Workgroup: OpenID Connect Working Group
Published: 22 May 2023
Authors: R. Hedberg, Ed. *independent*
M.B. Jones *Microsoft*
A.Å. Solberg *Sikt*
J. Bradley *Yubico*
G. De Marco *independent*
V. Dzhuvinov *Connect2id*

OpenID Connect Federation 1.0 - draft 29

Abstract

A federation can be expressed as an agreement between parties that trust each other. In bilateral federations, direct trust can be established between two organizations belonging to the same federation. In a multilateral federation, bilateral agreements might not be practical, in which case, trust can be mediated by a third party. That is the model used in this specification.

An Entity in the federation must be able to trust that other Entities it interacts with belong to the same federation. It must also be able to trust that the information the other Entities publish about themselves has not been tampered with during transport and that it adheres to the federation's policies.

This specification describes the basic components to build a multilateral federation and provides a guide on how to apply them when the underlying protocol used is OpenID Connect.

Table of Contents

1. Introduction
   1.1. Requirements Language
   1.2. Terminology
2. Overall Architecture
   2.1. Cryptographic Trust Mechanism
3. Components
   3.1. Entity Statement
   3.2. Trust Chain
      3.2.1. Trust Chain Header Parameter
4. Metadata Type Identifiers
   4.1. OpenID Connect and OAuth 2.0 Metadata Extensions
      4.1.1. Usage of jwks, jwks_uri, and signed_jwks_uri
   4.2. OpenID Connect Relying Party
   4.3. OpenID Provider
   4.4. OAuth Authorization Server

4.5. OAuth Client
4.6. OAuth Protected Resource
4.7. Federation Entity
5. Federation Policy
5.1. Metadata Policy
5.1.1. metadata_policy
5.1.2. Operators
5.1.3. Restrictions on Policy Entries
5.1.4. Combining Policies
5.1.4.1. Merging Operators
5.1.5. Applying Policies
5.1.6. Policy Combination Example
5.1.7. Enforcing Policy
5.1.8. Extending the Policy Language
5.1.9. Policy Example
5.2. Applying Constraints
5.2.1. Max Path Length
5.2.2. Naming Constraints
5.2.3. Leaf Entity Type Constraints
5.3. Trust Marks
5.3.1. Trust Mark Claims
5.3.2. Validating a Trust Mark
5.3.3. Trust Mark Examples
6. Obtaining Federation Entity Configuration Information
6.1. Federation Entity Configuration Request
6.2. Federation Entity Configuration Response
7. Federation Endpoints
7.1. Fetching Entity Statements
7.1.1. Fetch Entity Statements Request
7.1.2. Fetch Entity Statements Response
7.2. Resolve Entity Statement
7.2.1. Resolve Request
7.2.2. Resolve Response

7.2.3. Considerations
7.3. Subordinate Listings
7.3.1. Subordinate Listing Request
7.3.2. Subordinate Listing Response
7.4. Trust Mark Status
7.4.1. Status Request
7.4.2. Status Response
7.5. Trust Marked Entities Listing
7.5.1. Trust Marked Entities Listing Request
7.5.2. Trust Marked Entities Listing Response
7.6. Federation Historical Keys endpoint
7.6.1. Federation Historical Keys Request
7.6.2. Federation Historical Keys Response
7.6.3. Rationale for the Federation Historical Keys endpoint
7.7. Generic Error Response
8. Resolving Trust Chain and Metadata
8.1. Fetching Entity Statements to Establish a Trust Chain
8.2. Validating a Trust Chain
8.3. Choosing One of the Valid Trust Chains
8.4. Calculating the Expiration Time of a Trust Chain
9. Updating Metadata, Key Rollover, and Revocation
9.1. Protocol Key Rollover
9.2. Key Rollover for a Trust Anchor
9.3. Redundant Retrieval of Trust Anchor Keys
9.4. Revocation
10. OpenID Connect Communication
10.1. Automatic Registration
10.1.1. Authentication Request
10.1.1.1. Using a Request Object
10.1.1.1.1. Authorization Request with a Trust Chain
10.1.1.1.2. Processing the Authentication Request
10.1.1.2. Using Pushed Authorization

