Workgroup: OpenID Connect
Internet-Draft: openid-connect-self-issued-v2-1_0-12
Published: 1 January 2023
Intended Status: Standards Track
Authors: K. Yasuda *Microsoft*
M. Jones *Microsoft*
T. Lodderstedt *yes.com*

Self-Issued OpenID Provider v2

Abstract

OpenID Connect defines mechanisms by which an End-User can leverage an OpenID Provider (OP) to release identity information (such as authentication and claims) to a Relying Party (RP) which can act on that information. In this model, the RP trusts assertions made by the OP, i.e. the OP is the issuer of these assertions.

This specification extends OpenID Connect with the concept of a Self-Issued OpenID Provider (Self-Issued OP), an OP controlled by the End-User. The Self-Issued OP does not itself assert identity information about this End-user. Instead the End-user becomes the issuer of identity information. Using Self-Issued OPs, End-Users can authenticate themselves with Self-Issued ID Tokens signed with keys under the End-user's control and present self-attested claims directly to the RPs.

Self-Issued OPs can also present cryptographically verifiable claims issued by the third parties trusted by the RPs, when used with separate specifications such as [OpenID4VP], or Aggregated and Distributed Claims defined in Section 5.6.2 of [OpenID.Core]. This allows End-Users to interact with RPs, without RPs interacting directly with claims issuers.

Table of Contents

1. Introduction

This specification extends OpenID Connect with the concept of a *Self-Issued OpenID Provider* (Self-Issued OP), an OpenID Provider (OP) which is within the End-User's control. Using Self-Issued OPs, End-Users can authenticate themselves with Self-Issued ID Tokens and present self-attested claims directly to the RPs. Self-Issued OPs can also present cryptographically verifiable claims issued by the third parties trusted by the RPs, when used with separate specifications such as [OpenID4VP], or Aggregated and Distributed Claims defined in Section 5.6.2 of [OpenID.Core]. This allows End-Users to interact with RPs, without RPs interacting with claims issuers.

End-user control does not imply the Self-Issued OP is entirely locally hosted on an End-user's device. There are different ways to implement a Self-Issued OP: the Self-Issued OP can completely run on a End-user device; it might utilize cloud components; or it might completely run in the cloud.

The crucial difference between a traditional OP and the Self-Issued OP is that the Self-Issued OP allows the End-user to determine identifiers and claims released to the RP.

[OpenID.Core] defines that an OP releases End-User authentication information in the form of an ID Token. An RP will trust an ID Token based on the relationship between the RP and the issuer of this ID Token.

The extensions defined in this specification provide the protocol changes needed to support Self-Issued OpenID Provider model. Aspects not defined in this specification are expected to follow [OpenID.Core]. Most notably, a Self-Issued OP MAY implement all flows as specified in [OpenID.Core], e.g. the Authorization Code Flow, and OpenID Connect extension flows, such as [OpenID.CIBA], as permitted by its deployment model. If the Self-Issued OP is operated entirely locally on a user device, it might be unable to expose any endpoints beyond the authorization endpoint to the RPs. However, if the Self-Issued OP has cloud-based components, it MAY expose further endpoints, such as a token endpoint. The same is applicable for Dynamic Client Registration ([OpenID.Registration]).

This specification replaces Self-Issued OpenID Connect Provider DID Profile v0.1 and was written as a working item of a liaison between Decentralized Identity Foundation and OpenID Foundation.

1.1. Notable Differences between OpenID Connect Core and Self-Issued OP Models

In the traditional OpenID Connect model, when an OP acts as an ID Token issuer, it is common for the OP to have a legal stake with the RPs and a reputation-based stake with both RPs and End-Users to provide correct information. In the Self-Issued OP model, the RP's trust relationship is directly with the End-User. The Self-Issued OP allows the End-User to authenticate towards the RP with an identifier controlled by the End-User instead of an identifier assigned to the End-User by a third-party provided OP. An End-User controlled identifier might be a public key fingerprint or a Decentralized Identifier (see [DID-Core]). This changes the trust model and the way signatures of the Self-Issued ID Tokens are validated in comparison to the traditional OpenID Connect model.

In traditional OpenID Connect, the ID Token is signed by the OP as an entity, identified by the `iss` claim. The RP uses this identifier to obtain the key material to validate the ID Token's signature. This signature ensures the data is attested by the OP the RP trusts for that purpose and it also is an attestation of what service produced the ID Token (since both are the same entity).

In the Self-Issued OP case, the ID Token is self-signed with a private key under the user's control, identified by the `sub` claim. The RP uses this identifier to obtain the key material to validate the ID Token's signature. Unlike traditional OpenID Connect, this signature can no longer be used to cryptographically validate the software or service that created the ID Token. Self-issued ID Token can be detected when the `iss` value is set to the user identifier conveyed in the `sub` Claim, because from a conceptual perspective, the issuer of the ID Token is the user. This also aligns Self-Issued OP with the way self-signed certificates and W3C Verifiable Presentations handle subject and issuer of such certificates and assertions, respectively.

Because a Self-Issued OP within the End-User's control does not have the legal, reputational trust of a traditional OP, claims about the End-User (e.g., `birthdate`) included in a Self-Issued ID Token, are by default self-asserted and non-verifiable. A Self-Issued OP can also present cryptographically verifiable claims issued by the third-party sources trusted by the RP, as defined in separate specifications such as [OpenID4VP] or Aggregated and Distributed Claims in Section 5.6.2 of [OpenID.Core].

1.2. Terms and Definitions

Common terms in this document come from four primary sources: [OpenID.Core], [VC-DATA] and [DID-Core]. In the case where a term has a definition that differs, the definition below is authoritative.

Self-Issued OpenID Provider (Self-Issued OP) An OpenID Provider (OP) used by the End-users to prove control over a cryptographically Verifiable Identifier

Self-Issued Request Request to a Self-Issued OP from an RP

Self-Issued Response Response to an RP from a Self-Issued OP

Self-Issued ID Token ID Token signed using the key material controlled by the End-User. It is issued by a Self-Issued OP.

Cryptographically Verifiable Identifier An identifier that is either based upon or resolves to cryptographic key material that can be used to verify a signature on the ID Token or the Self-Issued Request.

Trust Framework A legally enforceable set of specifications, rules, and agreements that govern a multi-party system established for a common purpose, designed for conducting specific types of transactions among a community of participants, and bound by a common set of requirements, as defined in OIX.

Verifiable Credential (VC) A verifiable credential is a tamper-evident credential that has authorship that can be cryptographically verified. Verifiable credentials can be used to build verifiable presentations, which can also be cryptographically verified. The claims in a credential can be about different subjects. See [VC-DATA].

Wallet Entity that receives, stores, presents, and manages Credentials and key material of the End-User. There is no single deployment model of a Wallet: Credentials and keys can both be stored/managed locally by the end-user, or by using a remote self-hosted service, or a remote third party service. In the context of this specification, the Wallet acts as an Self-Issued OpenID Provider towards the RP.

Base64url Encoding Base64 encoding using the URL- and filename-safe character set defined in Section 5 of [RFC4648], with all trailing '=' characters omitted (as permitted by Section 3.2 of [RFC4648]) and without the inclusion of any line breaks, whitespace, or other additional characters. Note that the base64url encoding of the empty octet sequence is the empty string. (See Appendix C of [RFC7515] for notes on implementing base64url encoding without padding.)

1.3. Abbreviations

- OP: OpenID Provider
- RP: Relying Party
- Self-Issued OP or SIOP: Self-Issued OpenID Provider

2. Scope

2.1. In Scope

This specification extends the OpenID Connect Core in the following ways:

- **Invocation of a Self-Issued OP**: mechanisms for how the RP invokes/opens a Self-Issued OP.
- **Obtaining Self-Issued OP Metadata (Static and Dynamic Discovery)**: mechanisms for how the RP discovers Self-issued OP's metadata such as authorization endpoint.
- **Obtaining RP Metadata**: mechanisms for how the Self-Issued OPs obtain RP metadata both with or without RP pre-registering with the Self-Issued OP.
- **Self-Issued ID Token**: defines additional claims and processing requirements of ID Tokens issued by Self-Issued OPs. An ID Token is self-issued if the values of the `iss` and the `sub` claims are the same.
- **Support for Self-Asserted Claims**: transporting claims in a Self-Issued ID Token that are not verifiable by the RP
- **Support for Cryptographically Verifiable Identifiers**: an identifier that is either based upon or can be resolved to cryptographic key material and is used by the Self-Issued OPs in the `sub` Claim of the ID Token to prove possession of the cryptographic key used to sign the ID Token. Types of such identifiers are defined in this specification as Subject Syntax Types.

