIRED. Integer value specifying the maximum array size for the `proofs` parameter in a Credential Request. It MUST be 2 or greater.
- `display`: OPTIONAL. A non-empty array of objects, where each object contains display properties of a Credential Issuer for a certain language. Below is a non-exhaustive list of valid parameters that MAY be included:
  - `name`: OPTIONAL. String value of a display name for the Credential Issuer.
  - `locale`: OPTIONAL. String value that identifies the language of this object represented as a language tag taken from values defined in BCP47 [RFC5646]. There MUST be only one object for each language identifier.
  - `logo`: OPTIONAL. Object with information about the logo of the Credential Issuer. Below is a non-exhaustive list of parameters that MAY be included:
    - `uri`: REQUIRED. String value that contains a URI where the Wallet can obtain the logo of the Credential Issuer. The Wallet needs to determine the scheme, since the URI value could use the `https:` scheme, the `data:` scheme, etc.
    - `alt_text`: OPTIONAL. String value of the alternative text for the logo image.
- `credential_configurations_supported`: REQUIRED. Object that describes specifics of the Credential that the Credential Issuer supports issuance of. This object contains a list of name/value pairs, where each name is a unique identifier of the supported Credential being described. This identifier is used in the Credential Offer as defined in Section 4.1.1 to communicate to the Wallet which Credential is being offered. The value is an object that contains metadata about a specific Credential and contains the following parameters defined by this specification:
  - `format`: REQUIRED. A JSON string identifying the format of this Credential, i.e., `jwt_vc_json` or `ldp_vc`. Depending on the format value, the object contains further elements defining the type and (optionally) particular claims the Credential MAY contain and information about how to display the Credential. Appendix A contains Credential Format Profiles introduced by this specification.
  - `scope`: OPTIONAL. A JSON string identifying the scope value that this Credential Issuer supports for this particular Credential. The value can be the same across multiple `credential_configurations_supported` objects. The Authorization Server MUST be able to uniquely identify the Credential Issuer based on the scope value. The Wallet can use this value in the Authorization Request as defined in Section 5.1.2. Scope values in this Credential Issuer metadata MAY duplicate those in the `scopes_supported` parameter of the Authorization Server. If `scope` is absent, the only way to request the Credential is using `authorization_details` [RFC9396] - in this case, the OAuth Authorization Server metadata for one of the Authorization Servers found from the Credential Issuer's Metadata must contain an `authorization_details_types_supported` that contains `openid_credential`.
  - `credential_signing_alg_values_supported`: OPTIONAL. A non-empty array of algorithm identifiers that identify the algorithms that the Issuer uses to sign the issued Credential. Algorithm

identifier types and values used are determined by the Credential Format and are defined in Appendix A.

- `cryptographic_binding_methods_supported`: OPTIONAL. A non-empty array of case sensitive strings that identify the representation of the cryptographic key material that the issued Credential is bound to, as defined in Section 8.1. It MUST be present when Cryptographic Key Binding is required for a Credential, and omitted otherwise. If absent, Cryptographic Key Binding is not required for this credential. Support for keys in JWK format [RFC7517] is indicated by the value `jwk`. Support for keys expressed as a COSE Key object [RFC8152] (for example, used in [ISO.18013-5]) is indicated by the value `cose_key`. When the Cryptographic Key Binding method is a DID, valid values are a `did:` prefix followed by a method-name using a syntax as defined in Section 3.1 of [DID-Core], but without a `:` and method-specific-id. For example, support for the DID method with a method-name "example" would be represented by `did:example`.
- `proof_types_supported`: OPTIONAL. Object that describes specifics of the key proof(s) that the Credential Issuer supports. It MUST be present if `cryptographic_binding_methods_supported` is present, and omitted otherwise. If absent, the Wallet is not required to supply proofs when requesting this credential. This object contains a list of name/value pairs, where each name is a unique identifier of the supported proof type(s). Valid values are defined in Appendix F, other values MAY be used. The Wallet also uses this identifier in the Credential Request as defined in Section 8.2. The value in the name/value pair is an object that contains metadata about the key proof and contains the following parameters defined by this specification:
  - `proof_signing_alg_values_supported`: REQUIRED. A non-empty array of algorithm identifiers that the Issuer supports for this proof type. The Wallet uses one of them to sign the proof. Algorithm identifier types and values used are determined by the proof type and are defined in Appendix F.
  - `key_attestations_required`: OPTIONAL. Object that describes the requirement for key attestations as described in Appendix D, which the Credential Issuer expects the Wallet to send within the proof(s) of the Credential Request. If the Credential Issuer does not require a key attestation, this parameter MUST NOT be present in the metadata. If both `key_storage` and `user_authentication` parameters are absent, the `key_attestations_required` parameter may be empty, indicating a key attestation is needed without additional constraints.
    - `key_storage`: OPTIONAL. A non-empty array defining values specified in Appendix D.2 accepted by the Credential Issuer.
    - `user_authentication`: OPTIONAL. A non-empty array defining values specified in Appendix D.2 accepted by the Credential Issuer.
- `credential_metadata`: OPTIONAL. Object containing information relevant to the usage and display of issued Credentials. Credential Format-specific mechanisms can overwrite the information in this object to convey Credential metadata. Format-specific mechanisms, such as SD-JWT VC display metadata are always preferred by the Wallet over the information in this object, which serves as the default fallback. Below is a non-exhaustive list of parameters that MAY be included:
  - `display`: OPTIONAL. A non-empty array of objects, where each object contains the display properties of the supported Credential for a certain language. Below is a non-exhaustive list of parameters that MAY be included.
    - `name`: REQUIRED. String value of a display name for the Credential.
    - `locale`: OPTIONAL. String value that identifies the language of this object represented as a language tag taken from values defined in BCP47 [RFC5646]. Multiple `display` objects MAY be included for separate languages. There MUST be only one object for each language identifier.

        - `logo`: OPTIONAL. Object with information about the logo of the Credential. The following non-exhaustive set of parameters MAY be included:
            - `uri`: REQUIRED. String value that contains a URI where the Wallet can obtain the logo of the Credential from the Credential Issuer. The Wallet needs to determine the scheme, since the URI value could use the `https:` scheme, the `data:` scheme, etc.
            - `alt_text`: OPTIONAL. String value of the alternative text for the logo image.
        - `description`: OPTIONAL. String value of a description of the Credential.
        - `background_color`: OPTIONAL. String value of a background color of the Credential represented as numerical color values defined in CSS Color Module Level 3 [CSS-Color].
        - `background_image`: OPTIONAL. Object with information about the background image of the Credential. At least the following parameter MUST be included:
            - `uri`: REQUIRED. String value that contains a URI where the Wallet can obtain the background image of the Credential from the Credential Issuer. The Wallet needs to determine the scheme, since the URI value could use the `https:` scheme, the `data:` scheme, etc.
        - `text_color`: OPTIONAL. String value of a text color of the Credential represented as numerical color values defined in CSS Color Module Level 3 [CSS-Color].
    - `claims`: OPTIONAL. A non-empty array of claims description objects as defined in Appendix B.2.

An Authorization Server that only supports the Pre-Authorized Code grant type MAY omit the `response_types_supported` parameter in its metadata despite [RFC8414] mandating it.

Note: It can be challenging for a Credential Issuer that accepts tokens from multiple Authorization Servers to introspect an Access Token to check the validity and determine the permissions granted. Some ways to achieve this are relying on Authorization Servers that use [RFC9068] or by the Credential Issuer understanding the proprietary Access Token structures of the Authorization Servers.

Depending on the Credential Format, additional parameters might be present in the `credential_configurations_supported` object values, such as information about claims in the Credential. For Credential Format-specific claims, see the "Credential Issuer Metadata" subsections in Appendix A.

The Authorization Server MUST be able to determine from the Issuer metadata what claims are disclosed by the requested Credentials to be able to render meaningful End-User consent.

Additional Credential Issuer metadata parameters MAY be defined and used. The Wallet MUST ignore any unrecognized parameters.

The following is a non-normative example of Credential Issuer metadata of a Credential in the IETF SD-JWT VC [I-D.ietf-oauth-sd-jwt-vc] format:

```
{
  "credential_configurations_supported": {
    "SD_JWT_VC_example_in_OpenID4VCI": {
      "format": "dc+sd-jwt",
      "scope": "SD_JWT_VC_example_in_OpenID4VCI",
      "cryptographic_binding_methods_supported": [
        "jwk"
      ],
      "credential_signing_alg_values_supported": [
        "ES256"
      ],
      "proof_types_supported": {
        "jwt": {
          "proof_signing_alg_values_supported": [
            "ES256"
          ],
          "key_attestations_required": {
            "key_storage": [ "iso_18045_moderate" ],
            "user_authentication": [ "iso_18045_moderate" ]
          }
        }
      },
      "vct": "SD_JWT_VC_example_in_OpenID4VCI",
      "credential_metadata": {
        "display": [
          {
            "name": "IdentityCredential",
            "logo": {
              "uri": "https://university.example.edu/public/logo.png",
              "alt_text": "a square logo of a university"
            },
            "locale": "en-US",
            "background_color": "#12107c",
            "text_color": "#FFFFFF"
          }
        ],
        "claims": [
          {
            "path": ["given_name"],
            "display": [
              {
                "name": "Given Name",
                "locale": "en-US"
              },
              {
                "name": "Vorname",
                "locale": "de-DE"
              }
            ]
          },
          {
            "path": ["family_name"],
            "display": [
              {
                "name": "Surname",
                "locale": "en-US"
              },
              {
                "name": "Nachname",
                "locale": "de-DE"
              }
            ]
          },
          {"path": ["email"]},
          {"path": ["phone_number"]},
          {
            "path": ["address"],
            "display": [
              {
                "name": "Place of residence",
                "locale": "en-US"
              },
              {
                  "name": "Wohnsitz",
                  "locale": "de-DE"
                }
              ]
            },
            {"path": ["address", "street_address"]},
            {"path": ["address", "locality"]},
            {"path": ["address", "region"]},
            {"path": ["address", "country"]},
            {"path": ["birthdate"]},
            {"path": ["is_over_18"]},
            {"path": ["is_over_21"]},
            {"path": ["is_over_65"]}
          ]
        }
      }
    }
  }
```

Note: The Client MAY use other mechanisms to obtain information about the Verifiable Credentials that a Credential Issuer can issue.

See Appendix I.1 for additional examples of Credential Issuer Metadata.

12.3. OAuth 2.0 Authorization Server Metadata

This specification also defines a new OAuth 2.0 Authorization Server metadata [RFC8414] parameter to publish whether the Authorization Server that the Credential Issuer relies on for authorization supports anonymous Token Requests with the Pre-Authorized Grant Type. It is defined as follows:

- `pre-authorized_grant_anonymous_access_supported`: OPTIONAL. A boolean indicating whether the Credential Issuer accepts a Token Request with a Pre-Authorized Code but without a `client_id`. The default is `false`.

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

13. Security Considerations

13.1. Formal Security Analysis

The security properties of some features in a previous revision of this specification have been formally analyzed, see [secanalysis].

13.2. Best Current Practice for OAuth 2.0 Security

Implementers of this specification SHOULD follow the Best Current Practice for OAuth 2.0 Security given in [BCP240]. It is RECOMMENDED that implementers do this by following the [FAPI2_Security_Profile] where it is applicable.

The parts of [FAPI2_Security_Profile] that may not be applicable when using this specification are:

1. Client authentication: The private_key_jwt and MTLS client authentication methods may not be practical to implement in a secure way for native app Wallets. The use of Wallet Attestations as defined in Appendix E is RECOMMENDED instead.

2. Sender-constrained access tokens: MTLS sender constrained access token may not be practical to implement in a secure way for native app Wallets. The use of DPoP [RFC9449] is RECOMMENDED.

13.3. Trust between Wallet and Issuer

Credential Issuers often want to know what Wallet they are issuing Credentials to and how private keys are managed for the following reasons:

- The Credential Issuer MAY want to ensure that private keys are properly protected from exfiltration and replay to prevent an adversary from impersonating the legitimate Credential Holder by presenting their Credentials.
- The Credential Issuer MAY also want to ensure that the Wallet managing the Credentials adheres to certain policies and, potentially, was audited and approved under a certain regulatory and/or commercial scheme.

The following mechanisms in concert can be utilized to fulfill those objectives:

Key attestation is a mechanism where the key storage component or Wallet Provider asserts the cryptographic public keys and their security policy. The Wallet MAY provide this data in the Credential Request to allow the Credential Issuer to validate the cryptographic key management policy. This requires the Credential Issuer to rely on the trust anchor of the key attestation and the respective key management policy. While some existing platforms have key attestation formats, this specification introduces a common key attestation format that may be used by Credential Issuers for improved interoperability, see Appendix D.

Client Authentication enables a Wallet to authenticate with the Credential Issuer's Authorization Server. This authentication ensures that the Wallet complies with the Authorization Server's security, compliance, and governance standards, which may be required by trust frameworks, regulatory requirements, laws, or internal policies. Any method listed in the OAuth Token Endpoint Authentication Methods IANA Registry can be used for authentication during the Pushed Authorization or Token Request. The Authorization Server SHOULD specify its client authentication requirements using the `token_endpoint_auth_method` metadata parameter.

Wallet Attestation is a signed proof provided by the Wallet Provider, verifying the client's authenticity and genuineness. This process uses the mechanisms outlined in Attestation-Based Client Authentication, described in the Section Wallet Attestation. Once obtained, the Wallet Attestation can be used as a client authentication for the Wallet.

13.4. Split-Architecture Wallets

A Wallet may consist of multiple components with varying levels of trust, security, and privacy. A common example of this is an architecture that involves both a server-side component and a native application. While the server component can offer advantages such as enhanced reliability, centralized security controls, and certain privacy protections, it also introduces distinct risks. These include increased difficulty in conducting audits (particularly by external security experts), the potential for opaque or unverified updates, and a higher susceptibility to insider threats.

To ensure the server component can provide meaningful functionality while preserving user privacy, the principle of data minimization is encouraged to be employed. In particular, ensure the confidentiality of user data, it is best to apply application level encryption to the Credential Request/Response, from the device through the server component.

It is important to note that when the server component acts as the trust anchor (e.g., for Wallet Attestations or Key Attestations), it cannot also serve as a safeguard against itself.

If the server component has access to authorization codes, pre-authorization codes, or other sensitive tokens or proofs, and no additional mitigations are implemented beyond those outlined here, it MAY be able to impersonate the application and gain access to confidential data. In cases where the server component is not fully trusted, these sensitive elements MUST NOT be passed to or routed through it.

13.5. Credential Offer

The Wallet MUST consider the parameter values in the Credential Offer as not trustworthy, since the origin is not authenticated and the message integrity is not protected. The Credential Issuer is not considered trustworthy just because it sent the Credential Offer. Therefore, the Wallet MUST perform the same validation checks on the Credential Issuer as it would when initiating the flow directly from the Wallet. An attacker might attempt to use a Credential Offer to conduct a phishing or injection attack.