10.1.1.2.1. Processing the Authentication Request
10.1.2. Authentication Error Response
10.1.3. Possible Other Uses of Automatic Registration
10.2. Explicit Registration
10.2.1. Client Registration
10.2.1.1. Client Registration Request
10.2.1.2. Client Registration Response
10.2.1.2.1. OP Constructing the Response
10.2.1.2.2. RP Parsing the Response
10.2.2. After Explicit Client Registration
10.2.2.1. What the RP MUST Do
10.2.2.2. What the OP MUST Do
10.2.3. Expiration Times
10.3. Differences between Automatic Registration and Explicit Registration
10.4. Rationale for the Trust Chain in the Request
11. Claims Languages and Scripts
12. IANA Considerations
12.1. OAuth Authorization Server Metadata Registry
12.1.1. Registry Contents
12.2. OAuth Dynamic Client Registration Metadata Registration
12.2.1. Registry Contents
12.3. OAuth Extensions Error Registration
12.3.1. Registry Contents
12.4. Media Type Registration
12.4.1. Registry Contents
12.5. OAuth Parameter Registry
12.5.1. Registry Contents
12.6. JWS Header Registry
12.6.1. Registry Contents
13. Security Considerations
13.1. Denial-of-Service Attack Prevention
13.2. Unsigned Error Messages

14. References

14.1. Normative References

14.2. Informative References

Appendix A. Provider Information Discovery and Client Registration in a Federation

A.1. Setting Up a Federation

A.2. The LIGO Wiki Discovers the OP's Metadata

A.2.1. Configuration Information for op.umu.se

A.2.2. Configuration Information for 'https://umu.se'

A.2.3. Entity Statement Published by 'https://umu.se' about 'https://op.umu.se'

A.2.4. Configuration Information for 'https://swamid.se'

A.2.5. Entity Statement Published by 'https://swamid.se' about 'https://umu.se'

A.2.6. Configuration Information for 'https://edugain.geant.org'

A.2.7. Entity Statement Published by 'https://edugain.geant.org' about 'https://swamid.se'

A.2.8. Verified Metadata for op.umu.se

A.3. The Two Ways of Doing Client Registration

A.3.1. RP Sends Authentication Request (Automatic Registration)

A.3.1.1. OP Fetches Entity Statements

A.3.1.2. OP Evaluates the RP Metadata

A.3.2. Client Starts with Registration (Explicit Client Registration)

Appendix B. Notices

Appendix C. Acknowledgements

Appendix D. Document History

Authors' Addresses

1. Introduction

This specification describes how two Entities that would like to interact can resolve trust between them, by means of a third trusted party called Trust Anchor. A Trust Anchor is an Entity whose main purpose is to issue statements about Entities. An identity federation can be realized using this specification using one or more levels of authorities. Examples of authorities are federation operators, organizations, departments within organizations, and individual sites. This specification does not mandate a specific way or restrict how a federation may be built. Instead, the specification provides the basic technical trust infrastructure building blocks needed to create a dynamic and distributed trust network, such as a federation.

Note that this specification only concerns itself with how Entities in a federation get to know about each other. Furthermore, note that an organization MAY be represented by more than one Entity in a federation. It is also true that an Entity can be part of more than one federation.

1.1. Requirements Language

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].

1.2. Terminology

This specification uses the terms "Claim Name", "Claim Value", "JSON Web Token (JWT)", defined by JSON Web Token (JWT) [RFC7519], and the terms "OpenID Provider (OP)" and "Relying Party (RP)" defined by OpenID Connect Core 1.0 [OpenID.Core].

This specification also defines the following terms:

Entity
: Something that has a separate and distinct existence and that can be identified in a context. All Entities in an OpenID Connect federation MUST have a globally unique identifier.

Entity Identifier
: A globally unique URI that is bound to one Entity.

Trust Anchor
: An Entity that represents a trusted third party.

Entity Statement
: A signed JWT that contains the information needed for an Entity to participate in federation(s), including metadata about itself and policies that apply to other Entities that it is authoritative for.

Entity Configuration
: An Entity Statement issued by an Entity about itself. It contains the Entity's signing keys and further data used to control the Trust Chain resolution process, such as authority hints.

Entity Type
: The type of an Entity expresses its roles and functions within a federation. An Entity MUST be of at least one type and MAY be of many types. For example, an Entity can be both an OpenID Provider and Relying Party at the same time.