2.2. Out of Scope

The following are considered out of scope of this document.

2.2.1. Presentation of Claims from the Third-Party Issuers

A Self-Issued OP can present self-attested claims in the ID Token.

Methods to present of cryptographically verifiable claims issued by trusted third-party sources is defined in other specifications, such as [OpenID4VP], which extends OAuth 2.0 to enable presentation of Verifiable Credentials, supporting W3C Verifiable Credentials and ISO/IEC 18013-5:2021 mdoc as well as other credential formats.

2.3. Relationship with Section 7 of *OpenID.Core* Self-Issued OpenID Provider

This specification extends Section 7 of [OpenID.Core] Self-Issued OpenID Provider in the following ways:

- Added support for Decentralized Identifiers defined in [DID-Core] as Cryptographically Verifiable Identifiers in addition to the JWK thumbprint defined in Self-Issued OP v2. See Section 8.
- Added support for Cross-Device Self-Issued OP model. See Section 4.
- Extended Relying Party Registration mechanisms to support pre-registration and dynamic registration for not-pre-registered RPs. See Section 7.
- Added support for Dynamic Self-Issued OpenID Provider Discovery. See Section 6.1.
- Added support for claimed URLs (universal links, app links) in addition to the custom URL schemas as Self-Issued OP `authorization_endpoint`. See Section 6.2.
- Allows use of any OpenID Connect flow for Self-Issued OPs and Dynamic Client Registration

Note that while this specification extends the original Section 7 of [OpenID.Core] Self-Issued OpenID Provider, some sections of it could be applicable more generally to the entire OpenID Connect Core specification.

3. Protocol Flow

Self-Issued Request results in Self-Issued OP returning an ID Token to the Relying Party when the End-User authentication succeeds and the End-User provides necessary permission. The ID Token always includes claims about the Authentication event and MAY include additional claims about the End-User.

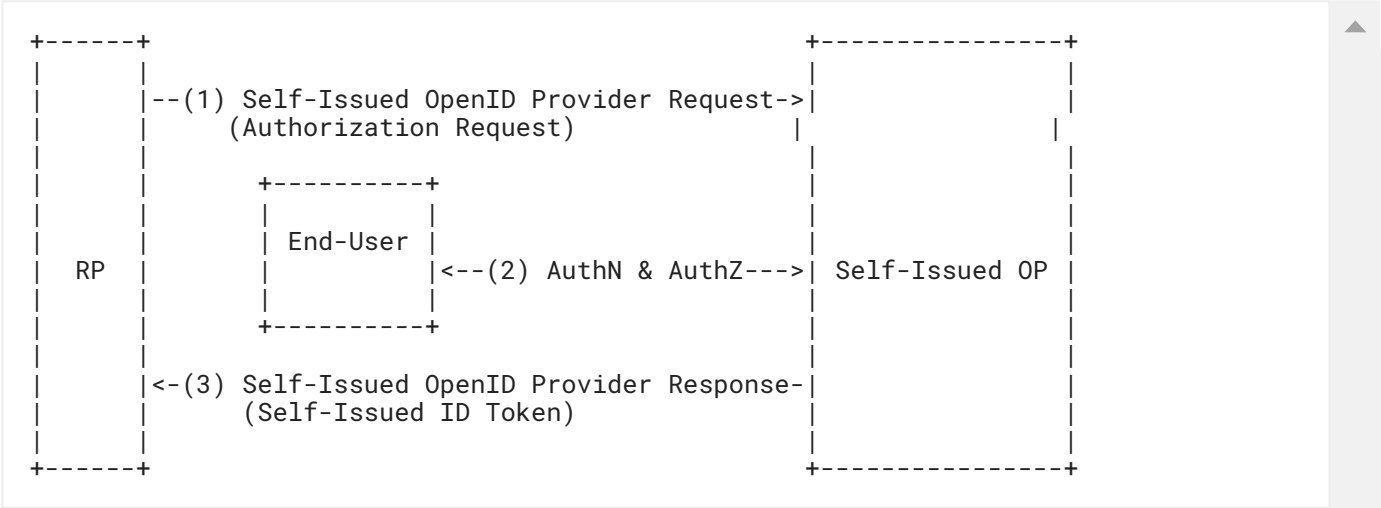

Figure 1: Self-Issued OP Protocol Flow

4. Cross-Device Self-Issued OP

There are two models of Self-Issued OP protocol flows:

- Same-Device Self-Issued OP model: Self-Issued OP is on the same device on which the End-User's user interactions are occurring. The RP might be a Web site on a different machine and still use the same-device Self-Issued OP protocol flow for authentication.
- Cross-device Self-Issued OP model: Self-Issued OP is on a different device than the one on which the End-User's user interactions are occurring.

This section outlines how Self-Issued OP is used in cross-device scenarios, and its differences with the same device model. In contrast to same-device scenarios, neither RP nor Self-Issued OP can communicate to each other via HTTP redirects through a user agent. The protocol flow is therefore modified as follows:

1. The RP prepares a Self-Issued Request and renders it as a QR code.
2. The End-User scans the QR code with her smartphone's camera app.
3. Self-Issued OP is invoked on the smartphone (custom URL scheme or claimed URLs).
4. The Self-Issued OP processes the Authorization Request.
5. Upon completion of the Authorization Request, the Self-Issued OP directly sends a HTTP POST request with the Authorization Response to an endpoint exposed by the RP.

The request in Step 5 is not a form post request where the Self-Issued OP would respond to a user agent with a form, which automatically triggers a POST request to the RP. The Self-Issued OP sends this request directly to the RP's endpoint.

For brevity, mainly the QR code method is discussed as a mechanism to initiate a cross-device protocol flow throughout this specification. However, other mechanisms to initiate a cross-device flow are possible.

5. Self-Issued OpenID Provider Invocation

When the End-user first interacts with the RP, there are currently no established, robust means for the RP to reliably determine a URI of the Self-Issued OP an End-user may have a relationship with or have installed. The RP is, therefore, responsible for selecting where to direct the request URL.

When the RP sends the request to the Self-Issued OP, there are two scenarios of how to reach and invoke an application that can process that request.

In the first scenario, the request is encoded in a QR code or a deep link, and the End-user scans it with the camera via the application that is intended to handle the request from the RP. In this scenario, the request does not need to be intended for a specific `authorization_endpoint` of a Self-Issued OP. Note that a QR code option will not work in a same-device Self-Issued OP protocol flow, when the RP and the Self-Issued OP are on the same device.

In the second scenario, the request includes the `authorization_endpoint` of a Self-Issued OP and will open a target application. In this scenario, there are two ways of how RP can obtain `authorization_endpoint` of the Self-Issued OP to construct a targeted request as defined in Section 6, either using the static set of Self-

Issued OP metadata, or by pre-obtaining `authorization_endpoint`. Note that this protocol flow would work both for the same-device Self-Issued OP protocol flow and the cross-device Self-Issued OP protocol flow.

The following is a non-normative example of a request with no specific `authorization_endpoint`, which must be scanned by the Self-Issued OP application manually opened by the End-user instead of an arbitrary camera application on a user-device. It is a request when the RP is pre-registered with the Self-Issued OP (line wraps within values are for display purposes only):

```
response_type=id_token
&client_id=https%3A%2F%2Fclient.example.org%2Fcb
&request_uri=https%3A%2F%2Fclient.example.org%2Frequest
&scope=openid
&nonce=n-0S6_WzA2Mj
```

6. Self-Issued OpenID Provider Discovery

RP can obtain `authorization_endpoint` of the Self-Issued OP to construct a request targeted to a particular application either by using the pre-agreed static configuration values, or by performing Dynamic Discovery as defined in Section 6.1.