The Wallet MUST NOT accept Credentials just because this mechanism was used. All protocol steps defined in this specification MUST be performed in the same way as if the Wallet would have started the flow.

The Credential Issuer MUST ensure the release of any privacy-sensitive data in Credential Offer is legal.

13.6. Pre-Authorized Code Flow

13.6.1. Replay Prevention

The Pre-Authorized Code Flow is vulnerable to the replay of the Pre-Authorized Code, because by design, it is not bound to a certain session (as the Authorization Code Flow does with PKCE). This means an attacker can replay the Pre-Authorized Code meant for a victim at another device. In a 'shoulder surfing' scenario, the attacker might scan the QR code while it is displayed on the victim's screen, and thereby get access to the Credential. As the Pre-Authorized Code is presented as a link or QR code, either may be shared beyond its intended use, allowing others to replay the transaction. Such replay attacks must be prevented using other means. The design facilitates the following options:

- Transaction Code: the Credential Issuer might set up a Transaction Code with the End-User (e.g., via text message or email) that needs to be presented in the Token Request.

13.6.2. Transaction Code Phishing

An attacker might leverage the Credential issuance process and the End-User's trust in the Wallet to phish Transaction Codes sent out by a different service that grant the attacker access to services other than Credential issuance. The attacker could set up a Credential Issuer site and in parallel to the issuance request, trigger transmission of a Transaction Code to the End-User's phone from a service other than Credential issuance, e.g., from a payment service. The End-User would then be asked to enter this Transaction Code into the Wallet and since the Wallet sends this Transaction Code to the Token Endpoint of the Credential Issuer (the attacker), the attacker would get access to the Transaction Code, and access to that other service.

In order to cope with that issue, the Wallet is RECOMMENDED to interact with trusted Credential Issuers only. In that case, the Wallet would not process a Credential Offer with an untrusted issuer URL. The Wallet MAY also show the End-User the endpoint of the Credential Issuer it will be sending the Transaction Code to and ask the End-User for confirmation.

13.7. Credential Lifecycle Management

The Credential Issuer is supposed to be responsible for the lifecycle of its Credentials. This means the Credential Issuer will invalidate Credentials when it deems appropriate, e.g., if it detects fraudulent behavior.

The Wallet is supposed to detect signs of fraudulent behavior related to the Credential management in the Wallet (e.g., device rooting) and to act upon such signals. Options include Credential revocation at the Credential Issuer and/or invalidation of the key material used to cryptographically bind the Credential to the identifier of the End-User possessing that Credential.

13.8. Proof replay

If an adversary obtains a key proof (as outlined in Appendix F), they could potentially replay that proof to a Credential Issuer to obtain a duplicate Credential or multiple duplicates issued to the same key. The `c_nonce` parameter serves as the main defense against this, by providing Credential Issuers with a mechanism to determine freshness of a generated proof and thus reject a proof when required. It is RECOMMENDED that Credential Issuers utilize the Nonce Endpoint as specified in Section 7 to protect against this. A Wallet can continue using a given nonce until it is rejected by the Credential Issuer. The Credential Issuer determines for how long a particular nonce can be used.

Note: For the attacker to be able to present a Credential bound to a replayed key proof to the Verifier, the attacker also needs to obtain the victim's private key. To limit this, Credential Issuers are RECOMMENDED to check how the Wallet protects the private keys, using mechanisms defined in Appendix D.

Note: To accommodate for clock offsets, the Credential Issuer server MAY accept proofs that carry an `iat` time in the reasonably near future (on the order of seconds or minutes). Because clock skews between servers and Clients may be large, servers MAY limit key proof lifetimes by using server-provided nonce values containing the time at the server rather than comparing the client-supplied `iat` time to the time at the server. Nonces created in this way yield the same result even in the face of arbitrarily large clock skews.

13.9. TLS Requirements

Implementations MUST follow [BCP195]. Whenever TLS is used, a TLS server certificate check MUST be performed, per [RFC6125].

13.10. Protecting the Access Token

Access Tokens represent End-User authorization and consent to issue certain Credential(s). Long-lived Access Tokens giving access to Credentials MUST not be issued unless sender-constrained. Access Tokens with lifetimes longer than 5 minutes are, in general, considered long lived.

To sender-constrain Access Tokens, see the recommendations in Section 13.2. If Bearer Access Tokens are stored by the Wallet, they MUST be stored in a secure manner, for example, encrypted using a key stored in a protected key store.

13.11. Application-Layer Encryption

Depending on the architecture of the Wallet and the Issuer, adding encryption to requests and responses, beyond transport-level security (TLS), can offer enhanced data confidentiality. This can come at the cost of added complexity. This approach can be particularly relevant when the Wallet consists of multiple components with varying trust levels, such as a backend server and a client application. Similar considerations may apply to more complex Issuer architectures.

It is important that the Wallet component responsible for encryption can establish trust in the Issuer's key material to prevent man-in-the-middle attacks. The simplest way to achieve this is by retrieving the keys directly from the Issuer’s hosted Issuer Metadata. Alternatively, in the case of signed Issuer Metadata, the signature can be verified to ensure authenticity.

In cases where the application-layer encryption begins and terminates in the same component as TLS, it provides no additional protection.

While application-layer encryption can enhance the confidentiality of data in transit, it does not protect against other threats, such as access token theft, client impersonation, or other forms of unauthorized access. These risks must be mitigated through additional security measures.

14. Implementation Considerations

14.1. Claims-based Holder Binding of the Credential to the End-User possessing the Credential

Credentials not cryptographically bound to the identifier of the End-User possessing it (see Section 8.1), should be bound to the End-User possessing the Credential, based on the claims included in the Credential.

In Claims-based Holder Binding, no Cryptographic Key Binding material is provided. Instead, the issued Credential includes End-User claims that can be used by the Verifier to verify possession of the Credential by requesting presentation of existing forms of physical or digital identification that includes the same claims (e.g., a driving license or other ID cards in person, or an online ID verification service).

14.2. Binding of the Credential without Cryptographic Key Binding or Claims-based Holder Binding

Some Credential Issuers might choose to issue bearer Credentials without either Cryptographic Key Binding or Claims-based Holder Binding because they are meant to be presented without proof of possession.

One such use case is low assurance Credentials, such as coupons or tickets.

Another use case is when the Credential Issuer uses cryptographic schemes that can provide binding to the End-User possessing that Credential without explicit cryptographic material being supplied by the application used by that End-User. For example, in the case of the BBS Signature Scheme, the issued Credential itself is a secret and only a derivation from the Credential is presented to the Verifier. Effectively, the Credential is bound to the Credential Issuer's signature on the Credential, which becomes a shared secret transferred from the Credential Issuer to the End-User.

14.3. Multiple Accesses to the Credential Endpoint

The Credential Endpoint can be accessed multiple times by a Wallet using the same Access Token, even for the same Credential. The Credential Issuer determines if the subsequent successful requests will return the same or an updated Credential, such as having a new expiration time or using the most current End-User claims.

The Credential Issuer MAY also decide to no longer accept the Access Token and a re-authentication or Token Refresh (see [RFC6749], Section 6) MAY be required at the Credential Issuer's discretion. The policies between the Credential Endpoint and the Authorization Server that MAY change the behavior of what is returned with a new Access Token are beyond the scope of this specification (see Section 7 of [RFC6749]).

The Credential Issuer SHOULD NOT revoke previously issued, valid Credentials solely as a result of a subsequent successful Credential Request. This, for example, ensures that the Wallet can keep a desired number of Credentials without causing additional revocation and issuance overhead.

The action leading to the Wallet performing another Credential Request can also be triggered by a background process, or by the Credential Issuer using an out-of-band mechanism (SMS, email, etc.) to inform the End-User.

14.4. Relationship between the Credential Issuer Identifier in the Metadata and the Issuer Identifier in the Issued Credential

The Credential Issuer Identifier is always a URL using the `https` scheme, as defined in Section 12.2.1. Depending on the Credential Format, the Issuer Identifier in the issued Credential may not be a URL using the `https` scheme. Some other forms that it can take are a DID included in the `issuer` property in a [VC_DATA] format, or the `Subject` value of the document signer certificate included in the `x5chain` element in an [ISO.18013-5] format.

When the Issuer Identifier in the issued Credential is a DID, a non-exhaustive list of mechanisms the Credential Issuer MAY use to bind to the Credential Issuer Identifier is as follows:

1. Use the [DIF.Well-Known_DID] Specification to provide binding between a DID and a certain domain.
2. If the Issuer Identifier in the issued Credential is an object, add to the object a `credential_issuer` claim, as defined in Section 12.2.1.

The Wallet MAY check the binding between the Credential Issuer Identifier and the Issuer Identifier in the issued Credential.

14.5. Refreshing Issued Credentials

After a Verifiable Credential has been issued to the Holder, claim values about the subject of a Credential or a signature on the Credential may need to be updated. There are two possible mechanisms to do so.

First, the Wallet may receive an updated version of a Credential from a Credential Endpoint using a valid Access Token. This does not involve interaction with the End-User. If the Credential Issuer issued a Refresh Token to the Wallet, the Wallet would obtain a fresh Access Token by making a request to the Token Endpoint, as defined in Section 6 of [RFC6749].

Second, the Credential Issuer can reissue the Credential by starting the issuance process from the beginning. This would involve interaction with the End-User. A Credential needs to be reissued if the Wallet does not have a valid Access Token or a valid Refresh Token. With this approach, when a new Credential is issued, the Wallet might need to check if it already has a Credential of the same type and, if necessary, delete the old Credential. Otherwise, the Wallet might end up with more than one Credential of the same type, without knowing which one is the latest.

Credential Refresh can be initiated by the Wallet independently from the Credential Issuer, or the Credential Issuer can send a signal to the Wallet asking it to request Credential refresh. How the Credential Issuer sends such a signal is out of scope of this specification.

It is up to the Credential Issuer whether to update both the signature and the claim values, or only the signature.

14.6. Batch Issuing Credentials

The Credential Issuer determines the number of the Credentials issued in the Credential Response, regardless of number of proofs/keys contained in the `proofs` parameter in the Credential Request.

14.7. Pre-Final Specifications

Implementers should be aware that this specification uses several specifications that are not yet final specifications. Those specifications are:

- OpenID Federation 1.0 draft -43 [OpenID.Federation]
- SD-JWT-based Verifiable Credentials (SD-JWT VC) draft -11 [I-D.ietf-oauth-sd-jwt-vc]
- Attestation-Based Client Authentication draft -07 [I-D.ietf-oauth-attestation-based-client-auth]
- Token Status List draft -12 [I-D.ietf-oauth-status-list]

While breaking changes to the specifications referenced in this specification are not expected, should they occur, OpenID4VCI implementations should continue to use the specifically referenced versions above in preference to the final versions, unless updated by a profile or new version of this specification.

15. Privacy Considerations

When [RFC9396] is used, the Privacy Considerations of that specification also apply.

The privacy principles of [ISO.29100] should be adhered to.

15.1. User Consent

The Credential Issuer SHOULD obtain the End-User's consent before issuing Credential(s) to the Wallet. It SHOULD be made clear to the End-User what information is being included in the Credential(s) and for what purpose.

15.2. Minimum Disclosure

To ensure minimum disclosure and prevent Verifiers from obtaining claims unnecessary for the transaction at hand, when issuing Credentials that are intended to be created once and then used a number of times by the End-User, the Credential Issuers and the Wallets SHOULD implement Credential Formats that support selective disclosure, or consider issuing a separate Credential for each user claim.

15.3. Storage of the Credentials

To prevent a leak of End-User data, especially when it is signed, which risks revealing private data of End-Users to third parties, systems implementing this specification SHOULD be designed to minimize the amount of End-User data that is stored. All involved parties SHOULD store Verifiable Credentials containing privacy-sensitive data only for as long as needed, including in log files. Any logging of End-User data should be carefully considered as to whether it is necessary at all. The time logs are retained for should be minimized.

After Issuance, Credential Issuers SHOULD NOT store the Issuer-signed Credentials if they contain privacy-sensitive data. Wallets SHOULD store Credentials only in encrypted form, and, wherever possible, use hardware-backed encryption. Wallets SHOULD not store Credentials longer than needed.

15.4. Correlation

15.4.1. Unique Values Encoded in the Credential

Issuance/presentation or two presentation sessions by the same End-User can be linked on the basis of unique values encoded in the Credential (End-User claims, identifiers, Issuer signature, etc.) either by colluding Issuer/Verifier or Verifier/Verifier pairs, or by the same Verifier.

To prevent these types of correlation, Credential Issuers and Wallets SHOULD use methods, including but not limited to the following ones:

- Issue a batch of Credentials with the same Credential Dataset to facilitate the use of a unique Credential per presentation or per Verifier. This approach solely aids in achieving Verifier-to-Verifier unlinkability.
- Use cryptographic schemes that can provide non-correlation.

Claims containing time-related information, such as issuance or expiration dates, SHOULD be either individually randomized within an appropriate time window (e.g., within the last 24 hours), or rounded (e.g., to the start of the day), to avoid unintended correlation factors.

Credential Issuers specifically SHOULD discard values that can be used in collusion with a Verifier to track a user, such as the Issuer's signature or cryptographic key material to which an issued credential was bound to.

15.4.2. Credential Offer

The Privacy Considerations in Section 11.2 of [RFC9101] apply to the `credential_offer` and `credential_offer_uri` parameters defined in Section 4.1.

15.4.3. Authorization Request

The Wallet SHOULD NOT include potentially sensitive information in the Authorization Request, for example, by including clear-text session information as a `state` parameter value or encoding it in a `redirect_uri` parameter. A third party may observe such information through browser history, etc. and correlate the user's activity using it.

15.4.4. Wallet Attestation Subject

The Wallet Attestation as defined in Appendix E SHOULD NOT introduce a unique identifier specific to a single client. The subject claim for the Wallet Attestation SHOULD be a value that is shared by all Wallet instances using this type of wallet implementation. The value should be understood as an identifier of the Wallet type, rather than the specific Wallet instance itself.

15.5. Identifying the Credential Issuer

Information in the credential identifying a particular Credential Issuer, such as a Credential Issuer Identifier, issuer's certificate, or issuer's public key may reveal information about the End-User.

For example, when a military organization or a drug rehabilitation center issues a vaccine credential, verifiers can deduce that the owner of the Wallet storing such a Credential is a military member or may have a substance use disorder.

In addition, when a Credential Issuer issues only one type of Credential, it might have privacy implications, because if the Wallet has a Credential issued by that Issuer, its type and claim names can be determined.

For example, if the National Cancer Institute only issued Credentials with cancer registry information, it is possible to deduce that the owner of the Wallet storing such a Credential is a cancer patient.

To mitigate these issues, a group of organizations may elect to use a common Credential Issuer, such that any credentials issued by this Issuer cannot be attributed to a particular organization through identifiers of the Credential Issuers alone. A group signature scheme may also be used instead of an individual signature.

When a common Credential Issuer is used, appropriate guardrails need to be in place to prevent one organization from issuing illegitimate credentials on behalf of other organizations.