Federation Operator
: An organization that is authoritative for a federation. A federation operator administers the Trust Anchor(s) for Entities in its federation.

Intermediate Entity
: An Entity that issues an Entity Statement appearing somewhere in between those issued by the Trust Anchor and the Leaf Entity in a Trust Chain. The terms Intermediate Entity and Intermediate are used interchangeably in this specification.

Leaf Entity
: An Entity defined by a protocol, e.g., OpenID Connect Relying Party or Provider.

Subordinate Entity
: An Entity accredited by a Trust Anchor or an Intermediate Entity, which can be a Leaf Entity but also an Entity that acts as Intermediate for other Entities.

Federation Entity Discovery
: A process that starts with an Entity Identifier and collects a number of Entity Statements until the chosen Trust Anchor is reached. From the collected Entity Statements, a Trust Chain is constructed and verified. The result of the Federation Entity Discovery is that the Leaf Entity's metadata is constructed from the Trust Chain.

Trust Chain
: A sequence of Entity Statements that represents a chain starting at a Leaf Entity and ending in a Trust Anchor.

Trust Mark
: Statement of conformance to a well-scoped set of trust and/or interoperability requirements as determined by an accreditation authority.

Federation Entity Keys
: Keys used for all the cryptogaphic signatures required by the trust mechanisms defined in this document. Every participant in a Federation publishes its Federation Entity Keys in its Entity Configuration.

2. Overall Architecture

The basic component is the Entity Statement, which is a cryptographically signed JSON Web Token (JWT) [RFC7519]. A set of Entity Statements can form a path from a Leaf Entity to a Trust Anchor. Entity Configurations issued by Entities about themselves control the Trust Chain resolution process.

The Entity Configuration of a Leaf Entity contains one or more references to its superiors, found in the authority_hints (Section 3.1) parameter. These references can be used to download the Entity Configuration of each superior. One or more Entity Configurations can occur during the Federation Entity Discovery until the Trust Anchor is reached.

The Trust Anchor and its Intermediates issue Entity Statements about their Subordinates. The sequence of Entity Configurations and Entity Statements that validate the relationship between a superior and a Subordinate, along a path towards the Trust Anchor, forms the proof that a Leaf Entity is part of the Federation configured by the Trust Anchor.

The chain that links the statements to one another is verifiable with the signature of each statement, as described in Section 3.2.

With a verified Trust Chain, the federation policy is finally applied and the metadata describing a Leaf Entity within the Federation is then derived. How this is done is described in Section 5.

This specification deals with trust operations; it doesn't cover or touch protocol operations outside those of metadata derivation and exchange. In OpenID Connect terms, these are protocol operations other than OpenID Connect Discovery 1.0 [OpenID.Discovery] and OpenID Connect Dynamic Client Registration 1.0 [OpenID.Registration].

OpenID Connect is used in all of the examples in this specification, however this doesn't mean that this specification can only be used together with OpenID Connect. On the contrary, it can also be used to build federations based on any other protocols.

2.1. Cryptographic Trust Mechanism

The objects defined by this specification that are used to establish cryptographic trust between participants are secured as signed JWTs using public key cryptography. In particular, the keys used for securing these objects are managed by the Entities controlling those objects, with the public keys securing them being

distributed through those objects themselves. This kind of trust mechanism has been utilized by research and academic federations for over a decade.

Note that this cryptographic trust mechanim intentionally does not rely on Web PKI / TLS certificates for signing keys. Which TLS certificates are considered trusted can vary considerably between systems depending upon which certificate authorities are considered trusted and there are have been notable examples of ostensibly trusted certificates being compromised. For those reasons, this specification explicitly eschews reliance on Web PKI in favor of self-managed public keys, in this case, keys represented as JSON Web Keys (JWKs) [RFC7517].

3. Components

3.1. Entity Statement

An Entity Statement contains the information needed for the Entity that is the subject of the Entity Statement to participate in federation(s). An Entity Statement is always a signed JWT. The subject of the JWT is the Entity itself. The issuer of the JWT is the party that issued the Entity Statement. All Entities in a federation SHOULD be prepared to publish an Entity Statement about themselves (Entity Configuration).

Entity Statement JWTs MUST be explicitly typed, by setting the `typ` header parameter to `entity-statement+jwt`. This prevents cross-JWT confusion (see [RFC8725], section 3.11).