6.1. Dynamic Discovery of Self-Issued OpenID Provider Metadata

As an alternative mechanism to the Section 15.1, the RP can pre-obtain Self-Issued OP Discovery Metadata prior to the transaction, either using [OpenID.Discovery], or out-of-band mechanisms.

How the RP obtains Self-Issued OP's issuer identifier is out of scope of this specification. The RPs MAY skip Section 2 of [OpenID.Discovery].

When [OpenID.Discovery] is used, the RP MUST obtain Self-Issued OP metadata from a JSON document that Self-Issued OP made available at the path formed by concatenating the string `/.well-known/openid-configuration` to the Self-Issued OP's Issuer Identifier.

Note that contrary to [OpenID.Discovery], `jwks_uri` parameter MUST NOT be present in Self-Issued OP Metadata. If it is, the RP MUST ignore it and use the sub Claim in the ID Token to obtain signing keys to validate the signatures from the Self-Issued OpenID Provider.

These OpenID Provider Metadata values are used by the Self-Issued OP:

- `authorization_endpoint`
  - REQUIRED. URL of the Self-Issued OP used by the RP to perform Authentication of the End-User. Can be custom URI scheme, or Universal Links/App links. See Section 6.2.
- `issuer`
  - REQUIRED. URL using the `https` scheme with no query or fragment component that the Self-Issued OP asserts as its Issuer Identifier. MUST be identical to the `iss` Claim value in ID Tokens issued from this Self-Issued OP.
- `response_types_supported`
  - REQUIRED. A JSON array of strings representing supported response types. MUST be `id_token`.
- `scopes_supported`

  - REQUIRED. A JSON array of strings representing supported scopes. MUST support the `openid` scope value.
- `subject_types_supported`
  - REQUIRED. A JSON array of strings representing supported subject types. Valid values include `pairwise` and `public`.
- `id_token_signing_alg_values_supported`
  - REQUIRED. A JSON array containing a list of the JWS signing algorithms (alg values) supported by the OP for the ID Token to encode the Claims in a JWT [RFC7519]. Valid values include `RS256`, `ES256`, `ES256K`, and `EdDSA`.
- `request_object_signing_alg_values_supported`
  - REQUIRED. A JSON array containing a list of the JWS signing algorithms (alg values) supported by the OP for Request Objects, which are described in Section 6.1 of [OpenID.Core]. Valid values include `none`, `RS256`, `ES256`, `ES256K`, and `EdDSA`.
- `subject_syntax_types_supported`
  - REQUIRED. A JSON array of strings representing URI scheme identifiers and optionally method names of supported Subject Syntax Types defined in {#sub-syntax-type}. When Subject Syntax Type is JWK Thumbprint, valid value is `urn:ietf:params:oauth:jwk-thumbprint` defined in [RFC9278]. When Subject Syntax Type is Decentralized Identifier, valid values MUST be a `did:` prefix followed by a supported DID method without a `:` suffix. For example, support for the DID method with a method-name "example" would be represented by `did:example`. Support for all DID methods listed in Section 13 of [DID_Specification_Registries] is indicated by sending did without any method-name.
- `id_token_types_supported`:
  - OPTIONAL. A JSON array of strings containing the list of ID Token types supported by the OP, the default value is `attester_signed_id_token`. The ID Token types defined in this specification are:
    - `subject_signed_id_token`: Self-Issued ID Token, i.e. the id token is signed with key material under the end-user's control.
    - `attester_signed_id_token`: the id token is issued by the party operating the OP, i.e. this is the classical id token as defined in [OpenID.Core].

Other Discovery parameters defined in Section 3 of [OpenID.Discovery] MAY be used.

The RP MUST use the `authorization_endpoint` defined in Self-Issued OP Discovery Metadata to construct the request.

The following is a non-normative example of a Self-Issued OP metadata obtained dynamically:

```
{
  "authorization_endpoint": "https://wallet.example.org",
  "issuer": "https://example.org",
  "response_types_supported": [
    "id_token"
  ],
  "scopes_supported": [
    "openid"
  ],
  "subject_types_supported": [
    "pairwise"
  ],
  "id_token_signing_alg_values_supported": [
    "ES256K",
    "EdDSA"
  ],
  "request_object_signing_alg_values_supported": [
    "ES256K",
    "EdDSA"
  ],
  "subject_syntax_types_supported": [
    "urn:ietf:params:oauth:jwk-thumbprint",
    "did:key"
  ],
  "id_token_types_supported": [
    "subject_signed_id_token"
  ]
}
```

6.2. Choice of `authorization_endpoint`

As the `authorization_endpoint` of a Self-Issued OP, the use of Universal Links or App Links is RECOMMENDED over the use of custom URI schemes. See Section 13.4 for details.

7. Relying Party Registration

The Self-Issued OP utilizing this specification has multiple options to obtain RP's metadata:

- Obtain it prior to a transaction, e.g using [OpenID.Registration] or out-of-band mechanisms. See Section 7.1 for the details.
- RP provides metadata to the Self-Issued OP just-in-time in the Self-Issued OP Request using one of the following mechanisms defined in this specification:
  - `client_id` equals `redirect_uri`
  - OpenID Federation 1.0 Automatic Registration
  - Decentralized Identifiers

Just-in-time metadata exchange allows SIOPv2 to be used in deployments models where the Self-Issued OP does not or cannot support pre-registration of Client metadata.

7.1. Pre-Registered Relying Party

See Section X.X of [OpenID4VP].

The following is a non-normative example of a same-device request when the RP is pre-registered with the Self-Issued OP. HTTP 302 redirect request by the RP triggers the User Agent to make an Authorization Request to the Self-Issued OP (with line wraps within values for display purposes only):

```
HTTP/1.1 302 Found
Location:  https://client.example.org/universal-link?
  response_type=id_token
  &client_id=s6BhdRkqt3
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
  &scope=openid%20profile
  &nonce=n-0S6_WzA2Mj
```

7.2. Non-Pre-Registered Relying Party

When the RP has not pre-registered, it may pass its metadata to the Self-Issued OP in the Authorization Request. This mechanism is different from registration (Dynamic Client Registration or an out-of-band pre-registration) since Self-Issued OP does not return `client_id` to the RP that the RP can re-use at the Self-Issued OP.

No registration response is returned. A successful Authorization Response implicitly indicates that the client metadata parameters were accepted.

7.2.1. `client_id` equals `redirect_uri`

As defined in Section X.X of [OpenID4VP].

7.2.2. OpenID Federation 1.0 Automatic Registration

As defined in Section X.X of [OpenID4VP].

The following is a non-normative example of a `client_id` resolvable using OpenID Federation 1.0 Automatic Registration:

```
"client_id": "https://client.example.org"
```

The following is a non-normative example of a signed cross-device request when the RP is not pre-registered with the Self-Issued OP and uses OpenID Federation 1.0 Automatic Registration. (with line wraps within values for display purposes only):

```
HTTP/1.1 302 Found
  Location: https://client.example.org/universal-link?
    response_type=id_token
    &client_id=https%3A%2F%2Fclient.example.org%2F
    &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
    &scope=openid%20profile
    &nonce=n-0S6_WzA2Mj
```

7.2.3. Decentralized Identifiers

As defined in Section X.X of [OpenID4VP].

The following is a non-normative example of a `client_id` resolvable using Decentralized Identifier Resolution:

```
"client_id": "did:example:EiDrihTRe0GMdc3K16kgJB3Xbl9Hb8oqVHjzm6ufHcYDGA"
```

The following is a non-normative example of a signed cross-device request when the RP is not pre-registered with the Self-Issued OP and uses Decentralized Identifier Resolution. (with line wraps within values for display purposes only):

```
siopv2://idtoken?
  scope=openid%20profile
  &response_type=id_token
  &client_id=did%3Aexample%3AEiDrihTRe0GMdc3K16kgJB3Xbl9Hb8oqVHjzm6ufHcYDGA
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
  &claims=...
  &registration=%7B%22subject_syntax_types_supported%22%3A
  %5B%22did%3Aexample%22%5D%2C%0A%20%20%20%20
  %22id_token_signing_alg_values_supported%22%3A%5B%22ES256%22%5D%7D
  &nonce=n-0S6_WzA2Mj
```

7.3. Relying Party Client Metadata parameter

As defined in Section X.X of [OpenID4VP].