15.6. Identifying the Wallet

There is a potential for leaking information about the Wallet to third parties when the Wallet reacts to a Credential Offer. An attacker may send Credential Offers using different custom URL schemes or claimed https urls, see if the Wallet reacts (e.g., whether the wallet retrieves Credential Issuer metadata hosted by an attacker's server), and, therefore, learn which Wallet is installed. To avoid this, the Wallet SHOULD require user interaction or establish trust in the Issuer before fetching any `credential_offer_uri` or acting on the received Credential Offer.

15.7. Untrusted Wallets

The Wallet transmits and stores sensitive information about the End-User. To ensure that the Wallet can handle those appropriately (i.e., according to a certain trust framework or a regulation), the Credential Issuer should properly authenticate the Wallet and ensure it is a trusted entity. For more details, see Section 13.3.

16. Normative References

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

[CSS-Color] Çelik, T., Lilley, C., and D. Baron, "CSS Color Module Level 3", 18 January 2022, <https://www.w3.org/TR/2022/REC-css-color-3-20220118/>.

[DID-Core] Sporny, M., Guy, A., Sabadello, M., and D. Reed, "Decentralized Identifiers (DIDs) v1.0", 19 July 2022, <https://www.w3.org/TR/2022/REC-did-core-20220719/>.

[FAPI2_Security_Profile] Fett, D., Tonge, D., and J. Heenan, "FAPI 2.0 Security Profile", 22 February 2025, <https://openid.net/specs/fapi-security-profile-2_0.html>.

[I-D.ietf-jose-fully-specified-algorithms] Jones, M. B. and O. Steele, "Fully-Specified Algorithms for JOSE and COSE", Work in Progress, Internet-Draft, draft-ietf-jose-fully-specified-algorithms-13, 11 May 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-jose-fully-specified-algorithms-13>.

[I-D.ietf-oauth-attestation-based-client-auth] Looker, T., Bastian, P., and C. Bormann, "OAuth 2.0 Attestation-Based Client Authentication", Work in Progress, Internet-Draft, draft-ietf-oauth-attestation-based-client-auth-07, 15 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-attestation-based-client-auth-07>.

[I-D.ietf-oauth-sd-jwt-vc] Terbu, O., Fett, D., and B. Campbell, "SD-JWT-based Verifiable Credentials (SD-JWT VC)", Work in Progress, Internet-Draft, draft-ietf-oauth-sd-jwt-vc-11, 15 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-sd-jwt-vc-11>.

[I-D.ietf-oauth-status-list] Looker, T., Bastian, P., and C. Bormann, "Token Status List (TSL)", Work in Progress, Internet-Draft, draft-ietf-oauth-status-list-12, 7 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-status-list-12>.

[IANA.COSE] IANA, "CBOR Object Signing and Encryption (COSE)", <https://www.iana.org/assignments/cose/cose.xhtml>.

[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 licence — Part 5: Mobile driving licence (mDL) application", 2021, <https://www.iso.org/standard/69084.html>.

[ISO.18045] ISO, "ISO/IEC 18045:2022 Information security, cybersecurity and privacy protection — Evaluation criteria for IT security — Methodology for IT security evaluation", 2022, <https://www.iso.org/standard/72889.html>.

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

[OpenID.Federation] Ed., R. H., Jones, M. B., Solberg, A., Bradley, J., Marco, G. D., and V. Dzhuvinov, "OpenID Federation 1.0", 2 June 2025, <https://openid.net/specs/openid-federation-1_0-43.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>.

[RFC3066] Alvestrand, H., "Tags for the Identification of Languages", RFC 3066, DOI 10.17487/RFC3066, January 2001, <https://www.rfc-editor.org/info/rfc3066>.

[RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <https://www.rfc-editor.org/info/rfc5646>.

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

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

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/info/rfc6750>.

[RFC6755] Campbell, B. and H. Tschofenig, "An IETF URN Sub-Namespace for OAuth", RFC 6755, DOI 10.17487/RFC6755, October 2012, <https://www.rfc-editor.org/info/rfc6755>.

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

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

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

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

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

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

[RFC7636] Sakimura, N., Ed., Bradley, J., and N. Agarwal, "Proof Key for Code Exchange by OAuth Public Clients", RFC 7636, DOI 10.17487/RFC7636, September 2015, <https://www.rfc-editor.org/info/rfc7636>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

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

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

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/info/rfc8615>.

[RFC8707] Campbell, B., Bradley, J., and H. Tschofenig, "Resource Indicators for OAuth 2.0", RFC 8707, DOI 10.17487/RFC8707, February 2020, <https://www.rfc-editor.org/info/rfc8707>.

[RFC8725] Sheffer, Y., Hardt, D., and M. Jones, "JSON Web Token Best Current Practices", BCP 225, RFC 8725, DOI 10.17487/RFC8725, February 2020, <https://www.rfc-editor.org/info/rfc8725>.

[RFC9068] Bertocci, V., "JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens", RFC 9068, DOI 10.17487/RFC9068, October 2021, <https://www.rfc-editor.org/info/rfc9068>.

[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>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[RFC9396] Lodderstedt, T., Richer, J., and B. Campbell, "OAuth 2.0 Rich Authorization Requests", RFC 9396, DOI 10.17487/RFC9396, May 2023, <https://www.rfc-editor.org/info/rfc9396>.

[RFC9449] Fett, D., Campbell, B., Bradley, J., Lodderstedt, T., Jones, M., and D. Waite, "OAuth 2.0 Demonstrating Proof of Possession (DPoP)", RFC 9449, DOI 10.17487/RFC9449, September 2023, <https://www.rfc-editor.org/info/rfc9449>.

[USASCII] American National Standards Institute, "Coded Character Set -- 7-bit American Standard Code for Information Interchange", 2002, <https://www.unicode.org/L2/L2006/06388-review-incits4.pdf>.

17. Informative References

[BCP240] IETF, "BCP240", January 2025, <https://www.rfc-editor.org/info/bcp240>.

[DIF.Well-Known_DID] Buchner, D., Steele, O., and T. Looker, "Well Known DID Configuration", <https://identity.foundation/specs/did-configuration/>.

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

[IANA.MediaTypes] IANA, "Media Types", <https://www.iana.org/assignments/media-types>.

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

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

[IANA.Well-Known.URIs] IANA, "Well-Known URIs", <https://www.iana.org/assignments/well-known-uris>.

[ISO.29100] ISO, "ISO/IEC 29100:2011 Information technology — Security techniques — Privacy framework", <https://standards.iso.org/ittf/PubliclyAvailableStandards/index.html>.

[JSON-LD] Kellogg, G., Sporny, M., Longley, D., Lanthaler, M., Champin, P., and N. Lindström, "JSON-LD 1.1: A JSON-based Serialization for Linked Data.", 16 July 2020, <https://www.w3.org/TR/2020/REC-json-ld11-20200716/>.

[LD_Suite_Registry] Sporny, M., Reed, D., and O. Steele, "Linked Data Cryptographic Suite Registry", 29 December 2020, <https://w3c-ccg.github.io/ld-cryptosuite-registry/>.

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

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

[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>.

[VC_DATA] Sporny, M., Noble, G., Longley, D., Burnett, D. C., Zundel, B., and K. D. Hartog, "Verifiable Credentials Data Model 1.1", 3 March 2022, <https://www.w3.org/TR/2022/REC-vc-data-model-20220303>.

[VC_DATA_2.0] Sporny, M., Jr, T. T., Herman, I., and G. Cohen, "Verifiable Credentials Data Model 2.0", 15 May 2025, <https://www.w3.org/TR/2025/REC-vc-data-model-2.0-20250515/>.

[VC_Data_Integrity] Sporny, M., Jr, T. T., Herman, I., Longley, D., and G. Bernstein, "Verifiable Credential Data Integrity 1.0", 15 May 2025, <https://www.w3.org/TR/2025/REC-vc-data-integrity-20250515/>.

[secanalysis] Hauck, F., "OpenID for Verifiable Credentials: Formal Security Analysis using the Web Infrastructure Model", 2 October 2023, <https://elib.uni-stuttgart.de/items/07055a8e-a85e-42b9-98b5-11f046d5fb91>.

Appendix A. Credential Format Profiles

This specification defines several extension points to accommodate the differences across Credential Formats. Sets of Credential Format-specific parameters or claims referred to as Credential Format Profiles are identified by the Credential Format Identifier and used at these extension points.

This section defines Credential Format Profiles for a few of the commonly used Credential Formats. Other specifications or deployments can define their own Credential Format Profiles. It is RECOMMENDED that new Credential Format Profiles use the media type of the particular Credential Format for the Credential Format

Identifier.

A.1. W3C Verifiable Credentials

Sections 6.1 and 6.2 of [VC_DATA] define how Verifiable Credentials MAY or MAY NOT use JSON-LD [JSON-LD]. As acknowledged in Section 4.1 of [VC_DATA], implementations can behave differently regarding processing of the `@context` property whether JSON-LD is used or not.

This specification therefore differentiates between the following three Credential Formats for W3C Verifiable Credentials:

- VC signed as a JWT, not using JSON-LD (`jwt_vc_json`)
- VC signed as a JWT, using JSON-LD (`jwt_vc_json-ld`)
- VC secured using Data Integrity, using JSON-LD, with a proof suite requiring Linked Data canonicalization (`ldp_vc`)

Note: VCs secured using Data Integrity MAY NOT necessarily use JSON-LD and MAY NOT necessarily use proof suites requiring Linked Data canonicalization. Credential Format Profiles for them may be defined in the future versions of this specification.

Distinct Credential Format Identifiers, extension parameters/claims, and processing rules are defined for each of the above-mentioned Credential Formats.

A.1.1. VC Signed as a JWT, Not Using JSON-LD

A.1.1.1. Format Identifier

The Credential Format Identifier is `jwt_vc_json`.

When the `format` value is `jwt_vc_json`, the entire Credential Offer, Authorization Details, Credential Request and Credential Issuer metadata, including `credential_definition` object, MUST NOT be processed using JSON-LD rules.

A.1.1.2. Credential Issuer Metadata

Cryptographic algorithm identifiers used in the `credential_signing_alg_values_supported` parameter are case sensitive strings and SHOULD be one of those JWS Algorithm Names defined in [IANA.JOSE].

The following additional Credential Issuer metadata parameters are defined for this Credential Format for use in the `credential_configurations_supported` parameter, in addition to those defined in Section 12.2.4.

- `credential_definition`: REQUIRED. Object containing the detailed description of the Credential type. It consists of the following parameter:
  - `type`: REQUIRED. Array designating the types a certain Credential type supports, according to [VC_DATA], Section 4.3.

The following is a non-normative example of an object containing the `credential_configurations_supported` parameter for Credential Format `jwt_vc_json`:

```
{
  "credential_configurations_supported": {
    "UniversityDegreeCredential": {
      "format": "jwt_vc_json",
      "scope": "UniversityDegree",
        "cryptographic_binding_methods_supported": [
          "did:example"
        ],
        "credential_signing_alg_values_supported": [
          "ES256"
        ],
        "credential_definition": {
          "type": [
            "VerifiableCredential",
            "UniversityDegreeCredential"
          ]
        },
        "proof_types_supported": {
          "jwt": {
            "proof_signing_alg_values_supported": [
              "ES256"
            ]
          }
        },
        "credential_metadata": {
          "claims": [
            {
              "path": ["credentialSubject", "given_name"],
              "display": [
                {
                  "name": "Given Name",
                  "locale": "en-US"
                }
              ]
            },
            {
              "path": ["credentialSubject", "family_name"],
              "display": [
                {
                  "name": "Surname",
                  "locale": "en-US"
                }
              ]
            },
            {"path": ["credentialSubject", "degree"]},
            {
              "path": ["credentialSubject", "gpa"],
              "mandatory": true,
              "display": [
                {
                  "name": "GPA"
                }
              ]
            }
          ],
          "display": [
            {
              "name": "University Credential",
              "locale": "en-US",
              "logo": {
                "uri": "https://university.example.edu/public/logo.png",
                "alt_text": "a square logo of a university"
              },
              "background_color": "#12107c",
              "text_color": "#FFFFFF"
            }
          ]
        }
```

```
      }
    }
  }
```

A.1.1.3. Authorization Details

The following is a non-normative example of an authorization details object with Credential Format `jwt_vc_json`:

```
[
  {
    "type": "openid_credential",
    "credential_configuration_id": "UniversityDegreeCredential",
    "claims": [
      {"path": ["credentialSubject", "given_name"]},
      {"path": ["credentialSubject", "family_name"]},
      {"path": ["credentialSubject", "degree"]}
    ]
  }
]
```

A.1.1.4. Credential Response

The value of the `credential` claim in the Credential Response MUST be a JWT. Credentials of this format are already a sequence of base64url-encoded values separated by period characters and MUST NOT be re-encoded.