The following is a non-normative example of an unsigned same-device request when the RP is not pre-registered with the Self-Issued OP. HTTP 302 redirect request by the RP triggers the User Agent to make an Authorization Request to the Self-Issued OP (with line wraps within values for display purposes only):

```
HTTP/1.1 302 Found
Location: https://client.example.org/universal-link?
  response_type=id_token
  &client_id=https%3A%2F%2Fclient.example.org%2Fcb
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
  &scope=openid%20profile
  &nonce=n-0S6_WzA2Mj
  &registration=%7B%22subject_syntax_types_supported%22%3A
  %5B%22urn%3Aietf%3Aparams%3Aoauth%3Ajwk-thumbprint%22%5D%2C%0A%20%20%20%20
  %22id_token_signing_alg_values_supported%22%3A%5B%22RS256%22%5D%7D
```

7.4. Relying Party Metadata Error Response

When Self-Issued OP receives a pre-registered `client_id` of a Section 7.1, but `client_metadata` or `client_metadata_uri` parameters of a Section 7.2 are also present, Self-Issued OP MUST return an error. When Self-Issued OP receives a `client_id` of a Section 7.2 that it has cached following one of the methods defined in Section 7.2, it does not not return an error.

Self-Issued OPs compliant with this specification MUST NOT proceed with the transaction when pre-registered client metadata has been found based on the `client_id`, but `client_metadata` parameter has also been present.

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

7.5. Relying Party Metadata Values

This extension defines the following RP parameter value, used by the RP to provide information about itself to the Self-Issued OP:

- `subject_syntax_types_supported`
  - REQUIRED. A JSON array of strings representing URI scheme identifiers and optionally method names of supported Subject Syntax Types defined in {#sub-syntax-type}. When Subject Syntax Type is JWK Thumbprint, valid value is `urn:ietf:params:oauth:jwk-thumbprint` defined in [RFC9278]. When Subject Syntax Type is Decentralized Identifier, valid values MUST be a `did:` prefix followed by a supported DID method without a `:` suffix. For example, support for the DID method with a method-name "example" would be represented by `did:example`. Support for all DID methods is indicated by sending `did` without any method-name.

Other client metadata parameters defined in [OpenID.Registration] MAY be used. Examples are explanatory parameters such as `policy_uri`, `tos_uri`, and `logo_uri`. If the RP uses more than one Redirection URI, the `redirect_uris` parameter would be used to register them. Finally, if the RP is requesting encrypted responses, it would typically use the `jwks_uri`, `id_token_encrypted_response_alg` and `id_token_encrypted_response_enc` parameters.

The following is a non-normative example of the supported RP parameter Values:

```
{
  "subject_syntax_types_supported": [
    "urn:ietf:params:oauth:jwk-thumbprint",
    "did:example",
    "did:key"
  ]
}
```

8. Subject Syntax Types

Subject Syntax Type refers to a type of an identifier used in a `sub` Claim in the ID Token issued by a Self-Issued OP. `sub` in Self-Issued OP protocol flow serves as an identifier of the Self-Issued OP's Holder and is used to obtain cryptographic material to verify the signature on the ID Token.

This specification defines the following two Subject Syntax Types. Additional Subject Syntax Types may be defined in future versions of this specification, or profiles of this specification.

- JWK Thumbprint subject syntax type. When this type is used, the `sub` Claim value MUST be the base64url encoded representation of the JWK thumbprint of the key in the `sub_jwk` Claim [RFC7638], and `sub_jwk` MUST be included in the Self-Issued Response.
- Decentralized Identifier subject syntax type. When this type is used, the `sub` value MUST be a DID as defined in [DID-Core], and `sub_jwk` MUST NOT be included in the Self-Issued Response. The subject syntax type MUST be cryptographically verified against the resolved DID Document as defined in {#siop-id_token-validation}.

The RP indicates Subject Syntax Types it supports in Client metadata parameter `subject_syntax_types_supported` defined in {#rp-metadata}.

9. Self-Issued OpenID Provider Authorization Request

Self-Issued OP Authorization Request is sent to the Authorization Endpoint, which performs Authentication of the End-User.

The Authorization Endpoint of the Self-Issued OP is used in the same manner as defined in Section 3 of [OpenID.Core], with the exception of the differences specified in this section.

Communication with the Authorization Endpoint MUST utilize TLS.

This specification defines the following new authorization request parameters in addition to [OpenID.Core]:

- `client_metadata`: OPTIONAL. This parameter is used by the RP to provide information about itself to a Self-Issued OP that would normally be provided to an OP during Dynamic RP Registration, as specified in {#rp-registration-parameter}. It MUST not be present if the RP uses OpenID Federation 1.0 Automatic Registration to pass its metadata.
- `client_metadata_uri`: OPTIONAL. This parameter is used by the RP to provide information about itself to a Self-Issued OP that would normally be provided to an OP during Dynamic RP Registration, as specified in {#rp-registration-parameter}. It MUST not be present if the RP uses OpenID Federation 1.0 Automatic Registration to pass its metadata.
- `id_token_type`: OPTIONAL. Space-separated string that specifies the types of ID Token the RP wants to obtain, with the values appearing in order of preference. The allowed individual values are `subject_signed_id_token` and `attester_signed_id_token` (see Section 6.1). The default value is `attester_signed_id_token`. The RP determines the type if ID Token returned based on the comparison of the `iss` and `sub` claims values (see(see Section 11.1). In order to preserve compatibility with existing OpenID Connect deployments, the OP MAY return an ID Token that does not fulfill the requirements as expressed in this parameter. So the RP SHOULD be prepared to reliably handle such an outcome.

This specification allows RPs to sent authorization request parameters by using "request by value" and "request by reference" as defined in [RFC9101] through the request parameters `request` or `request_uri`.

Note: When using the parameters `request` or `request_uri` the only further required parameter of the authorization request is the `client_id`.

When `request` or `request_uri` parameters are NOT present, and RP is NOT using OpenID Federation 1.0 Automatic Registration to pass entire RP metadata, `client_metadata` or `client_metadata_uri` parameters MUST be present in the request. `client_metadata` and `client_metadata_uri` are mutual exclusiv.

RPs MUST send a `nonce` parameter with every Self-Issued OP Authorization Request as a basis for replay detection complying with the security considerations given in [OpenID.Core], Section 15.5.2.

The ID Token to be used as an `id_token_hint` may have been encrypted to the RP in a previous transaction. Encrypted ID Tokens are Nested JWTs as defined in [JWT]. The RP MUST decrypt the ID Token value to retrieve the payload, which is a Self-Issued ID Token. The signed Self-Issued ID Token MAY be used as a hint.

Alternatively, the RP MAY re-encrypt the resulting Self-Issued ID Token to the subject for confidentiality. Re-encryption requires a mutually supported set of algorithms between the RP and SIOP, and at least one subject public key usable for encryption. Supported algorithms MAY be advertised with the `request_object_encryption_alg_values_supported` and `request_object_encryption_enc_values_supported` OP Discovery parameters.

When re-encrypting the ID Token value, the sub value from the signed ID Token MUST be included as a sub parameter within the JWE protected header. If the sub has multiple public keys associated, the JWE protected header MUST distinguish the appropriate key with the JWE kid protected header. The JWE protected header MUST specify alg and enc header parameters unless the use of specific alg and enc values have been pre-negotiated.

Other parameters MAY be sent. Note that all Claims are returned in the ID Token.

The entire URL is NOT RECOMMENDED to exceed 2048 ASCII characters.

Note that multiple size limitations exist: the majority of browsers and mobile OS in general have approximately 1MB of URL length restrictions, while QR codes, intermediary CDN or firewalls might have a lower URL length restriction.