The following is a non-normative example of a Credential Response with Credential Format `jwt_vc_json` (with line breaks within values for display purposes only):

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

{
  "credentials": [
    {
      "credential": "eyJ0eXAiOiJKV1QiLCJhbGciOiJFUzI1NiIsImtpZCI6I
      nVybjppZXRmOnBhcmFtczpvYXV0aDpqd2stdGh1bWJwcmludDpzaGEtMjU2O
      m1sVXBvZzd2RWV3RkJlbTZVbDA5YzJkdFR3YzhkRnpWcElEWDNzcUdXVzAif
      Q.eyJ2YyI6eyJAY29udGV4dCI6WyJodHRwczovL3d3dy53My5vcmcvMjAxOC
      9jcmVkZW50aWFscy92MSIsImh0dHBzOi8vd3d3LnczLm9yZy8yMDE4L2NyZW
      RlbnRpYWxzL2V4YW1wbGVzL3YxIl0sImlkIjoiaHR0cHM6Ly9jcmVkZW50aW
      FsLWlzc3Vlci5leGFtcGxlLmNvbS9jcmVkZW50aWFscy8zNzMyIiwidHlwZS
      I6WyJWZXJpZmlhYmxlQ3JlZGVudGlhbCIsIlVuaXZlcnNpdHlEZWdyZWVDcm
      VkZW50aWFsIl0sImlzc3VlciI6Imh0dHBzOi8vY3JlZGVudGlhbC1pc3N1ZX
      IuZXhhbXBsZS5jb20iLCJpc3N1YW5jZURhdGUiOiIyMDI1LTAxLTAxVDAwOj
      AwOjAwWiIsImNyZWRlbnRpYWxTdWJqZWN0Ijp7ImlkIjoiZGlkOmp3azpleU
      pyYVdRaU9pSjFjbTQ2YVdWMFpqcHdZWEpoYlhNNmIyRjFkR2c2YW5kckxYUm
      9kVzFpY0hKcGJuUTZjMmhoTFRJMU5qcFdZa3BQVTNacWVGVTJURGhETjBkVl
      R6UmtjMmhKV1ZZemVtSjJSbmRyV1VJME0xbEtOVXQwZERoRklpd2lhM1I1SW
      pvaVJVTWlMQ0pqY25ZaU9pSlFMVEkxTmlJc0ltRnNaeUk2SWtWVE1qVTJJaX
      dpZUNJNklrMWtReTFQUzNFMFFWRktabFpEV0RWNmNGRnZURGhxTkZaRlpuWl
      FXRGs0ZEZVNWFIaGpUbGhIY204aUxDSjVJam9pYm5OWGJtWmlOazVYYzBzek
      9VSklMV2hCWVZOclExTmxORUo1YldWT2MyTktSVjl6WVVRelJETmlUU0o5Ii
      wiZGVncmVlIjp7InR5cGUiOiJCYWNoZWxvckRlZ3JlZSIsIm5hbWUiOiJCYW
      NoZWxvciBvZiBTY2llbmNlIGFuZCBBcnRzIn19fSwiaXNzIjoiaHR0cHM6Ly
      9jcmVkZW50aWFsLWlzc3Vlci5leGFtcGxlLmNvbSIsIm5iZiI6MTczNTY4OT
      YwMCwianRpIjoiaHR0cHM6Ly9jcmVkZW50aWFsLWlzc3Vlci5leGFtcGxlLm
      NvbS9jcmVkZW50aWFscy8zNzMyIiwic3ViIjoiZGlkOmp3azpleUpyYVdRaU
      9pSjFjbTQ2YVdWMFpqcHdZWEpoYlhNNmIyRjFkR2c2YW5kckxYUm9kVzFpY0
      hKcGJuUTZjMmhoTFRJMU5qcFdZa3BQVTNacWVGVTJURGhETjBkVlR6UmtjMm
      hKV1ZZemVtSjJSbmRyV1VJME0xbEtOVXQwZERoRklpd2lhM1I1SWpvaVJVTW
      lMQ0pqY25ZaU9pSlFMVEkxTmlJc0ltRnNaeUk2SWtWVE1qVTJJaXdpZUNJNk
      lrMWtReTFQUzNFMFFWRktabFpEV0RWNmNGRnZURGhxTkZaRlpuWlFXRGs0ZE
      ZVNWFIaGpUbGhIY204aUxDSjVJam9pYm5OWGJtWmlOazVYYzBzek9VSklMV2
      hCWVZOclExTmxORUo1YldWT2MyTktSVjl6WVVRelJETmlUU0o5In0.k13xQC
      nQIKAIuwQIbg37dwlNr8D6_2YUQtDTVQCq-ZsjcXxHagGC_VIZtd7RpR8OvB
      zTBHVwrBRD-_RzoV2Ofg"
    }
  ]
}
```

The following is the dereferenced document for the Issuer HTTP URL identifier that matches the Credential in the above example (with line breaks within values for display purposes only):

```
{
  "jwks": [
    {
      "kid": "urn:ietf:params:oauth:jwk-thumbprint:sha-256:mlUpog7vEewFBem6Ul09
       c2dtTwc8dFzVpIDX3sqGWW0",
      "kty": "EC",
      "crv": "P-256",
      "alg": "ES256",
      "x": "_LC1FTUl0MltKAOQzXNsofVMpWFV2obLGrNCat_CQ-g",
      "y": "kBjoyjNuMVAOq--qVUgylDoLKuMdk4imS-Kk5ahuYIU"
    }
  ]
}
```

A.1.2. VC Secured using Data Integrity, using JSON-LD, with a Proof Suite Requiring Linked Data Canonicalization

A.1.2.1. Format Identifier

The Credential Format Identifier is `ldp_vc`.

When the `format` value is `ldp_vc`, the entire Credential Offer, Authorization Details, Credential Request and Credential Issuer metadata, including `credential_definition` object, MUST NOT be processed using JSON-LD rules.

The `@context` value in the `credential_definition` object can be used by the Wallet to check whether it supports a certain VC or not. If necessary, the Wallet could apply JSON-LD processing to the Credential issued by the Credential Issuer.

Note: Data Integrity used to be called Linked Data Proofs, hence the "ldp" in the Credential Format Identifier.

A.1.2.2. Credential Issuer Metadata

Cryptographic algorithm identifiers used in the `credential_signing_alg_values_supported` parameter are case sensitive strings and SHOULD be one of those signature suite identifiers defined in [LD_Suite_Registry].

The following additional Credential Issuer metadata parameters are defined for this Credential Format for use in the `credential_configurations_supported` parameter, in addition to those defined in Section 12.2.4:

- `credential_definition`: REQUIRED. Object containing the detailed description of the Credential type. It consists of the following parameters:
  - `@context`: REQUIRED. Array as defined in [VC_DATA], Section 4.1.
  - `type`: REQUIRED. Array designating the types a certain credential type supports, according to [VC_DATA], Section 4.3.

The following is a non-normative example of an object containing the `credential_configurations_supported` parameter for Credential Format `ldp_vc`:

```
{
  "credential_configurations_supported": {
    "UniversityDegree_LDP_VC": {
      "format": "ldp_vc",
      "cryptographic_binding_methods_supported": [
        "did:example"
      ],
      "credential_signing_alg_values_supported": [
        "Ed25519Signature2018"
      ],
      "credential_definition": {
        "@context": [
          "https://www.w3.org/2018/credentials/v1",
          "https://www.w3.org/2018/credentials/examples/v1"
        ],
        "type": [
          "VerifiableCredential",
          "UniversityDegreeCredential"
        ]
      },
      "credential_metadata": {
        "claims": [
          {
            "path": ["credentialSubject", "given_name"],
            "display": [
              {
                "name": "Given Name",
                "locale": "en-US"
              }
            ]
          },
          {
            "path": ["credentialSubject", "family_name"],
            "display": [
              {
                "name": "Surname",
                "locale": "en-US"
              }
            ]
          },
          {
            "path": ["credentialSubject", "degree"]
          },
          {
            "path": ["credentialSubject", "gpa"],
            "mandatory": true,
            "display": [
              {
                "name": "GPA"
              }
            ]
          }
        ],
        "display": [
          {
            "name": "University Credential",
            "locale": "en-US",
            "logo": {
              "uri": "https://university.example.edu/public/logo.png",
              "alt_text": "a square logo of a university"
            },
            "background_color": "#12107c",
            "text_color": "#FFFFFF"
          }
        ]
      }
    }
  }
}
```

A.1.2.3. Authorization Details

The following is a non-normative example of an authorization details object with Credential Format `ldp_vc`:

```
[
  {
    "type": "openid_credential",
    "credential_configuration_id": "UniversityDegree_LDP_VC",
    "claims": [
      {"path": ["credentialSubject", "given_name"]},
      {"path": ["credentialSubject", "family_name"]},
      {"path": ["credentialSubject", "degree"]}
    ]
  }
]
```

A.1.2.4. Credential Response

The value of the `credential` claim in the Credential Response MUST be a JSON object. Credentials of this format MUST NOT be re-encoded.

The following is a non-normative example of a Credential Response with Credential Format `ldp_vc`:

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

{
  "credentials": [
    {
      "credential": {
        "@context": [
          "https://www.w3.org/2018/credentials/v1",
          "https://www.w3.org/2018/credentials/examples/v1"
        ],
        "id": "http://example.edu/credentials/3732",
        "type": [
          "VerifiableCredential",
          "UniversityDegreeCredential"
        ],
        "issuer": "https://example.edu/issuers/565049",
        "issuanceDate": "2010-01-01T00:00:00Z",
        "credentialSubject": {
          "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
          "degree": {
            "type": "BachelorDegree",
            "name": "Bachelor of Science and Arts"
          }
        },
        "proof": {
          "type": "Ed25519Signature2020",
          "created": "2022-02-25T14:58:43Z",
          "verificationMethod": "https://example.edu/issuers/565049#key-1",
          "proofPurpose": "assertionMethod",
          "proofValue": "zeEdUoM7m9cY8ZyTpey83yBKeBcmcvbyrEQzJ19rD2UXArU2U1
                         jPGoEtrRvGYppdiK37GU4NBeoPakxpWhAvsVSt"
        }
      }
    }
  ]
}
```

A.1.3. VC signed as a JWT, Using JSON-LD

A.1.3.1. Format Identifier

The Credential Format Identifier is `jwt_vc_json-ld`.

When the `format` value is `jwt_vc_json-ld`, the entire Credential Offer, Authorization Details, Credential Request and Credential Issuer metadata, including `credential_definition` object, MUST NOT be processed using JSON-LD rules.

The `@context` value in the `credential_definition` parameter can be used by the Wallet to check whether it supports a certain VC or not. If necessary, the Wallet could apply JSON-LD processing to the Credential issued by the Credential Issuer.

A.1.3.2. Credential Issuer Metadata

The definitions in Appendix A.1.2.2 apply for metadata of Credentials of this type as well.

A.1.3.3. Authorization Details

The definitions in Appendix A.1.2.3 apply for Credentials of this type as well.

A.1.3.4. Credential Response

The definitions in Appendix A.1.1.4 apply for Credentials of this type as well.

A.2. Mobile Documents or mdocs (ISO/IEC 18013)

This section defines a Credential Format Profile for credentials that conform to the mobile document (mdoc) format specified in ISO/IEC 18013-5 [ISO.18013-5].

ISO/IEC 18013-5:2021 [ISO.18013-5] defines the mdoc format in the context of mobile driving licences (mDLs). While the specification is focused on mDLs, the mdoc format itself is general-purpose and can be used for other types of Credentials.

A.2.1. Format Identifier

The Credential Format Identifier is `mso_mdoc`. This refers to the Mobile Security Object (MSO) which secures the mdoc data model encoded as CBOR.

A.2.2. Credential Issuer Metadata

Cryptographic algorithm identifiers used in the `credential_signing_alg_values_supported` parameter correspond to the numeric COSE algorithm identifiers used to secure the `IssuerAuth` COSE structure, as defined in [ISO.18013-5]. The values SHOULD be chosen from the following:

- The value exactly matches the `alg` value in the `IssuerAuth` COSE header;
- Or, the value is a fully specified algorithm, as defined in [I-D.ietf-jose-fully-specified-algorithms], and the combination of the `alg` value and the signing key's curve in the `IssuerAuth` COSE structure matches the combination specified by the fully specified algorithm.

Consequently, depending on the approach taken by the Issuer as described above, the `alg` value in the `IssuerAuth` COSE header MAY differ from the identifier listed in the `credential_signing_alg_values_supported` metadata parameter. This metadata parameter is primarily informational for the Wallet.

Example: If the `IssuerAuth` structure contains an `alg` header value of `-7` (ECDSA with SHA-256, per [IANA.COSE]) and is signed using a P-256 key, it matches both `-7` and `-9` in `credential_signing_alg_values_supported`. The latter (`-9`) corresponds to ECDSA with P-256 and SHA-256, as defined in [I-D.ietf-jose-fully-specified-algorithms].

The following additional Credential Issuer metadata parameters are defined for this Credential Format for use in the `credential_configurations_supported` parameter, in addition to those defined in Section 12.2.4.

- `doctype`: REQUIRED. String identifying the Credential type, as defined in [ISO.18013-5].

The following is a non-normative example of an object containing the `credential_configurations_supported` parameter for Credential Format `mso_mdoc`:

```
{
  "credential_configurations_supported": {
    "org.iso.18013.5.1.mDL": {
      "format": "mso_mdoc",
      "doctype": "org.iso.18013.5.1.mDL",
      "cryptographic_binding_methods_supported": [
        "cose_key"
      ],
      "credential_signing_alg_values_supported": [
        -7, -9
      ],
      "credential_metadata": {
        "display": [
          {
            "name": "Mobile Driving License",
            "locale": "en-US",
            "logo": {
              "uri": "https://state.example.org/public/mdl.png",
              "alt_text": "state mobile driving license"
            },
            "background_color": "#12107c",
            "text_color": "#FFFFFF"
          },
          {
            "name": "モバイル運転免許証",
            "locale": "ja-JP",
            "logo": {
              "uri": "https://state.example.org/public/mdl.png",
              "alt_text": "米国州発行のモバイル運転免許証"
            },
            "background_color": "#12107c",
            "text_color": "#FFFFFF"
          }
        ],
        "claims": [
          {
            "path": ["org.iso.18013.5.1", "given_name"],
            "display": [
              {
                "name": "Given Name",
                "locale": "en-US"
              },
              {
                "name": "名前",
                "locale": "ja-JP"
              }
            ]
          },
          {
            "path": ["org.iso.18013.5.1", "family_name"],
            "display": [
              {
                "name": "Surname",
                "locale": "en-US"
              }
            ]
          },
          {
            "path": ["org.iso.18013.5.1", "birth_date"],
            "mandatory": true
          },
          {"path": ["org.iso.18013.5.1.aamva", "organ_donor"]}
        ]
      }
    }
  }
}
```

A.2.3. Authorization Details

The following is a non-normative example of an authorization details object with Credential Format `mso_mdoc`:

```
[
  {
    "type": "openid_credential",
    "credential_configuration_id": "org.iso.18013.5.1.mDL",
    "claims": [
      {"path": ["org.iso.18013.5.1","given_name"]},
      {"path": ["org.iso.18013.5.1","family_name"]},
      {"path": ["org.iso.18013.5.1","birth_date"]},
      {"path": ["org.iso.18013.5.1.aamva","organ_donor"]}
    ]
  }
]
```

A.2.4. Credential Response

The value of the `credential` claim in the Credential Response MUST be a string that is the base64url-encoded representation of the CBOR-encoded `IssuerSigned` structure, as defined in [ISO.18013-5].

The following is a non-normative example of a Credential Response containing a Credential of format `mso_mdoc`.

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

{
  "credentials": [
    {
      "credential": "omppc3N1ZXJBdXRohEOhASahG...ArQwggKwMIICVqADAgEC"
    }
  ]
}
```

A.3. IETF SD-JWT VC

This section defines a Credential Format Profile for Credentials complying with [I-D.ietf-oauth-sd-jwt-vc].

A.3.1. Format Identifier

The Credential Format Identifier is `dc+sd-jwt`.

A.3.2. Credential Issuer Metadata

Cryptographic algorithm identifiers used in the `credential_signing_alg_values_supported` parameter are case sensitive strings and SHOULD be one of those JWS Algorithm Names defined in [IANA.JOSE].

The following additional Credential Issuer metadata parameters are defined for this Credential Format for use in the `credential_configurations_supported` parameter, in addition to those defined in Section 12.2.4.

- `vct`: REQUIRED. String designating the type of the Credential, as defined in [I-D.ietf-oauth-sd-jwt-vc].

The following is a non-normative example of an object comprising the `credential_configurations_supported` parameter for Credential Format `dc+sd-jwt`.