The following is a non-normative example HTTP 302 redirect request by the RP which triggers the User Agent to make an Authorization Request to the Self-Issued OP in a same-device protocol flow (with line wraps within values for display purposes only):

```
  HTTP/1.1 302 Found
  Location: siopv2://?
    scope=openid
    &response_type=id_token
    &client_id=https%3A%2F%2Fclient.example.org%2Fcb
    &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
    &id_token_type=subject_signed_id_token
    &claims=...
    &registration=%7B%22subject_syntax_types_supported%22%3A
    %5B%22urn%3Aietf%3Aparams%3Aoauth%3Ajwk-thumbprint%22%5D%2C%0A%20%20%20%20
    %22id_token_signing_alg_values_supported%22%3A%5B%22ES256%22%5D%7D
    &nonce=n-0S6_WzA2Mj
```

The following is a non-normative example of an authorization request utilizing a request_uri (with line wraps within values for display purposes only):

```
  HTTP/1.1 302 Found
  Location: siopv2://?
    client_id=https%3A%2F%2Fclient.example.org%2Fcb
    &request_uri=https%3A%2F%2Fclient.example.org%2Frequests%2FGkurKxf5T0Y-
mnPFCHqWOMiZi4VS138cQO_V7PZHAdM
```

9.1. aud of a Request Object

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

- the aud claim MUST equal to the issuer Claim value, when Dynamic Self-Issued OP Discovery is performed.
- the aud claim MUST be "https://self-issued.me/v2", when Static Self-Issued OP Discovery Metadata is used.

9.2. Cross-Device Self-Issued OpenID Provider Request

The cross-device Authorization Request differs from the same-device variant (with response type id_token) as defined in Section 9 as follows:

- This specification introduces a new response mode `post` in accordance with [OAuth.Responses]. This response mode is used to request the Self-Issued OP to deliver the result of the authentication process to a certain endpoint using the HTTP `POST` method. The additional parameter `response_mode` is used to carry this value.

- This endpoint to which the Self-Issued OP shall deliver the authentication result is conveyed in the standard parameter `redirect_uri`.

Self-Issued OP is on a different device than the one on which the End-User's user interactions are occurring.

The following is a non-normative example of a Self-Issued Request URL in a cross-device protocol flow Section 4:

```
siopv2://?
  scope=openid%20profile
  &response_type=id_token
  &client_id=https%3A%2F%2Fclient.example.org%2Fpost_cb
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fpost_cb
  &response_mode=post
  &claims=...
  &registration=%7B%22subject_syntax_types_supported%22%3A
  %5B%22urn%3Aietf%3Aparams%3Aoauth%3Ajwk-thumbprint%22%5D%2C%0A%20%20%20%20
  %22id_token_signing_alg_values_supported%22%3A%5B%22ES256%22%5D%7D
  &nonce=n-0S6_WzA2Mj
```

Note that the Authorization Request might only include request parameters and not be targeted to a particular `authorization_endpoint`, in which case, the End-User must use a particular Self-Issued OP application to scan the QR code with such request.

Such an Authorization Request might result in a large QR code, especially when including a `claims` parameter and extensive registration data. A RP MAY consider using a `request_uri` in such a case.

10. Self-Issued OpenID Provider Authorization Response

A Self-Issued OpenID Provider Response is an OpenID Connect Authorization Response, whose parameters depend on the `response_type` used in the request. Depending on the `response_type` the result of the transaction is either obtained directly from the Authorization Response or from a token endpoint response.

A Self-Issued OpenID Provider Response is returned when Self-Issued OP supports all Relying Party parameter values received from the Relying Party in the `client_metadata` parameter. If one or more of the Relying Party parameter Values is not supported, Self-Issued OP MUST return an error according to Section 10.3.

In a same-device protocol flow with `response_type` `id_token`, the response parameters will be returned in the URL fragment component, unless a different Response Mode was specified.

In a same-device protocol flow with `response_type` `code`, the response parameters will be returned in HTTPS POST response body of the token response.

In a cross-device protocol flow, upon completion of the Authorization Request, the Self-Issued OP directly sends a HTTP POST request with the Authorization Response to an endpoint exposed by the RP.

The following is an informative example of a Self-Issued Response in a same-device protocol flow (`response_type=id_token`):

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

10.1. Self-Issued OpenID Provider Response (Authorization Code Flow)

The following is an informative example of a Self-Issued Response for the authorization code flow (`response_type=code`):

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

  {
   "access_token": "SlAV32hkKG",
   "token_type": "Bearer",
   "expires_in": 3600,
   "id_token": "..."
  }
```

Note: the SIOP MAY also provide end-user claims to the RP via the Userinfo endpoint.

10.2. Cross-Device Self-Issued OpenID Provider Response

The Self-Issued OP sends the Authorization Response to the endpoint passed in the `redirect_uri` Authorization Request parameter using a HTTP POST request using "application/x-www-form-urlencoded" encoding. The Authorization Response contains the parameters as defined in Section 10.

The Self-Issued OP MUST NOT follow redirects on this request.

The following is an informative example of a Self-Issued Response in a cross-device protocol flow: Section 4:

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

  id_token=...
```

10.3. Self-Issued OpenID Provider Error Response

The error response must be made in the same manner as defined in Section 3.1.2.6 of [OpenID.Core].

In addition to the error codes defined in Section 4.1.2.1 of OAuth 2.0 and Section 3.1.2.6 of [OpenID.Core], this specification also defines the following error codes:

- **`user_cancelled`**: End-User cancelled the Authorization Request from the RP.
- **`registration_value_not_supported`**: the Self-Issued OP does not support some Relying Party parameter values received in the request.
- **`subject_syntax_types_not_supported`**: the Self-Issued OP does not support any of the Subject Syntax Types supported by the RP, which were communicated in the request in the `subject_syntax_types_supported` parameter.

- **invalid_registration_uri**: the `client_metadata_uri` in the Self-Issued OpenID Provider request returns an error or contains invalid data.
- **invalid_registration_object**: the `client_metadata` parameter contains an invalid RP parameter Object.

Other error codes MAY be used.

Note that HTTP error codes do not work in the cross-device Self-Issued OP protocol flows.

The following is a non-normative example of an error response in the same-device Self-Issued OP protocol flow:

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

11. Self-Issued ID Token

The response contains an ID Token and, if applicable, further response parameters as defined in extensions. As an example, the response MAY also include a VP token as defined in [OpenID4VP].

This extension defines the following claims to be included in the ID Token for use in Self-Issued OpenID Provider Responses:

- `iss`
  - REQUIRED. in case of a Self-Issued ID Token, this claim MUST be set to the value of the `sub` claim in the same ID Token.
- `sub`
  - REQUIRED. Subject identifier value. When Subject Syntax Type is JWK Thumbprint, the value is the base64url encoded representation of the thumbprint of the key in the `sub_jwk` Claim. When Subject Syntax Type is Decentralized Identifier, the value is a Decentralized Identifier. The thumbprint value of JWK Thumbprint Subject Syntax Type is computed as the SHA-256 hash of the octets of the UTF-8 representation of a JWK constructed containing only the REQUIRED members to represent the key, with the member names sorted into lexicographic order, and with no white space or line breaks. For instance, when the `kty` value is `RSA`, the member names `e`, `kty`, and `n` are the ones present in the constructed JWK used in the thumbprint computation and appear in that order; when the `kty` value is `EC`, the member names `crv`, `kty`, `x`, and `y` are present in that order. Note that this thumbprint calculation is the same as that defined in the JWK Thumbprint [RFC7638] specification.
- `sub_jwk`
  - OPTIONAL. A JSON object that is a public key used to check the signature of an ID Token when Subject Syntax Type is JWK Thumbprint. The key is a bare key in JWK [RFC7517] format (not an X.509 certificate value). MUST NOT be present when Subject Syntax Type other than JWK Thumbprint is used.

Note that the use of the `sub_jwk` Claim is NOT RECOMMENDED when the OP is not Self-Issued.

Whether the Self-Issued OP is a mobile application or a Web application, the ID Token is the same as defined in [OpenID.Core] with the following refinements:

1. The sub (subject) Claim value is either the base64url encoded representation of the thumbprint of the key in the sub_jwk Claim or a Decentralized Identifier.

2. When the sub Claim value is the base64url encoded representation of the thumbprint, a sub_jwk Claim is present, with its value being the public key used to check the signature of the ID Token.

The following is a non-normative example of a base64url decoded Self-Issued ID Token body when the JWK Thumbprint Subject Syntax type is used (with line wraps within values for display purposes only):

```
{
  "iss": "NzbLsXh8uDCcd-6MNwXF4W_7noWXFZAfHkxZsRGC9Xs",
  "sub": "NzbLsXh8uDCcd-6MNwXF4W_7noWXFZAfHkxZsRGC9Xs",
  "aud": "https://client.example.org/cb",
  "nonce": "n-0S6_WzA2Mj",
  "exp": 1311281970,
  "iat": 1311280970,
  "sub_jwk": {
    "kty": "RSA",
    "n": "0vx7agoebGcQSuuPiLJXZptN9nndrQmbXEps2aiAFbWhM78LhWx4cbbfAAt
    VT86zwu1RK7aPFFxuhDR1L6tSoc_BJECPebWKRXjBZCiFV4n3oknjhMstn64tZ_2W
    -5JsGY4Hc5n9yBXArwl93lqt7_RN5w6Cf0h4QyQ5v-65YGjQR0_FDW2QvzqY368QQ
    MicAtaSqzs8KJZgnYb9c7d0zgdAZHzu6qMQvRL5hajrn1n91CbOpbISD08qNLyrdk
    t-bFTWhAI4vMQFh6WeZu0fM4lFd2NcRwr3XPksINHaQ-G_xBniIqbw0Ls1jF44-cs
    FCur-kEgU8awapJzKnqDKgw",
    "e": "AQAB"
  }
}
```

11.1. Self-Issued ID Token Validation

See [OpenID4VP] on how to support multiple credential formats such as JWT and Linked Data Proofs.

To validate the ID Token received, the RP MUST do the following:

1. The Relying Party (RP) MUST determine whether the ID Token has been issued by the Self-Issued OP. The ID Token is self-issued if the issclaims and the subclaim have the same value. If both values differ, the ID Token MUST be processed as defined in [OpenID.Core], section 3.2.2.11..

2. The RP MUST validate that the aud (audience) Claim contains the value of the client_id that the RP sent in the Authorization Request as an audience. When the request has been signed, the value might be an HTTPS URL, or a Decentralized Identifier.

3. The RP MUST identify which Subject Syntax Type is used based on the URI of the sub Claim. Valid values defined in this specification are urn:ietf:params:oauth:jwk-thumbprint for JWK Thumbprint Subject Syntax Type and did: for Decentralized Identifier Subject Syntax Type.

4. The RP MUST validate the signature of the ID Token. When Subject Syntax Type is JWK Thumbprint, validation is done according to JWS [RFC7515] using the algorithm specified in the alg header parameter of the JOSE Header, using the key in the sub_jwk Claim. The key MUST be a bare key in JWK format (not an X.509 certificate value). The RP MUST validate that the algorithm is one of the allowed algorithms (as in id_token_signing_alg_values_supported). When Subject Syntax Type is Decentralized Identifier, validation is performed against the key obtained from a DID Document. DID Document MUST be obtained by resolving a Decentralized Identifier included in the sub Claim using DID Resolution as defined

by a DID Method specification of the DID Method used. Since `verificationMethod` property in the DID Document may contain multiple public key sets, public key identified by a key identifier `kid` in a Header of a signed ID Token MUST be used to validate that ID Token.

5. The RP MUST validate the `sub` value. When Subject Syntax Type is JWK Thumbprint, the RP MUST validate that the `sub` Claim value equals the base64url encoded representation of the thumbprint of the key in the `sub_jwk` Claim, as specified in Section 10. When Subject Syntax Type is Decentralized Identifier, the RP MUST validate that the `sub` Claim value equals the `id` property in the DID Document.

6. The current time MUST be before the time represented by the `exp` Claim (possibly allowing for some small leeway to account for clock skew). The `iat` Claim can be used to reject tokens that were issued too far away from the current time, limiting the amount of time that nonces need to be stored to prevent attacks. The acceptable range is RP-specific.

7. The RP MUST validate that a `nonce` Claim is present and is the same value as the one that was sent in the Authorization Request. The Client MUST check the `nonce` value for replay attacks. The precise method for detecting replay attacks is RP specific.

Any claim in the Self-Issued ID Token is considered to be self-attested. Verifying attestation by a third party requires additional artifacts and processing steps - see [OpenID4VP].

11.2. Cross-Device Self-Issued ID Token Validation

The RP MUST perform all the check as defined in Section 11.1.

Additionally, the RP MUST check whether the `nonce` Claim value provided in the ID Token is known to the RP and was not used before in an Authorization Response.

The following is a non-normative example of a base64url decoded Self-Issued ID Token body when the dynamic Self-Issued OP Discovery and Decentralized Identifier Subject Syntax type are used (with line wraps within values for display purposes only):

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

12. Verifiable Presentation Support

Self-Issued OP and the RP that wish to support request and presentation of Verifiable Presentations MUST be compliant with OpenID for Verifiable Presentations [OpenID4VP] and W3C Verifiable Credentials Specification [VC-DATA].

Verifiable Presentation is a tamper-evident presentation encoded in such a way that authorship of the data can be trusted after a process of cryptographic verification. Certain types of verifiable presentations might contain selectively disclosed data that is synthesized from, but does not contain, the original verifiable credentials (for example, zero-knowledge proofs). [VC-DATA]

To prevent replay attacks, any Verifiable Presentations presented in a Self-Issued OP protocol flow MUST be bound to the `nonce` provided by the RP and the `client_id` of the RP, as described in [OpenID4VP].

13. Security Considerations

13.1. Handling End-User Data

Using the mechanisms described in this specification and [OpenID4VP], data about the End-User can be transported to the Relying Party in different ways. It is important that implementers take into consideration the specific security properties associated with the mechanisms.

13.1.1. End-User Claims in ID Tokens

Regarding the ID Token as a transport mechanism, the RP can trust that the Self-Issued OP has access to the private key required to sign the ID Token. The value of the `sub` Claim is bound to this key. It is in the Self-Issued OP's interest to not make the signing available to other parties, in particular attackers. The RP can therefore use the `sub` Claim as a primary identifier for the End-User, assuming that the ID Token was checked properly.

Other Claims within the ID Token MUST be considered self-asserted: The Self-Issued OP can use arbitrary data that can usually not be checked by the RP. RPs therefore MUST NOT use other Claims than `sub` to (re-)identify users, for example, for login.

13.1.2. Additional Data in Verifiable Presentations

The validity of data presented in W3C Veriable Presentations is attested by the issuer of the underlying W3C Verifiable Credential. The RP MUST ensure that it trusts the specific issuer, and verify that the Verifiable Presentation is correctly bound to the Self-Issued OP transaction (`nonce` and `client_id` binding as described above) before using the data. The cryptographic keys within the Verifiable Presentation and for signing the ID Token are not necessarily related and the RP SHOULD NOT make assumptions in this regard.

RPs MUST consider that Verifiable Presentations can be revoked and that user data within the W3C Verifiable Credential may change over time. Such changes can only be noticed by the RP if the W3C Verifiable Presentation is checked at each login.

13.2. RP and Self-Issued OP Metadata Integrity

The integrity and authenticity of Self-Issued OP and RP metadata is paramount for the security of the protocol flow. For example, a modified `authorization_endpoint` could be used by an attacker to launch phishing or mix-up-style attacks (see [I-D.oauth-security-topics]). A modified `redirect_uri` could be used by an attacker to gather information that can then be used in replay attacks. The provisions defined in this specification ensure the authenticity and integrity of the metadata if all checks (signature validation, etc.) are performed as described.