```
{
  "credential_configurations_supported": {
    "SD_JWT_VC_example_in_OpenID4VCI": {
      "format": "dc+sd-jwt",
      "scope": "SD_JWT_VC_example_in_OpenID4VCI",
      "cryptographic_binding_methods_supported": [
        "jwk"
      ],
      "credential_signing_alg_values_supported": [
        "ES256"
      ],
      "proof_types_supported": {
        "jwt": {
          "proof_signing_alg_values_supported": [
            "ES256"
          ],
          "key_attestations_required": {
            "key_storage": [ "iso_18045_moderate" ],
            "user_authentication": [ "iso_18045_moderate" ]
          }
        }
      },
      "vct": "SD_JWT_VC_example_in_OpenID4VCI",
      "credential_metadata": {
        "display": [
          {
            "name": "IdentityCredential",
            "logo": {
              "uri": "https://university.example.edu/public/logo.png",
              "alt_text": "a square logo of a university"
            },
            "locale": "en-US",
            "background_color": "#12107c",
            "text_color": "#FFFFFF"
          }
        ],
        "claims": [
          {
            "path": ["given_name"],
            "display": [
              {
                "name": "Given Name",
                "locale": "en-US"
              },
              {
                "name": "Vorname",
                "locale": "de-DE"
              }
            ]
          },
          {
            "path": ["family_name"],
            "display": [
              {
                "name": "Surname",
                "locale": "en-US"
              },
              {
                "name": "Nachname",
                "locale": "de-DE"
              }
            ]
          },
          {"path": ["email"]},
          {"path": ["phone_number"]},
          {
```

```
          "path": ["address"],
          "display": [
            {
              "name": "Place of residence",
              "locale": "en-US"
            },
            {
              "name": "Wohnsitz",
              "locale": "de-DE"
            }
          ]
        },
        {"path": ["address", "street_address"]},
        {"path": ["address", "locality"]},
        {"path": ["address", "region"]},
        {"path": ["address", "country"]},
        {"path": ["birthdate"]},
        {"path": ["is_over_18"]},
        {"path": ["is_over_21"]},
        {"path": ["is_over_65"]}
      ]
    }
  }
}
}
```

A.3.3. Authorization Details

The following is a non-normative example of an authorization details object with Credential Format `dc+sd-jwt`.

```
[
  {
    "type": "openid_credential",
    "credential_configuration_id": "PID_SD_JWT_VC",
    "claims": [
      {"path": ["given_name"]},
      {"path": ["family_name"]}
    ]
  }
]
```

A.3.4. Credential Response

The value of the `credential` claim in the Credential Response MUST be a string that is an SD-JWT VC. Credentials of this format are already suitable for transfer and, therefore, they need not and MUST NOT be re-encoded.

The following is a non-normative example of a Credential Response containing a Credential of format `dc+sd-jwt` (with line breaks within values for display purposes only).

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

{
  "credentials": [
    {
      "credential": "eyJhbGciOiAiRVMyNTYiLCAidHlwIjogImRjK3NkLWp3d
      CIsICJraWQiOiAiZG9jLXNpZ25lci0wNS0yNS0yMDIyIn0.eyJfc2QiOiBbI
      jA5dktySk1PbHlUV00wc2pwdV9wZE9CVkJRMk0xeTNLaHBINTE1blhrcFkiL
      CAiMnJzakdiYUMwa3k4bVQwcEpyUGlvV1RxMF9kYXcxc1g3NnBvVWxnQ3diS
      SIsICJFa084ZGhXMGRIRUpidlVIbEVfVkNldUM5dVJFTE9pZUxaaGg3WGJVV
      HRBIiwgIklsRHpJS2VpWmREd3BxcEs2WmZieXBoRnZ6NUZnbldhLXNONndxU
      VhDaXciLCAiSnpZakg0c3ZsaUgwUjNQeUVNZmVadTZKdDY5dTVxZWhabzdGN
      0VQWWxTRSIsICJQb3JGYnBLdVZ1Nnh5bUphZ3ZrRnNGWEFiUm9jMkpHbEFVQ
      TJCQTRvN2NJIiwgIlRHZjRvTGJnd2Q1SlFhSHlLVlFaVTlVZEdFMHc1cnREc
      3JaemZVYW9tTG8iLCAiamRyVEU4WWNiWTRFaWZ1Z2loaUFlX0JQZWt4SlFaS
      UNlaVVRd1k5UXF4SSIsICJqc3U5eVZ1bHdRUWxoRmxNXzNKbHpNYVNGemdsa
      FFHMERwZmF5UXdMVUs0Il0sICJpc3MiOiAiaHR0cHM6Ly9leGFtcGxlLmNvb
      S9pc3N1ZXIiLCAiaWF0IjogMTY4MzAwMDAwMCwgImV4cCI6IDE4ODMwMDAwM
      DAsICJ2Y3QiOiAiaHR0cHM6Ly9jcmVkZW50aWFscy5leGFtcGxlLmNvbS9pZ
      GVudGl0eV9jcmVkZW50aWFsIiwgIl9zZF9hbGciOiAic2hhLTI1NiIsICJjb
      mYiOiB7Imp3ayI6IHsia3R5IjogIkVDIiwgImNydiI6ICJQLTI1NiIsICJ4I
      jogIlRDQUVSMTladnUzT0hGNGo0VzR2ZlNWb0hJUDFJTGlsRGxzN3ZDZUdlb
      WMiLCAieSI6ICJaeGppV1diWk1RR0hWV0tWUTRoYlNJaXJzVmZ1ZWNDRTZ0N
      GpUOUYySFpRIn19fQ.dVjA0sh4xGD32uPqc9h4WHiEL3A08kiKNE08IIrtn3
      PJvljLU7n19LBTtuzPFZoc_GoPuS97SIDbz96K8pkZew~WyIyR0xDNDJzS1F
      2ZUNmR2ZyeU5STjl3IiwgImdpdmVuX25hbWUiLCAiSm9obiJd~WyJlbHVWNU
      9nM2dTTklJOEVZbnN4QV9BIiwgImZhbWlseV9uYW1lIiwgIkRvZSJd~WyI2S
      Wo3dE0tYTVpVlBHYm9TNXRtdlZBIiwgImVtYWlsIiwgImpvaG5kb2VAZXhhb
      XBsZS5jb20iXQ~WyJlSThaV205UW5LUHBOUGVOZW5IZGhRIiwgInBob25lX2
      51bWJlciIsICIrMS0yMDItNTU1LTAxMDEiXQ~WyJRZ19PNjR6cUF4ZTQxMmE
      xMDhpcm9BIiwgImFkZHJlc3MiLCB7InN0cmVldF9hZGRyZXNzIjogIjEyMyB
      NYWluIFN0IiwgImxvY2FsaXR5IjogIkFueXRvd24iLCAicmVnaW9uIjogIkF
      ueXN0YXRlIiwgImNvdW50cnkiOiAiVVMifV0~WyJBSngtMDk1VlBycFR0TjR
      RTU9xUk9BIiwgImJpcnRoZGF0ZSIsICIxOTQwLTAxLTAxIl0~WyJQYzMzSk0
      yTGNoY1VfbEhnZ3ZfdWZRIiwgImlzX292ZXJfMTgiLCB0cnVlXQ~WyJHMDJO
      U3JRZmpGWFE3SW8wOXN5YWpBIiwgImlzX292ZXJfMjEiLCB0cnVlXQ~WyJsa
      2x4RjVqTVlsR1RQVW92TU5JdkNBIiwgImlzX292ZXJfNjUiLCB0cnVlXQ~"
    }
  ]
}
```

Appendix B. Claims Description

Claims description objects are used in two places in this specification, in the Credential Issuer metadata and in the Authorization Details. The following sections define the structure and semantics of claims description objects.

B.1. Claims Description for Authorization Details

A claims description object as used in authorization details is an object that defines the requirements for the claims that the Wallet requests to be included in the Credential.

For a claims description object, the following keys defined by this specification can be used to describe the claim or claims (other keys MAY also be used):

- `path`: REQUIRED. Non-empty array. Claim path pointer as defined in Appendix C to identify the claim(s) in the Credential.
- `mandatory`: OPTIONAL. Boolean which, when set to `true`, indicates that the Wallet will only accept a Credential that includes this claim. If set to `false`, the claim is not required to be included in the Credential. If the `mandatory` parameter is omitted, the default value is `false`.

The rules defined in Appendix B.3 apply.

B.2. Claims Description for Issuer Metadata

A claims description object as used in the Credential Issuer metadata is an object used to describe how a certain claim in the Credential is displayed to the End-User. It is used in the `claims` parameter in the Credential Issuer metadata defined in Appendix A. The following keys can be used to describe the claim or claims:

- `path`: REQUIRED. The value MUST be a non-empty array representing a claims path pointer that specifies the path to a claim within the credential, as defined in Appendix C.
- `mandatory`: OPTIONAL. Boolean which, when set to `true`, indicates that the Credential Issuer will always include this claim in the issued Credential. If set to `false`, the claim is not included in the issued Credential if the wallet did not request the inclusion of the claim, and/or if the Credential Issuer chose to not include the claim. If the `mandatory` parameter is omitted, the default value is `false`.
- `display`: OPTIONAL. A non-empty array of objects, where each object contains display properties of a certain claim in the Credential for a certain language. Below is a non-exhaustive list of valid parameters that MAY be included:
    - `name`: OPTIONAL. String value of a display name for the claim.
    - `locale`: OPTIONAL. String value that identifies the language of this object, represented as language tag values defined in BCP47 [RFC5646]. There MUST be only one object for each language identifier.

The rules defined in Appendix B.3 apply.

B.3. Processing Rules for Claims Description Objects

The order of claims description objects in the `claims` array is used by the Wallet to determine the order in which the claims are displayed to the End-User, unless another mechanism is defined by the profile.

When a repeated or contradictory claim description is provided, the processing MUST be aborted. This is in particular the case if

- the same claim is addressed by two or more claims description objects in the `claims` array, or
- there is a claims description object with a `path` that addresses a set of claims in an array (using `null`, as defined in Appendix C) and another object that uses a non-negative integer to address a specific claim in the same array, or
- there is a claims description object indicating that a certain claim is an array (implied by using `null` or a non-negative integer in the `path`), and another object indicating that the same claim is an object (implied by using a string in the `path`).

Appendix C. Claims Path Pointer

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

C.1. Semantics for JSON-based credentials

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

A string value indicates that the respective key is to be selected, a null value indicates that all elements of the currently selected array(s) are to be selected; and a non-negative integer indicates that the respective index in an array is to be selected. Negative integer values are not used for JSON-based credentials. The path is formed as follows:

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

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

C.1.1. Processing

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

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

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

C.2. Semantics for ISO mdoc-based credentials

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

A claims path pointer into an ISO mdoc always contains at least two path components of type string. Start with an array of two elements where the first element represents the namespace and the second element represents the data element identifier in that namespace and repeat the following until the full path is formed. * To address a claim within a data structure (i.e., map), append the corresponding key (string or integer). * To address an element within an array, append the index (as a non-negative, 0-based integer). * To address all elements of an array, append a null value.

Whether a non-negative integer is interpreted as a map key or an array index depends on the type of the currently selected element.

C.2.1. Processing

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

1. If the path contains fewer than two path components, abort processing and return an error.
2. Select the namespace referenced by the first path component. If the namespace does not exist in the mdoc, abort processing and return an error.
3. Select the data element with the data element identifier that matches the second path component. If the data element does not exist in the mdoc, abort processing and return an error.
4. Process each subsequent path component as follows:
    1. If all the currently selected element(s) are arrays, and the path component is a non-negative integer, select the element at the given index in each array. If the index is out of bounds in a selected array, remove that array from the selection.
    2. If all the currently selected element(s) are maps, and the path component is a string or integer, select the element(s) associated with the key. If the key does not exist in a selected map, remove that map from the selection.
    3. If the path component is null, select all elements in each currently selected array. If any selected element is not an array, abort processing and return an error.
    4. If none of the above applied, abort processing and return an error.
5. If the set of elements currently selected is empty, abort processing and return an error.

The result of the processing is the set of selected elements contained within the selected data element value, or the value itself.

C.3. Claims Path Pointer Example

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

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

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

- `["name"]`: The claim `name` with the value `Arthur Dent` is selected.
- `["address"]`: The claim `address` with its sub-claims as the value is selected.

- `["address", "street_address"]`: The claim `street_address` with the value `42 Market Street` is selected.
- `["degrees", null, "type"]`: All `type` claims in the `degrees` array are selected.
- `["nationalities", 1]`: The second nationality is selected.

Appendix D. Key Attestations

A key attestation, as defined by this specification, is a verifiable statement that demonstrates the authenticity and security properties of a key and its storage component to the Credential Issuer. Keys can be stored in various key storage components, which vary in their ability to protect the private key from extraction and duplication, as well as in the methods used for User authentication to enable key operations. These key storage components can be software-based or hardware-based and may reside on the same device as the Wallet, on external security tokens, or on remote services that facilitate cryptographic key operations. Key attestations are issued either by the Wallet's key storage component itself or by the Wallet Provider. When the Wallet Provider creates the key attestation, it MUST verify the authenticity of its claims about the keys, possibly using platform-specific key attestations.

A Wallet MAY provide key attestations to inform the Credential Issuer about the properties of the cryptographic public keys, such as those used in proof types sent in the Credential Request. Credential Issuers SHOULD evaluate these key attestations to ensure the cryptographic keys meet their security requirements, based on the trust framework, regulatory requirements, laws, or internal policies. A Credential Issuer MUST communicate the need to evaluate key attestations through its metadata or via an out-of-band mechanism.

Key attestations are used by various Credential Issuers with different trust frameworks and requirements, so a common approach is needed for interoperability. Therefore, key attestations SHOULD follow a common format. This helps Credential Issuers create consistent evaluation processes, reducing complexity and errors. Common formats also simplify compliance with regulatory requirements across jurisdictions and support the creation of shared best practices and security standards.

D.1. Key Attestation in JWT format

The JWT is signed by the Wallet Provider or the Wallet's key storage component itself and contains the following elements:

- in the JOSE header,
  - `alg`: REQUIRED. A digital signature algorithm identifier such as per IANA "JSON Web Signature and Encryption Algorithms" registry [IANA.JOSE]. It MUST NOT be `none` or an identifier for a symmetric algorithm (MAC).
  - `typ`: REQUIRED. MUST be `key-attestation+jwt`, which explicitly types the key proof JWT as recommended in Section 3.11 of [RFC8725].

The key attestation may use `x5c`, `kid` or `trust_chain` (as defined in Appendix F.1) to convey the public key and the associated trust mechanism to sign the key attestation.

- in the JWT body,
  - `iat`: REQUIRED (number). Integer for the time at which the key attestation was issued using the syntax defined in [RFC7519].
  - `exp`: OPTIONAL (number). Integer for the time at which the key attestation and the key(s) it is attesting expire, using the syntax defined in [RFC7519]. MUST be present if the attestation is used with the JWT proof type.

    - `attested_keys` : REQUIRED. A non-empty array of attested keys from the same key storage component using the syntax of JWK as defined in [RFC7517].
    - `key_storage` : OPTIONAL. A non-empty array of case sensitive strings that assert the attack potential resistance of the key storage component and its keys attested in the `attested_keys` parameter. This specification defines initial values in Appendix D.2.
    - `user_authentication` : OPTIONAL. A non-empty array of case sensitive strings that assert the attack potential resistance of the user authentication methods allowed to access the private keys from the `attested_keys` parameter. This specification defines initial values in Appendix D.2.
    - `certification` : OPTIONAL. A String that contains a URL that links to the certification of the key storage component.
    - `nonce`: OPTIONAL. String that represents a nonce provided by the Issuer to prove that a key attestation was freshly generated.
    - `status`: OPTIONAL. JSON Object representing the supported revocation check mechanisms, such as the one defined in [I-D.ietf-oauth-status-list]

If used with the `jwt` proof type, the Credential Issuer MUST validate that the JWT used as a proof is signed by a key contained in the attestation in the JOSE Header.

This is an example of a Key Attestation:

```
{
  "typ": "key-attestation+jwt",
  "alg": "ES256",
  "x5c": ["MIIDQjCCA..."]
}
.
{
  "iss": "<identifier of the issuer of this key attestation>",
  "iat": 1516247022,
  "exp": 1541493724,
  "key_storage": [ "iso_18045_moderate" ],
  "user_authentication": [ "iso_18045_moderate" ],
  "attested_keys": [
    {
      "kty": "EC",
      "crv": "P-256",
      "x": "TCAER19Zvu3OHF4j4W4vfSVoHIP1ILilDls7vCeGemc",
      "y": "ZxjiWWbZMQGHVWKVQ4hbSIirsVfuecCE6t4jT9F2HZQ"
    }
  ]
}
```

D.2. Attack Potential Resistance

This specification defines the following values for `key_storage` and `user_authentication`:

- `iso_18045_high`: It MUST be used when key storage or user authentication is resistant to attack with attack potential "High", equivalent to VAN.5 according to [ISO.18045].
- `iso_18045_moderate`: It MUST be used when key storage or user authentication is resistant to attack with attack potential "Moderate", equivalent to VAN.4 according to [ISO.18045].
- `iso_18045_enhanced-basic`: It MUST be used when key storage or user authentication is resistant to attack with attack potential "Enhanced-Basic", equivalent to VAN.3 according to [ISO.18045].

- `iso_18045_basic`: It MUST be used when key storage or user authentication is resistant to attack with attack potential "Basic", equivalent to VAN.2 according to [ISO.18045].

Specifications that extend this list MUST choose collision-resistant values.

When [ISO.18045] is not used, ecosystems may define their own values. If the value does not map to a well-known specification, it is RECOMMENDED that the value is a URL that gives further information about the attack potential resistance and possible relations to level of assurances.

Appendix E. Wallet Attestations in JWT format

The Wallet Attestation defined in this section is a client authentication method especially designed for native App Wallets. Instead of using platform-specific key, app, and/or device attestations directly, it uses a key-bound, platform agnostic common format based on JWTs. This allows Authorization Servers to authenticate Wallets across different platforms in a unified fashion and exposes only a minimum dataset. Also, it allows the Wallet app to directly interact with the Credential Issuer without involving the Wallet Provider's backend. The Authorization Server MUST verify that the Wallet Attestation is signed by an issuer that the Credential Issuer trusts for this purpose.

In a typical architecture of a native Wallet App, the Wallet Provider's backend will use attestations provided by the mobile operating system, like iOS's DeviceCheck or Android's Play Integrity, to validate the app's integrity and authenticity before issuing the Wallet Attestation.

There are two requests where Wallets may need to authenticate during Credential issuance:

- The Wallet sends it in the Pushed Authorization Request
- The Wallet sends it in the Token Request

The Wallet Attestation format follows Section 5.1 "Client Attestation JWT" of [I-D.ietf-oauth-attestation-based-client-auth]. The Wallet Attestation additionally includes the following JWT Claims:

- `wallet_name`: OPTIONAL. String containing a human-readable name of the Wallet.
- `wallet_link`: OPTIONAL. String containing a URL to get further information about the Wallet and the Wallet Provider.
- `status`: OPTIONAL. Status mechanism for the Wallet Attestation as defined in [I-D.ietf-oauth-status-list]

The following is a non-normative example of a decoded Wallet Attestation:

```
{
  "typ": "oauth-client-attestation+jwt",
  "alg": "ES256",
  "kid": "11"
}
.
{
  "iss": "https://wallet-provider.example.com",
  "sub": "https://wallet.example.org",
  "wallet_name": "Wallet Solution X by Wonderland State Department",
  "wallet_link": "https://example.com/wallet/detail_info.html",
  "nbf": 1300815780,
  "exp": 1300819380,
  "cnf": {
    "jwk": {
      "kty": "EC",
      "use": "sig",
      "crv": "P-256",
      "x": "18wHLeIgW9wVN6VD1Txgpqy2LszYkMf6J8njVAibvhM",
      "y": "-V4dS4UaLMgP_4fY4j8ir7cl1TXlFdAgcx55o7TkcSA"
    }
  }
}
```

See Appendix I.2 for an example of a signed Wallet Attestation JWT.

To use the Wallet Attestation towards the Authorization Server, the Wallet MUST generate a proof of possession according to Section 5.2 "Client Attestation PoP JWT" of Attestation-Based Client Authentication.

The `sub` claim of the Wallet Attestation JWT is the `client_id` of the Wallet instance. For privacy reasons, this value is the same across Wallet instances of that Wallet Provider, see Section 15.4.4 for more details.

Appendix F. Proof Types

A proof type communicates a proof of cryptographic key material used for binding a Credential in the Credential Request.

This specification defines the following proof types:

- `jwt`: A JWT [RFC7519] is used for proof of possession. When a `proofs` object is using a `jwt` proof type, it MUST include a `jwt` parameter with its value being a non-empty array of JWTs, where each JWT is formed as defined in Appendix F.1.
- `di_vp`: A W3C Verifiable Presentation object signed using the Data Integrity Proof [VC_Data_Integrity] as defined in [VC_DATA_2.0] or [VC_DATA] is used for proof of possession. When a `proofs` object is using a `di_vp` proof type, it MUST include an `di_vp` parameter with its value being a non-empty array of W3C Verifiable Presentations as defined by [VC_DATA_2.0] or [VC_DATA], where each of these W3C Verifiable Presentation is formed as defined in Appendix F.2.
- `attestation`: A JWT [RFC7519] representing a key attestation without using a proof of possession of the cryptographic key material that is being attested. When a `proofs` object is using an `attestation` proof type, the object MUST include an `attestation` parameter with its value being an array that contains exactly one JWT that is formed as defined in Appendix D.1.

There are two ways to convey key attestations (as defined in Appendix D) of the cryptographic key material during Credential issuance:

- The Wallet uses the `jwt` proof type in the Credential Request to create a proof of possession for one of the attested keys and adds the key attestation in the JOSE header.
- The Wallet uses the `attestation` proof type in the Credential Request to provide a key attestation without a proof of possession of any of the keys.

Depending on the Wallet's implementation, the `attestation` may avoid unnecessary End-User interaction during Credential issuance, as the key(s) to which the Credential(s) will be bound does not necessarily need to perform signature operations, and one key attestation can be used to attest multiple keys.

Additional proof types MAY be defined and used.

F.1. `jwt` Proof Type

The JWT MUST contain the following elements:

- in the JOSE header,
  - `alg`: REQUIRED. A digital signature algorithm identifier such as per IANA "JSON Web Signature and Encryption Algorithms" registry [IANA.JOSE]. It MUST NOT be `none` or an identifier for a symmetric algorithm (MAC).
  - `typ`: REQUIRED. MUST be `openid4vci-proof+jwt`, which explicitly types the key proof JWT as recommended in Section 3.11 of [RFC8725].
  - `kid`: OPTIONAL. JOSE Header containing the key ID. If the Credential is to be bound to a DID, the `kid` refers to a DID URL which identifies a particular key in the DID Document that the Credential is to be bound to. It MUST NOT be present if `jwk` or `x5c` is present.
  - `jwk`: OPTIONAL. JOSE Header containing the key material the new Credential is to be bound to. It MUST NOT be present if `kid` or `x5c` is present.
  - `x5c`: OPTIONAL. JOSE Header containing at least one certificate where the first certificate contains the key that the Credential is to be bound to, additional certificates may also be present. It MUST NOT be present if `kid` or `jwk` is present.
  - `key_attestation`: OPTIONAL. JOSE Header containing a key attestation as described in Appendix D. If the Credential Issuer provided a `c_nonce`, the `nonce` claim in the key attestation MUST be set to a server-provided `c_nonce`.
  - `trust_chain`: OPTIONAL. JOSE Header containing an [OpenID.Federation] Trust Chain. This element MAY be used to convey key attestation, metadata, metadata policies, federation Trust Marks and any other information related to a specific federation, if available in the chain. When used for signature verification, the header parameter `kid` MUST be present.

- in the JWT body,
  - `iss`: OPTIONAL (string). The value of this claim MUST be the `client_id` of the Client making the Credential request. This claim MUST be omitted if the access token authorizing the issuance call was obtained from a Pre-Authorized Code Flow through anonymous access to the token endpoint.
  - `aud`: REQUIRED (string). The value of this claim MUST be the Credential Issuer Identifier.
  - `iat`: REQUIRED (number). The value of this claim MUST be the time at which the key proof was issued using the syntax defined in [RFC7519].
  - `nonce`: OPTIONAL (string). The value type of this claim MUST be a string, where the value is a server-provided `c_nonce`. It MUST be present when the issuer has a Nonce Endpoint as defined in Section 7.

The Credential Issuer MUST validate that the JWT used as a proof is actually signed by a key identified in the JOSE Header through either `kid`, `jwk` or `x5c` element.

The Credential Issuer SHOULD issue a Credential for each cryptographic public key specified in the `attested_keys` claim within the `key_attestation` parameter.

Cryptographic algorithm identifiers used in the `proof_signing_alg_values_supported` Credential Issuer metadata parameter for this proof type are case sensitive strings and SHOULD be one of the Algorithm Names defined in [IANA.JOSE]. If Credential Issuer metadata is provided, the `alg` JWT header of the key proof, and if present, the `alg` JOSE headers of both `key_attestation` and `trust_chain`, MUST match one of the values listed in the `proof_signing_alg_values_supported` metadata parameter.

Below is a non-normative example of a `proofs` parameter (with line breaks within values for display purposes only):

```
{
  "jwt": [
    "eyJ0eXAiOiJvcGVuaWQ0dmNpLXByb29mK2p3dCIsImFsZyI6IkVTMjU2Iiwiand
    rIjp7Imt0eSI6IkVDIiwiY3J2IjoiUC0yNTYiLCJ4IjoiblVXQW9BdjNYWml0aDh
    FN2kxOU9kYXhPTFlGT3dNLVoyRXVNMDJUaXJUNCIsInkiOiJIc2tIVThCalVpMVU
    5WHFpN1N3bWo4Z3dBS18weGtjRGpFV183MVNvc0VZIn19.eyJhdWQiOiJodHRwcz
    ovL2NyZWRlbnRpYWwtaXNzdWVyLmV4YW1wbGUuY29tIiwiaWF0IjoxNzAxOTYwND
    Q0LCJub25jZSI6IkxhclJHU2JtVVBZdFJZTzZCUTR5bjgifQ.-a3EDsxClUB4O3L
    eDD5DVGEnNMT01FCQW4P6-2-BNBqc_Zxf0Qw4CWayLEpqkAomlkLb9zioZoipdP-
    jvh1WlA"
  ]
}
```

where the decoded JWT looks like this:

```
{
  "typ": "openid4vci-proof+jwt",
  "alg": "ES256",
  "jwk": {
    "kty": "EC",
    "crv": "P-256",
    "x": "nUWAoAv3XZith8E7i19OdaxOLYFOwM-Z2EuM02TirT4",
    "y": "HskHU8BjUi1U9Xqi7Swmj8gwAK_0xkcDjEW_71SosEY"
  }
}.{
  "aud": "https://credential-issuer.example.com",
  "iat": 1701960444,
  "nonce": "LarRGSbmUPYtRYO6BQ4yn8"
}
```

Here is another example of a JWT not only proving possession of a private key but also providing key attestation data for that key:

```
{
  "typ": "openid4vci-proof+jwt",
  "alg": "ES256",
  "kid": "0",
  "key_attestation": <key attestation in JWT format>
}.
{
  "iss": "s6BhdRkqt3",
  "aud": "https://server.example.com",
  "iat": 1659145924,
  "nonce": "tZignsnFbp"
}
```

F.2. `di_vp` Proof Type

When a W3C Verifiable Presentation as defined by [VC_DATA_2.0] or [VC_DATA] secured using Data Integrity [VC_Data_Integrity] is used as key proof, it MUST contain at least the following properties, in addition to any other properties required by [VC_DATA_2.0] or [VC_DATA]:

- `holder`: OPTIONAL. If present, it MUST be equivalent to the controller identifier part (e.g., DID) of the `verificationMethod` value identified by the `proof.verificationMethod` property.
- `proof`: REQUIRED. The `proof` property of the W3C Verifiable Presentation MUST be a Data Integrity Proof, as defined in [VC_Data_Integrity], and its properties MUST conform with the following rules, in addition to those specified in [VC_Data_Integrity]:
  - `cryptosuite`: REQUIRED. If Credential Issuer metadata is provided, the value MUST match one of the entries in the `proof_signing_alg_values_supported` metadata parameter.
  - `proofPurpose`: REQUIRED. MUST be set to `authentication`.
  - `domain`: REQUIRED. MUST be set to the Credential Issuer Identifier.
  - `challenge`: REQUIRED when the Credential Issuer has provided a `c_nonce`. It MUST NOT be used otherwise. String, where the value is a server-provided `c_nonce`. It MUST be present when the issuer has a Nonce Endpoint as defined in Section 7.

The Credential Issuer MUST validate that the W3C Verifiable Presentation used as a proof is actually signed with a key in the possession of the Holder.

Additional properties in the W3C Verifiable Presentation and Data Integrity Proof MUST be ignored if not understood. Cryptographic algorithm identifiers used in the `proof_signing_alg_values_supported` Credential Issuer metadata parameter for this proof type are case sensitive strings and SHOULD be one of those defined in, or referenced by, [VC_Data_Integrity].

Below is a non-normative example of a `proofs` parameter (with line breaks within values for display purposes only):

```
{
  "di_vp": [
    {
      "@context": [
        "https://www.w3.org/ns/credentials/v2",
        "https://www.w3.org/ns/credentials/examples/v2"
      ],
      "type": [
        "VerifiablePresentation"
      ],
      "holder": "did:key:z6MkvrFpBNCoYewiaeBLgjUDvLxUtnK5R6mqh5XPvLsrPsro",
      "proof": [
        {
          "type": "DataIntegrityProof",
          "cryptosuite": "eddsa-2022",
          "proofPurpose": "authentication",
          "verificationMethod": "did:key:z6MkvrFpBNCoYewiaeBLgjUDvLxUtnK5R6mqh5
           XPvLsrPsro#z6MkvrFpBNCoYewiaeBLgjUDvLxUtnK5R6mqh5XPvLsrPsro",
          "created": "2023-03-01T14:56:29.280619Z",
          "challenge": "82d4cb36-11f6-4273-b9c6-df1ac0ff17e9",
          "domain": "did:web:audience.company.com",
          "proofValue": "z5hrbHzZiqXHNpLq6i7zePEUcUzEbZKmWfNQzXcUXUrqF7bykQ7ACi
           WFyZdT2HcptF1zd1t7NhfQSdqrbPEjZceg7"
        }
      ]
    }
  ]
}
```

F.3. `attestation` Proof Type

A key attestation in JWT format as defined in Appendix D.1.