13.3. Usage of Cross-Device Self-Issued OP for Authentication

A known attack in cross-device Self-Issued OP is an Authorization Request replay attack, where a victim is tricked to send a response to an Authorization Request that an RP has generated for an attacker. In other words, the attacker would trick a victim to respond to a request that the attacker has generated for him/herself at a good RP, for example, by showing the QR code encoding the Authorization Request that would normally be presented on the RP's website on the attacker's website. In this case, the victim might not be able to detect that the request was generated by the RP for the attacker. (Note that the same attack applies when methods other than a QR code are used to encode the Authorization Request. For brevity, only the QR code method will be discussed in the following, but the same considerations apply to other transport methods as well.)

This attack is based on the fact that the Authorization Request is not tied to a specific channel, i.e., it can be used both in its original context (on the RP's website) as well as in a replay scenario (on the attacker's website, in an email, etc.). Such a binding cannot be established across devices with currently deployed technology. Therefore, in the cross-device protocol flow, the contents transported in the authentication (including any Verfiable Presentations) are not wrong, but do not necessarily come from the End-User the RP website thinks it is interacting with.

Implementers MUST take this fact into consideration when using cross-device Self-Issued OP. There are a number of measures that can be taken to reduce the risk of such an attack, but none of these can completely prevent the attack. For example:

- The Self-Issued OP can show the origin of the request (e.g., the domain where the response will be sent) and/or ask the End-User to confirm this origin. In an attack, this origin would be different from the attacker's origin where the QR code is displayed to the End-User. It cannot be expected that all End-Users will identify an attack in this way, but at least for some users it might help to detect the attack. Note that depending on the device on which the attacker is showing the QR code, there might be no indication of the origin, e.g., on a terminal devices that does not show a browser with a URL bar.
- The Authorization Request can be made short-lived. An attacker would have to request a new Authorization Request (e.g., QR code) frequently and update it on its website as well. This requires more effort from the attacker.
- Self-Issued OP can warn the End-User when logging in at a new RP for the first time.

Implementors should be cautious when using cross-device Self-Issued OP model for authentication and should implement mitigations according to the desired security level.

This attack does not apply for the same-device Self-Issued OP protocol flows as the RP checks that the Authorization Response comes from the same browser where the Authorization Request was sent to. Same-device Self-Issued OP protocol flows therefore can be used for authentication, given all other security measures are put in place.

13.4. Invocation using Private-Use URI Schemes (Custom URL Schemes)

Usage of private-use URI schemes with a certain path such as `siopv2://`, also referred to as custom URL schemes, as a way to invoke a Self-Issued OP may lead to phishing attacks and undefined behavior, as described in [RFC8252].

Private-use URI schemes are a mechanism offered by mobile operating system providers. If an application developer registers a custom URL scheme with the application, that application will be invoked when a request containing custom scheme is received by the device. If no available application supports the custom URI scheme, the platform or browser will typically generate a modal error and present it to the End-User.

Implementors are highly encouraged to explore other options before using private-use URI schemes due to the known security issues of such schemes, such as lack of controls to prevent unknown applications from attempting to service Authorization Requests. Any malicious app can register the custom scheme already used by another app, imitate the user interface and impersonate a good app.

The browser or operating system typically has a process by which native applications and websites can register support that one or more apps be called when a HTTPS URI is triggered in lieu of a system browser. This feature goes by several names including "App Links" and "Universal Links", and MAY be used to invoke an installed/registered Self-Issued OP. If no appropriate application has been rendered, the request for a Self-Issued OP will go to a browser, which MAY display additional information such as appropriate software options.

Further details are discussed in [app-2-app-sec].

13.5. Self-Issued OP Authorization Response Confidentiality

The Authorization Response MUST NOT leak to a party outside of the Self-Issued OP, the RP and the system browser participating in the protocol flow.

Implementers of Self-Issed OPs are encouraged to take operating system dependent measures to ensure that the Authorization Response is only passed to a legitimate system browser in the same-device protocol flow. If implementers wish to support native applications acting as an RP, they MAY alternatively invoke the RP application if the Self-Issed OP can verify the RP application to be legitimate.

Further details of application to application and application to web communication are discussed in [app-2-app-sec].

14. Privacy Considerations

14.1. Selective Disclosure and Unlinkable Presentations

Usage of decentralized identifiers does not automatically prevent possible RP correlation. If a status check of the presentation is done, Identity Provider / Self-Issued OP correlation can occur.

Consider supporting selective disclosure and unlinkable presentations using zero-knowledge proofs or single-use credentials instead of traditional correlatable signatures.

15. Implementation Considerations

15.1. Static Configuration Values of the Self-Issued OPs

This document lists profiles that define static configuration values of Self-Issued OPs and defines two set of static configuration values for Self-Issued OPs as native apps that can be used by the RP when it is unable to perform dynamic discovery and is not using any of the profiles.

15.1.1. Profiles that Define Static Configuration Values

Below is a list of profiles that define static configuration values of Self-Issued OPs:

- JWT VC Presentation Profile

15.1.2. A Set of Static Configuration Values bound to `siopv2://`

Below is a set of static configuration values that can be used with `id_token` as a supported `response_type`, bound to a custom URL scheme `siopv2://` as an `authorization_endpoint`:

```
{
  "authorization_endpoint": "siopv2:",
  "response_types_supported": [
    "id_token"
  ],
  "scopes_supported": [
    "openid"
  ],
  "subject_types_supported": [
    "pairwise"
  ],
  "id_token_signing_alg_values_supported": [
    "ES256"
  ],
  "request_object_signing_alg_values_supported": [
    "ES256"
  ],
  "subject_syntax_types_supported": [
    "urn:ietf:params:oauth:jwk-thumbprint"
  ],
  "id_token_types_supported": [
    "subject_signed_id_token"
  ]
}
```

15.1.3. A Set of Static Configuration Values bound to `openid://`

Another set of static configuration values is used with both `vp_token` and `id_token` as supported `response_type`, bound to a custom URL scheme `openid://` as an `authorization_endpoint`:

```
{
  "authorization_endpoint": "openid:",
  "response_types_supported": [
    "vp_token",
    "id_token"
  ],
  "vp_formats_supported": {
    "jwt_vp": {
      "alg": ["ES256"]
    },
    "jwt_vc": {
      "alg": ["ES256"]
    }
  },
  "scopes_supported": [
    "openid"
  ],
  "subject_types_supported": [
    "pairwise"
  ],
  "id_token_signing_alg_values_supported": [
    "ES256"
  ],
  "request_object_signing_alg_values_supported": [
    "ES256"
  ],
  "subject_syntax_types_supported": [
    "urn:ietf:params:oauth:jwk-thumbprint"
  ],
  "id_token_types_supported": [
    "subject_signed_id_token"
  ]
}
```

15.2. Supporting Multiple Self-Issued OPs

When the RP wants to provide End-User choice to select from multiple possible Self-Issued OPs, it MAY present a static list of the supported options. This is similar to the process of supporting multiple different social networks represented as traditional OPs.

Note that if Self-Issued OP implementations belong to a trust framework, the trust framework may dictate a common `authorization_endpoint` for a set of implementations. If `authorization_endpoint` is pre-registered with the underlying browser or operating system, invocation of this endpoint that leads to prompting the End-User to select a Self-Issued OP is handled by the underlying browser or operating system.

15.3. Receiving Cross-Device Responses

In case of the cross-device flow, the Self-Issued OPwill send the result as a HTTP POST message to the RP. This requires connectivity between Self-Issued OPand RP. There are different ways this can be achieved. The RP may, for example, expose a suitable endpoint from their backend. Alternatively, it may employ a separate service able to receive and store such messages, where the RP then queries the Self-Issued OPresponses.

16. Relationships to Other Documents

The scope of this draft was an extension to Chapter 7 Self-Issued OpenID Provider in [OpenID.Core]. However, some sections of it could become applicable more generally to the entire OpenID Connect specification.

17. Normative References

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

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

[RFC9101] Sakimura, N., Bradley, J., Jones, M., and RFC Publisher, "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>.

[RFC4648] Josefsson, S. and RFC Publisher, "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

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

[RFC9278] Jones, M., Yasuda, K., and RFC Publisher, "JWK Thumbprint URI", RFC 9278, DOI 10.17487/RFC9278, August 2022, <https://www.rfc-editor.org/info/rfc9278>.