If the Credential Issuer has a Nonce Endpoint (as defined in Section 7), the `c_nonce` value provided by the Credential Issuer MUST be provided in the key attestation's `nonce` parameter.

Cryptographic algorithm identifiers used in the `proof_signing_alg_values_supported` Credential Issuer metadata parameter for this proof type are case sensitive strings and SHOULD be one of those defined in [IANA.JOSE].

Below is a non-normative example of a `proofs` parameter (with line breaks within values for display purposes only):

```
{
  "attestation": [
    "<key attestation in JWT format>"
  ]
}
```

The Credential Issuer SHOULD issue a Credential for each cryptographic public key specified in the `attested_keys` claim within the `key_attestation` parameter.

If Credential Issuer metadata is provided, the value of the `alg` JWT header of the key attestation MUST match one of the entries in the `proof_signing_alg_values_supported` metadata parameter.

F.4. Verifying Proof

To validate a key proof, the Credential Issuer MUST ensure that:

- all required claims for that proof type are contained as defined in Appendix F,
- the key proof is explicitly typed using header parameters as defined for that proof type,
- the header parameter indicates a registered asymmetric digital signature algorithm, `alg` parameter value is not none, is supported by the application, and is acceptable per local policy,
- the signature on the key proof verifies with the public key contained in the header parameter,
- the header parameter does not contain a private key,
- if the server has a Nonce Endpoint, the nonce in the key proof matches the server-provided `c_nonce` value,
- the creation time of the JWT, as determined by either the issuance time, or a server managed timestamp via the nonce claim, is within an acceptable window (see Section 13.8).

These checks may be performed in any order.

Appendix G. IANA Considerations

G.1. OAuth URI Registry

This specification registers the following URN in the IANA "OAuth URI" registry [IANA.OAuth.Parameters] established by [RFC6755].

G.1.1. urn:ietf:params:oauth:grant-type:pre-authorized_code

- URN: `urn:ietf:params:oauth:grant-type:pre-authorized_code`
- Common Name: Pre-Authorized Code
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 4.1.1 of this specification

G.2. OAuth Parameters Registry

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

G.2.1. issuer_state

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

G.2.2. pre-authorized_code

- Name: `pre-authorized_code`
- Parameter Usage Location: token request

- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 6.1 of this specification

G.2.3. tx_code

- Name: `tx_code`
- Parameter Usage Location: token request
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 6.1 of this specification

G.3. OAuth Authorization Server Metadata Registry

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

G.3.1. pre-authorized_grant_anonymous_access_supported

- Metadata Name: `pre-authorized_grant_anonymous_access_supported`
- Metadata Description: Boolean indicating whether Credential Issuer accepts Token Request with Pre-Authorized Code but without `client_id`
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 12.3 of this specification

G.4. OAuth Dynamic Client Registration Metadata Registry

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

G.4.1. credential_offer_endpoint

- Client Metadata Name: `credential_offer_endpoint`
- Client Metadata Description: Credential Offer Endpoint
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 12.1 of this specification

G.5. Well-Known URI Registry

This specification registers the following well-known URI in the IANA "Well-Known URI" registry [IANA.Well-Known.URIs] established by [RFC8615].

G.5.1. .well-known/openid-credential-issuer

- URI suffix: `openid-credential-issuer`
- Change controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Specification document: Section 12.2.2 of this specification
- Related information: (none)

G.6. Media Types Registry

This specification registers the following media type [RFC2046] in the IANA "Media Types" registry [IANA.MediaTypes] in the manner described in [RFC6838].

G.6.1. application/openid4vci-proof+jwt

- Type name: `application`
- Subtype name: `openid4vci-proof+jwt`
- Required parameters: n/a
- Optional parameters: n/a
- Encoding considerations: Uses JWS Compact Serialization, as specified in [RFC7515].
- Security considerations: See the Security Considerations in [RFC7519].
- Interoperability considerations: n/a
- Published specification: Appendix F.1 of this specification
- Applications that use this media type: Applications that issue and store verifiable credentials
- Additional information:
  - Magic number(s): n/a
  - File extension(s): n/a
  - Macintosh file type code(s): n/a
- Person & email address to contact for further information: Torsten Lodderstedt, torsten@lodderstedt.net
- Intended usage: COMMON
- Restrictions on usage: none
- Author: Torsten Lodderstedt, torsten@lodderstedt.net
- Change controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Provisional registration? No

G.6.2. application/key-attestation+jwt

- Type name: `application`
- Subtype name: `key-attestation+jwt`
- Required parameters: n/a
- Optional parameters: n/a
- Encoding considerations: Uses JWS Compact Serialization, as specified in [RFC7515].
- Security considerations: See the Security Considerations in [RFC7519].
- Interoperability considerations: n/a
- Published specification: Appendix D.1 of this specification
- Applications that use this media type: Applications that use the key attestation format defined in this specification
- Additional information:
  - Magic number(s): n/a
  - File extension(s): n/a
  - Macintosh file type code(s): n/a

- Person & email address to contact for further information: Torsten Lodderstedt, torsten@lodderstedt.net
- Intended usage: COMMON
- Restrictions on usage: none
- Author: Torsten Lodderstedt, torsten@lodderstedt.net
- Change controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Provisional registration? No

G.6.3. application/openidvci-issuer-metadata+jwt

- Type name: `application`
- Subtype name: `openidvci-issuer-metadata+jwt`
- Required parameters: n/a
- Optional parameters: n/a
- Encoding considerations: Uses JWS Compact Serialization, as specified in [RFC7515].
- Security considerations: See the Security Considerations in [RFC7519].
- Interoperability considerations: n/a
- Published specification: Section 12.2.3 of this specification
- Applications that use this media type: Applications that use signed metadata format defined in this specification
- Additional information:
  - Magic number(s): n/a
  - File extension(s): n/a
  - Macintosh file type code(s): n/a
- Person & email address to contact for further information: Torsten Lodderstedt, torsten@lodderstedt.net
- Intended usage: COMMON
- Restrictions on usage: none
- Author: Torsten Lodderstedt, torsten@lodderstedt.net
- Change controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Provisional registration? No

G.7. Uniform Resource Identifier (URI) Schemes Registry

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

G.7.1. openid-credential-offer

- URI Scheme: openid-credential-offer
- Description: Custom scheme used for credential offers
- Status: Permanent
- Well-Known URI Support: -
- Change Controller: OpenID Foundation Digital Credentials Protocols Working Group - openid-specs-digital-credentials-protocols@lists.openid.net
- Reference: Section 12.1.1 of this specification

Appendix H. Use Cases

This is a non-exhaustive list of sample use cases.

H.1. Credential Offer - Same-Device

While browsing the university's home page, the End-User finds a link "request your digital diploma". The End-User clicks on this link and is redirected to a digital Wallet. The Wallet notifies the End-User that a Credential Issuer offered to issue a diploma Credential. The End-User confirms this inquiry and is taken to the university's Credential issuance service's End-User experience. Upon successful authentication at the university and consent to the issuance of a digital diploma, the End-User is redirected back to the Wallet. Here, the End-User can verify the successful creation of the digital diploma.

H.2. Credential Offer - Cross-Device (with Information Pre-Submitted by the End-User)

The End-User is starting a job with a new employer. The employer requests the End-User to upload specific documents to the employee portal. After a few days, the End-User receives an email from the employer indicating that the employee Credential is ready to be claimed and provides instructions to scan a presented QR code for its retrieval. The End-User scans the QR code with a smartphone, which opens the Wallet. Meanwhile, the End-User has received a text message with a Transaction Code to the smartphone. After entering the Transaction Code in the Wallet for security reasons, the End-User approves the Credential issuance and receives the Credential in the Wallet.

H.3. Credential Offer - Cross-Device & Deferred

The End-User intends to acquire a digital criminal record. This involves a visit to the local administration's office to request the official criminal record be issued as a digital Credential. After presenting an ID document, the End-User is prompted to scan a QR code using the Wallet and is informed that the issuance of the Credential will require some time, due to necessary background checks by the authority.

While using the Wallet, the End-User notices an indication that the issuance of the digital record is in progress. After a few days, the End-User receives a notification from the Wallet indicating that the requested Credential was successfully issued. Upon opening the Wallet, the End-User is queried about the download of the Credential. After confirmation, the Wallet fetches and saves the new Credential.

H.4. Wallet-Initiated Issuance during Presentation

An End-User comes across a Verifier app that is requesting the End-User to present a Credential, e.g., a driving license. The Wallet determines the requested Credential type(s) from the presentation request and notifies the End-User that there is currently no matching Credential in the Wallet. The Wallet selects a Credential Issuer capable of issuing the missing Credential and, upon End-User consent, sends the End-User to the Credential Issuer's End-User experience (website or app). Once authenticated and consent is provided for the issuance of the Credential into the Wallet, the End-User is redirected back to the Wallet. The Wallet informs the End-User that Credential was successfully issued into the Wallet and is ready to be presented to the Verifier app that originally requested presentation of that Credential.

H.5. Wallet-Initiated Issuance during Presentation (Requires Presentation of Additional

Credentials During Issuance)

An End-User comes across a Verifier app that is requesting the End-User to present a Credential, e.g., a university diploma. The Wallet determines the requested Credential type(s) from the presentation request and notifies the End-User that there is currently no matching Credential in the Wallet. The Wallet then offers the End-User a list of Credential Issuers, which might be based on a Credential Issuer list curated by the Wallet provider. The End-User selects the university of graduation and is subsequently redirected to the corresponding university's website or app.

The End-User logs into the university, which identifies that the corresponding End-User account is not yet verified. Among various identification options, the End-User opts to present a Credential from the Wallet. The End-User is redirected back to the Wallet to consent to present the requested Credential(s) to the university. Following this, the End-User is redirected back to the university End-User experience. Based on the presented Credential, the university finalizes the End-User verification, retrieves data about the End-User from its database, and proposes to issue a diploma as a Verifiable Credential.

Upon providing consent, the End-User is sent back to the Wallet. The Wallet informs the End-User that the Credential was successfully issued into the Wallet and is ready to be presented to the Verifier app that originally requested presentation of that Credential.

H.6. Wallet-Initiated Issuance after Installation

The End-User installs a new Wallet and opens it. The Wallet offers the End-User a selection of Credentials that the End-User may obtain from a Credential Issuer, e.g., a national identity Credential, a mobile driving license, or a public transport ticket. The corresponding Credential Issuers (and their URLs) are pre-configured by the Wallet or follow some discovery processes that are out of scope for this specification. By clicking on one of these options corresponding to the Credentials available for issuance, the issuance process starts using a flow supported by the Credential Issuer (Pre-Authorized Code flow or Authorization Code flow).

Wallet Providers may also provide a market place where Issuers can register to be found for Wallet-initiated flows.