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

[OpenID.Discovery] Sakimura, N., Bradley, J., Jones, M., and E. Jay, "OpenID Connect Core 1.0 incorporating errata set 1", 8 November 2014, <https://openid.net/specs/openid-connect-discovery-1_0.html>.

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

[OpenID4VP] Terbu, O., Lodderstedt, T., Yasuda, K., Lemmon, A., and T. Looker, "OpenID for Verifiable Presentations", 20 May 2021, <https://openid.net/specs/openid-4-verifiable-presentations-1_0.html>.

[OpenID.CIBA] Fernandez, G., Walter, F., Nennker, A., Tonge, D., and B. Campbell, "OpenID Connect Client-Initiated Backchannel Authentication Flow - Core 1.0", 1 September 2021, <https://openid.net/specs/openid-client-initiated-backchannel-authentication-core-1_0.html>.

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

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

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

[VC-DATA] Sporny, M., Noble, G., Longley, D., Burnett, D. C., and B. Zundel, "Decentralized Identifiers (DIDs) v1.0", 19 November 2019, <https://www.w3.org/TR/vc-data-model/>.

18. Informative References

[I-D.oauth-security-topics] Lodderstedt, T., Bradley, J., Labunets, A., and D. Fett, "OAuth 2.0 Security Best Current Practice", 16 December 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-security-topics-19>.

[app-2-app-sec] Hauck, F., Fett, D., and J. Heenan, "Improving OAuth App-to-App Security", 27 November 2020, <https://danielfett.de/2020/11/27/improving-app2app/>.

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

[DID_Specification_Registries] Steele, O., Sporny, M., and M. Prorock, "DID Specification Registries", August 2022, <https://www.w3.org/TR/did-spec-registries/>.

Appendix A. IANA Considerations

TBD

Appendix B. Acknowledgements

We would like to thank Christina Bauer, John Bradley, Kim Cameron, Giuseppe De Marco, Daniel Fett, Alen Horvat, Edmund Jay, Tom Jones, Niels Klomp, Torsten Lodderstedt, Tobias Looker, Jeremie Miller, Brandon Murdoch, Nat Sakimura, Oliver Terbu, David Waite, and Dmitri Zagidulin for their contributions to this specification.

Appendix C. Notices

against other contributors and against implementers. The OpenID Foundation invites any interested party to bring to its attention any copyrights, patents, patent applications, or other proprietary rights that may cover technology that may be required to practice this specification.

Appendix D. Use Cases

D.1. Resilience against Sudden or Planned Hosted OP Unavailability

A hosted third-party provided OP's infrastructure may become unavailable or even destroyed due to natural disasters such as hurricanes, tsunamis and fires, or may be removed from service as a planned business decision. End-Users using Self-Issued OPs local to their environment, have lower chances of being simultaneously affected by such events.

D.2. Authentication at the Edge

As internet-connected smartphones have risen in availability, traditionally in-person interactions and services have begun to be optimized with digital alternatives. These services often have requirements for digital authentication and for other identity credentials. Self-Issued OPs can provide this authentication directly, without needing to delegate to remote, hosted OPs. This potentially allows for increased efficiency as well as allowing for authentication in environments which may have reduced connectivity.

D.3. Authentication and Presentations of User Claims without the involvement of the Issuer

The RP can directly receive the issuer-signed claims about the End-User from the Self-Issued OP, without talking to the Issuer. This prevents the Issuer from knowing where the End-User is presenting these issuer-signed claims. In this use-case, obtaining and potentially storing the issuer-signed credentials is the Self-Issued OP's responsibility using specifications such as [OpenID4VP].

D.4. Sharing Claims (e.g. VC) from Several Issuers in One Transaction

When End-Users apply to open a banking account online, in most countries, they are required to submit scanned versions of the required documents. These documents are usually issued by different authorities, and are hard to verify in a digital form. A Self-issued OP directly representing the End-User may have access to a greater set of such information for example in the format of Verifiable Credentials, while a traditional OP may not have a business relationship which enables access to such a breadth of information. Self-Issued OPs could aggregate claims from multiple sources, potentially in multiple formats, then release them within a single transaction to a Relying Party. The Relying Party can then verify the authenticity of the information to make the necessary business decisions.

D.5. Aggregation of Multiple Personas under One Self-Issued OP

End-Users often use several hosted OpenID Providers for different Relying Parties. Some of the reasons to do this is to separate a work-related persona from a personal persona or to prevent the RPs from correlating their activities by using the same OP.

The usage of multiple OPs can create friction later, as the End-User may return later having forgotten which OP they used for the Relying Party. A single Self-Issued OP can be chosen by the End-User based on its capability to meet specific needs and privacy concerns.

D.6. Identifier Portability

With a hosted third-party provider, a user identifier used at the RP is assigned by the third-party, while with a Self-Issued OP, a user identifier can be created by an application locally controlled by the End-User. This would allow the End-User to maintain the relationship with the RP while minimizing the third-party influence, as long as the End-User can keep control of the Self-Issued identifier.

D.7. Cloud Wallet

End-Users may decide to store their credentials in a cloud wallet, in order to be able to access her existing credentials across devices without hazzle (e.g. no need to re-obtain credentials after changing phone). A cloud wallet is an applicatication that is not hosted entirely locally on the End-User's device, but has cloud-based components and is capable of hosting backend endpoints. Such a wallet can protect the user's credental on a high security level. It may, for example, utilize hardware security modules to protect the user's keys from cloning and replay. Since there is a backend involved and endpoints can be exposed, a cloud wallet can utilize the OpenID Connect authorization code flow, which allows verifier and wallet to mutually authenticate and exchange date via a direct HTTPS protected connection.

A cloud wallet may utilize a native user experience, it may also (in addition or exclusively) offer a web based experience, which can be used on any device without the need for an app installation. This also means such a wallet can always use the more secure on-device flow instead of the cross-device flow. End-user authentication can be implemented using roaming authenticators or a private protocol with an authentication app.

Appendix E. Document History

[[To be removed from the final specification]]

-12

- Made SIOPv2 compatible with JAR
- Clean up of the text

-11

- Replaced JWK-Thumbprint-URI reference with RFC 9278.
- Renamed registration and registration*uri to client*metadata and client*metadata*uri, respectively
- Relaxed URI length restrictions
- Added definition for base64url encoding
- Added definition of term "wallet"

-10

- Updated referenced spec names from "OpenID Connect for Verifiable ..." to "OpenID for Verifiable ...".

-09

- updated definition of SIOP to be "an OP within the End-User's control", not local control

- added reference to Distributed and Aggregated Claims as an option to send third party attested claims using SIOP
- added metadata and request parameter to explicitely publish and request SIOP support
- extended specification to allow for use the code flow and all other OpenID Connect flows
- added examples of possible SIOP architectures

-08

- added security consideration for confidentiality response (same-device)
- added implementation consideration for cross device flow
- `iss` in the ID Token must equal `sub`

-07

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

-06

- Added statically registered clients
- Cleanups in preparation for the first proposed Implementer's Draft
- Updated Security Considerations
- Added an Acknowledgements section
- Clarified which Hash Function is to be used to compute/verify JWK Thumbprints
- Added more examples

-05

- Merged `did_methods_supported` metadata into `subject_syntax_type_supported`
- Added RP Metadata resolution methods
- Editorial - language in Relying Party Registration Metadata Error Response
- Introduced mandatory `i_am_siop` claim

-04

- Added cross-device protocol flow
- Clarified handling for did-based sub and sub_jwk
- Revising of introductory text and scope of Self-Issued OP v2
- Corrected typos and reworked registration example data

-03

- sub_jwk made optional for Subject Syntax Type DID and mandatory for subtype jwk thumbprint

- Added text that nonce is mandatory
- Replaced vp claim with reference to OpenID4VP draft
- Adopted Self-Issued OP chooser as Self-Issued OP Discovery
- Deprecated openid:// for Self-Issued OP Discovery as not recommended
- Clarified Discovery and Registration metadata
- Formatted Normative Reference Section to mmarkdown

-02

- Converted into mmarkdown

-01

- Version proposed for working group adoption

Authors' Addresses

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

Michael B. Jones
Microsoft
Email: mbj@microsoft.com

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