Appendix I. Additional Examples

I.1. Credential Issuer Metadata

This section contains additional examples for Credential issuer Metadata

The following is a non-normative example of Credential Issuer metadata of a Credential in the IETF SD-JWT VC [I-D.ietf-oauth-sd-jwt-vc] format in signed form (JWT):

```
eyJhbGciOiJFUzI1NiIsInR5cCI6Im9wZW5pZHZjaS1pc3N1ZXItbWV0YWRhdGErand0Ii
wieDVjIjpbIk1JSUI5RENDQVpxZ0F3SUJBZ0lVRkFIcFd2VjdOR1J4T05JSFQvQ0N3RUU5
QmE0d0NnWUlLb1pJemowRUF3SXdLREVtTUNRR0ExVUVBd3dkWTNKbFpHVnVkR2xoYkMxcG
MzTjFaWEl1WlhoaGJYQnNaUzVqYjIwd0lCY05NalV3TnpJeU1EZ3lPREEyV2hnUE1qQTFO
VEEzTVRVd09ESTRNRFphTUNneEpqQWtCZ05WQkFNTUhXTnlaV1JsYm5ScFlXd3RhWE56ZF
dWeUxtVjRZVzF3YkdVdVkyOXRNRmt3RXdZSEtvWkl6ajBDQVFZSUtvWkl6ajBEQVFjRFFn
QUVsSG50aWhHdTV2Q3RrWituMllhQ28rdW5ubndrd2xVUHBUc25PMk9TdnpvVHNYdGQxaz
E4WWhBUGhPTjYvbERXei9mN0FNcmRWN0xNZ0JZU3BwMndKS09CbnpDQm5EQWRCZ05WSFE0
RUZnUVVhTWo0bmEvUWpjdHA3V2dyR2xxam0zUFQxU2t3SHdZRFZSMGpCQmd3Rm9BVWFNaj
RuYS9RamN0cDdXZ3JHbHFqbTNQVDFTa3dEd1lEVlIwVEFRSC9CQVV3QXdFQi96QkpCZ05W
SFJFRVFqQkFnaDFqY21Wa1pXNTBhV0ZzTFdsemMzVmxjaTVsZUdGdGNHeGxMbU52YllJZk
tpNWpjbVZrWlc1MGFXRnNMV2x6YzNWbGNpNWxlR0Z0Y0d4bExtTnZiVEFLQmdncWhrak9Q
UVFEQWdOSUFEQkZBaUJMWjNuTGhMT1diR2hBTGNSUHhQTGlIdFV1bGRqRFQ0MUZrOTBUan
d5MEpRSWhBT3NtWXpTcHNGenBqTjBPV1R2UEV5dURrL05nUnVmSjNkRGFnOHNaSjZkTSJd
```

```
fQ.eyJzdWIiOiJodHRwczovL2NyZWRlbnRpYWwtaXNzdWVyLmV4YW1wbGUuY29tIiwiaWF
0IjoxNTE2MjM5MDIyLCJjcmVkZW50aWFsX2lzc3VlciI6Imh0dHBzOi8vY3JlZGVudGlhb
C1pc3N1ZXIuZXhhbXBsZS5jb20iLCJhdXRob3JpemF0aW9uX3NlcnZlcnMiOlsiaHR0cHM
6Ly9zZXJ2ZXIuZXhhbXBsZS5jb20iXSwiY3JlZGVudGlhbF9lbmRwb2ludCI6Imh0dHBzO
i8vY3JlZGVudGlhbC1pc3N1ZXIuZXhhbXBsZS5jb20vY3JlZGVudGlhbCIsImRlZmVycmV
kX2NyZWRlbnRpYWxfZW5kcG9pbnQiOiJodHRwczovL2NyZWRlbnRpYWwtaXNzdWVyLmV4Y
W1wbGUuY29tL2RlZmVycmVkX2NyZWRlbnRpYWwiLCJjcmVkZW50aWFsX3Jlc3BvbnNlX2V
uY3J5cHRpb24iOnsiYWxnX3ZhbHVlc19zdXBwb3J0ZWQiOlsiRUNESC1FUyJdLCJlbmNfd
mFsdWVzX3N1cHBvcnRlZCI6WyJBMTI4R0NNIl0sImVuY3J5cHRpb25fcmVxdWlyZWQiOmZ
hbHNlfSwiZGlzcGxheSI6W3sibmFtZSI6IkV4YW1wbGUgVW5pdmVyc2l0eSIsImxvY2FsZ
SI6ImVuLVVTIn0seyJuYW1lIjoiRXhhbXBsZSBVbml2ZXJzaXTDqSIsImxvY2FsZSI6ImZ
yLUZSIn1dLCJjcmVkZW50aWFsX2NvbmZpZ3VyYXRpb25zX3N1cHBvcnRlZCI6eyJTRF9KV
1RfVkNfZXhhbXBsZV9pbl9PcGVuSUQ0VkNJIjp7ImZvcm1hdCI6ImRjK3NkLWp3dCIsInN
jb3BlIjoiU0RfSldUX1ZDX2V4YW1wbGVfaW5fT3BlbklENFZDSSIsImNyeXB0b2dyYXBoa
WNfYmluZGluZ19tZXRob2RzX3N1cHBvcnRlZCI6WyJqd2siXSwiY3JlZGVudGlhbF9zaWd
uaW5nX2FsZ192YWx1ZXNfc3VwcG9ydGVkIjpbIkVTMjU2Il0sInByb29mX3R5cGVzX3N1c
HBvcnRlZCI6eyJqd3QiOnsicHJvb2Zfc2lnbmluZ19hbGdfdmFsdWVzX3N1cHBvcnRlZCI
6WyJFUzI1NiJdLCJrZXlfYXR0ZXN0YXRpb25zX3JlcXVpcmVkIjp7ImtleV9zdG9yYWdlI
jpbImlzb18xODA0NV9tb2RlcmF0ZSJdLCJ1c2VyX2F1dGhlbnRpY2F0aW9uIjpbImlzb18
xODA0NV9tb2RlcmF0ZSJdfX19LCJ2Y3QiOiJTRF9KV1RfVkNfZXhhbXBsZV9pbl9PcGVuS
UQ0VkNJIiwiY3JlZGVudGlhbF9tZXRhZGF0YSI6eyJkaXNwbGF5IjpbeyJuYW1lIjoiSWR
lbnRpdHlDcmVkZW50aWFsIiwibG9nbyI6eyJ1cmkiOiJodHRwczovL3VuaXZlcnNpdHkuZ
XhhbXBsZS5lZHUvcHVibGljL2xvZ28ucG5nIiwiYWx0X3RleHQiOiJhIHNxdWFyZSBsb2d
vIG9mIGEgdW5pdmVyc2l0eSJ9LCJsb2NhbGUiOiJlbi1VUyIsImJhY2tncm91bmRfY29sb
3IiOiIjMTIxMDdjIiwidGV4dF9jb2xvciI6IiNGRkZGRkYifV0sImNsYWltcyI6W3sicGF
0aCI6WyJnaXZlbl9uYW1lIl0sImRpc3BsYXkiOlt7Im5hbWUiOiJHaXZlbiBOYW1lIiwib
G9jYWxlIjoiZW4tVVMifSx7Im5hbWUiOiJWb3JuYW1lIiwibG9jYWxlIjoiZGUtREUifV1
9LHsicGF0aCI6WyJmYW1pbHlfbmFtZSJdLCJkaXNwbGF5IjpbeyJuYW1lIjoiU3VybmFtZ
SIsImxvY2FsZSI6ImVuLVVTIn0seyJuYW1lIjoiTmFjaG5hbWUiLCJsb2NhbGUiOiJkZS1
ERSJ9XX0seyJwYXRoIjpbImVtYWlsIl19LHsicGF0aCI6WyJwaG9uZV9udW1iZXIiXX0se
yJwYXRoIjpbImFkZHJlc3MiXSwiZGlzcGxheSI6W3sibmFtZSI6IlBsYWNlIG9mIHJlc2l
kZW5jZSIsImxvY2FsZSI6ImVuLVVTIn0seyJuYW1lIjoiV29obnNpdHoiLCJsb2NhbGUiO
iJkZS1ERSJ9XX0seyJwYXRoIjpbImFkZHJlc3MiLCJzdHJlZXRfYWRkcmVzcyJdfSx7InB
hdGgiOlsiYWRkcmVzcyIsImxvY2FsaXR5Il19LHsicGF0aCI6WyJhZGRyZXNzIiwicmVna
W9uIl19LHsicGF0aCI6WyJhZGRyZXNzIiwiY291bnRyeSJdfSx7InBhdGgiOlsiYmlydGh
kYXRlIl19LHsicGF0aCI6WyJpc19vdmVyXzE4Il19LHsicGF0aCI6WyJpc19vdmVyXzIxI
l19LHsicGF0aCI6WyJpc19vdmVyXzY1Il19XX19fX0.KkM97TGKU9yTAPnbTfSPVFT3Ryw
X9_5QPEtNwMhiu8V_UPBGMYA2mkVc-VxGTm9vdeV482-TWnVfbub_kzS9Eg
```

The following is a non-normative example of Credential Issuer metadata of a Credential in the IETF SD-JWT VC [I-D.ietf-oauth-sd-jwt-vc] format with optional parameters present:

```
{
  "credential_issuer": "https://credential-issuer.example.com",
  "authorization_servers": [
    "https://server.example.com"
  ],
  "credential_endpoint": "https://credential-issuer.example.com/credential",
  "nonce_endpoint": "https://credential-issuer.example.com/nonce",
  "deferred_credential_endpoint": "https://credential-
issuer.example.com/deferred_credential",
  "notification_endpoint": "https://credential-issuer.example.com/notification",
  "credential_request_encryption": {
    "jwks": [
      {
        "kty":"EC", "kid":"ac", "use":"enc", "crv":"P-256", "alg":"ECDH-ES",
        "x":"YO4epjifD-KWeq1sL2tNmm36BhXnkJ0He-WqMYrp9Fk",
        "y":"Hekpm0zfK7C-YccH5iBjcIXgf6YdUvNUac_0At55Okk"
      }
    ],
    "enc_values_supported": ["A128GCM"],
    "zip_values_supported": ["DEF"],
    "encryption_required": true
  },
  "credential_response_encryption": {
    "alg_values_supported": ["ECDH-ES"],
    "enc_values_supported": ["A128GCM"],
    "zip_values_supported": ["DEF"],
    "encryption_required": true
  },
  "batch_credential_issuance": {
    "batch_size": 10
  },
  "display": [
    {
      "name": "Example University",
      "locale": "en-US",
      "logo": {
        "uri": "https://university.example.edu/public/logo.png",
        "alt_text":"a square logo of a university"
      }
    },
    {
      "name": "Example Université",
      "locale": "fr-FR",
      "logo": {
        "uri": "https://university.example.edu/public/logo.png",
        "alt_text":"Un logo universitaire carré"
      }
    }
  ],
  "credential_configurations_supported": {
    "SD_JWT_VC_example_in_OpenID4VCI": {
      "format": "dc+sd-jwt",
      "scope": "SD_JWT_VC_example_in_OpenID4VCI",
      "credential_signing_alg_values_supported": ["ES256"],
      "cryptographic_binding_methods_supported": ["jwk"],
      "proof_types_supported": {
        "jwt": {
          "proof_signing_alg_values_supported": ["ES256"],
          "key_attestations_required": {
            "key_storage": ["iso_18045_moderate"],
            "user_authentication": ["iso_18045_moderate"]
          }
        }
      },
      "vct": "SD_JWT_VC_example_in_OpenID4VCI",
      "credential_metadata": {
        "display": [
          {
            "name": "IdentityCredential",
            "locale": "en-US",
            "logo": {
              "uri": "https://university.example.edu/public/logo_credential.png",
              "alt_text": "a square logo of a university credential"
            },
            "description": "A credential that signals the membership of a university",
            "background_color": "#12107c",
            "text_color": "#FFFFFF"
          }
        ],
        "claims": [
          {
            "path": ["given_name"],
            "display": [
              {
                "name": "Given Name",
                "locale": "en-US"
              },
              {
                "name": "Vorname",
                "locale": "de-DE"
              }
            ]
          },
          {
            "path": ["family_name"],
            "display": [
              {
                "name": "Surname",
                "locale": "en-US"
              },
              {
                "name": "Nachname",
                "locale": "de-DE"
              }
            ]
          },
          {"path": ["email"]},
          {"path": ["phone_number"]},
          {
            "path": ["address"],
            "display": [
              {
                "name": "Place of residence",
                "locale": "en-US"
              },
              {
                "name": "Wohnsitz",
                "locale": "de-DE"
              }
            ]
          },
          {"path": ["address", "street_address"]},
          {"path": ["address", "locality"]},
          {"path": ["address", "region"]},
          {"path": ["address", "country"]},
          {"path": ["birthdate"]},
          {"path": ["is_over_18"]},
          {"path": ["is_over_21"]},
          {"path": ["is_over_65"]}
        ]
      }
    }
  }
}
```

I.2. Wallet Attestation JWT

The following is a non-normative example of a signed Wallet Attestation:

```
eyJ0eXAiOiJvYXV0aC1jbGllbnQtYXR0ZXN0YXRpb24rand0IiwiYWxnIjoiRVMyNTYiLC
J4NWMiOlsiTUlJQnl6Q0NBWEdnQXdJQkFnSVVKblorYXY3a0ZSYUNyS2xOaTRSU2dSTWxN
emN3Q2dZSUtvWkl6ajBFQXdJd0pqRWtNQ0lHQTFVRUF3d2JkMkZzYkdWMExYQnliM1pwWk
dWeUxtVjRZVzF3YkdVdVkyOXRNQ0FYRFRJMU1EZ3hNakU0TXpFek4xb1lEekl3TlRVd09E
QTFNVGd6TVRNM1dqQW1NU1F3SWdZRFZRUURQnQzWVd4c1pYUXRjSEp2ZG1sa1pYSXVaWG
hoYlhCc1pTNWpiMjB3V1RBVEJnY3Foa2pPUFFJQkJnZ3Foa2pPUFFNQkJ3TkNBQVFKREZp
WGZ0THp2bVFmVnduQytGVFVKSFlMZ0pWTVlQbXM3ZWlGYytEQ0EreFUxVTgzSms4WEtyMH
FrRGxzYlcxU3JVRXB0V0pvSmx0NU43S1dhdGl1bzNzd2VUQWRCZ05WSFE0RUZnUVVMNStl
WnVrY3kvTFFVN2VYdnN6V0tHVEEraWt3SHdZRFZSMGpCQmd3Rm9BVUw1K2VadWtjeS9MUV
U3ZVh2c3pXS0dUQStpa3dEd1lEVlIwVEFRSC9CQVV3QXdFQi96QW1CZ05WSFJFRUh6QWRn
aHQzWVd4c1pYUXRjSEp2ZG1sa1pYSXVaWGhoYlhCc1pTNWpiMjB3Q2dZSUtvWkl6ajBFQX
dJRFNBQXdSUUlnZkVVdkFWMzhSUDdCU1JDSllLK0UvTWVSU291amlmRTZBT2Z1aGVKVVZG
OENNJUURiZDduMytoRU5Rd2kyanY1SHhWUnoxSHJxSkhSOWlYUnNzSjJPSnhObHF3PT0iXX
0.eyJpc3MiOiJodHRwczovL3dhbGxldC1wcm92aWRlci5leGFtcGxlLmNvbSIsInN1YiI6
Imh0dHBzOi8vd2FsbGV0LmV4YW1wbGUuY29tIiwid2FsbGV0X25hbWUiOiJXYWxsZXQgU2
9sdXRpb24gWCBieSBXb25kZXJsYW5kIFN0YXRlIERlcGFydG1lbnQiLCJ3YWxsZXRfbGlu
ayI6Imh0dHBzOi8vZXhhbXBsZS5jb20vd2FsbGV0L2RldGFpbF9pbmZvLmh0bWwiLCJuYm
YiOjE3NTUwMjM1NDcsImV4cCI6MTc1NTEwMjc0NSwiY25mIjp7Imp3ayI6eyJrdHkiOiJF
QyIsInVzZSI6InNpZyIsImNydiI6IlAtMjU2IiwieCI6IjE4d0hMZUlnVzl3Vk42VkQxVH
hncHF5MkxzellrTWY2SjhualZBaWJ2aE0iLCJ5IjoiLVY0ZFM0VWFMTWdQXzRmWTRqOGly
N2NsMVRYbEZkQWdjeDU1bzdUa2NTQSJ9fX0.3rasFv7xis_VxGjm6xbyb7wjPuaEaCi0-Z
_8PzutOVTBVoPEq7dDzZT-I5gJlQrS0HY6gU_Z30bOnwhkYJQW9g
```

Appendix J.   Acknowledgements

We would like to thank Patrick Amrein, Masayoshi Arakawa, Richard Barnes, Paul Bastian, Vittorio Bertocci, Christian Bormann, John Bradley, Brian Campbell, Lee Campbell, Tim Cappalli, Lenah Chacha, David Chadwick, Stefan Charsley, Gabe Cohen, Thomas Darimont, Andrii Deinega, Giuseppe De Marco, Rajvardhan Deshmukh, Mark Dobrinic, Daniel Fett, Pedro Felix, George Fletcher, Christian Fries, Timo Glasta, Mark Haine, Martijn Haring, Fabian Hauck, Roland Hedberg, Joseph Heenan, Bjorn Hjelm, Alen Horvat, Andrew Hughes, Jacob Ideskog, Lukasz Jaromin, Edmund Jay, Michael B. Jones, Tom Jones, Judith Kahrer, Micha Kraus, Takahiko Kawasaki, Niels Klomp, Ronald Koenig, Micha Kraus, Markus Kreusch, Philipp Lehwalder, Adam Lemmon, Dave Longley, Hicham Lozi, David Luna, Daniel McGrogan, Jeremie Miller, Mirko Mollik, Kenichi Nakamura, Andres Olave, Gareth Oliver, Nemanja Patrnogic, Dima Postnikov, Rolson Quadras, Helen Quinn, Sami Rosendahl, Babis Routis, Nat Sakimura, Sudesh Shetty, Peter Sorotokin, Oliver Terbu, Dimitri James Tsiflitzis, Mike Varley, Niels van Dijk, Arjen van Veen, Jan Vereecken, David Waite, Jacob Ward, David Zeuthen for their valuable feedback and contributions to this specification.

Appendix K.   Notices

Authors' Addresses

Torsten Lodderstedt
SPRIND
Email: torsten@lodderstedt.net

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

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

Paul Bastian
Bundesdruckerei
Email: paul.bastian@bdr.de