An Entity Statement is composed of the following claims:

iss
: REQUIRED. The Entity Identifier of the issuer of the statement. If the `iss` and the `sub` are identical, the issuer is making a statement about itself and this Entity Statement is an Entity Configuration.

sub
: REQUIRED. The Entity Identifier of the subject.

iat
: REQUIRED. The time the statement was issued. Its value is a JSON number representing the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time. See [RFC3339] for details regarding date/times in general and UTC in particular.

exp
: REQUIRED. Expiration time on or after which the statement MUST NOT be accepted for processing. Its value is a JSON number representing the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time.

jwks
: REQUIRED. A JSON Web Key Set (JWKS) [RFC7517] representing the public part of the subject Federation Entity signing keys. The corresponding private key is used by Entities to sign Entity Configurations about themselves, and Intermediate Entities to sign Entity Statements about their Subordinates. The keys that can be found here are intended to verify the signature of the issued Entity Statements and Trust Marks and SHOULD NOT be used in other protocols. Keys to be used in other protocols, such as OpenID Connect, are conveyed in the `metadata` elements of the respective Entity Statement. This claim is only OPTIONAL for the Entity Statement returned from an OP when the client is doing Explicit Registration. In all other cases it is REQUIRED. Every JWK in the JWK Set MUST have a Key ID (`kid`). It is RECOMMENDED that the Key ID be the JWK Thumbprint [RFC7638] using the SHA-256 hash function of the key.

aud
: OPTIONAL. The Entity Statement MAY be specifically created for an Entity. The Entity Identifier for that Entity MUST appear in this claim.

authority_hints
: OPTIONAL. An array of strings representing the Entity Identifiers of Intermediate Entities or Trust Anchors that MAY issue an Entity Statement about the issuer Entity. For all Entities except for Trust Anchors that do not have any superiors, this is REQUIRED and MUST NOT be the empty list [ ]. This claim MUST be absent from an Entity Configuration issued by a Trust Anchor with no superiors.

metadata
: REQUIRED. JSON object that includes protocol-specific metadata claims that represent the Entity's Type and metadata. Each key of the JSON object represents a metadata type identifier, and each value MUST be a JSON object representing the metadata according to the metadata schema of that metadata type. Such a JSON object may be empty. An Entity Statement MAY contain multiple metadata statements, but only one for each metadata type. If the Entity Statement is an Entity Configuration, then the Entity Statement SHOULD contain a `metadata` claim. In some cases, a superior may want to be explicit about what metadata should be used for a Subordinate. In that case `metadata` can be used. If `metadata` is used it is only valid for the subject of the Entity Statement and has no impact on its Subordinates. If a `metadata` claim appears beside a `metadata_policy` claim in an Entity Statement, then for each Entity Type, claims that appear in `metadata` MUST NOT appear in `metadata_policy`

metadata_policy
: OPTIONAL. JSON object that describes a metadata policy. A `metadata_policy` applies to the Entity described in this Entity Statement as well as to all Entities that appear beneath the Entity described in this Entity Statement. Applying to Subordinate Entities distinguishes `metadata_policy` from `metadata`, which only applies to the subject itself. Each key of the JSON object represents a metadata type identifier, and each value MUST be a JSON object representing the metadata policy according to the metadata schema of that metadata type. An Entity Statement MAY contain multiple metadata policy statements, but only one for each metadata type. Only non Leaf Entities MAY contain a `metadata_policy` claim. Entity Configurations MUST NOT contain a `metadata_policy` claim.

constraints
: OPTIONAL. JSON object that describes a set of Trust Chain constraints. There is more about this in Section 5.2.

crit
: OPTIONAL. The `crit` (critical) Entity Statement claim indicates that extensions to Entity Statement claims defined by this specification are being used that MUST be understood and processed. It is used in the same way that `crit` is used for extension JWS [RFC7515] header parameters that MUST be understood and processed. Its value is an array listing the Entity Statement claims present in the Entity Statement that use those extensions. If any of the listed extension Entity Statement claims are not understood and supported by the recipient, then the Entity Statement is invalid. Producers MUST NOT include Entity Statement claim names defined by this specification or names that do not occur as Entity Statement claim names in the Entity Statement in the `crit` list. Producers MUST NOT use the empty list [ ] as the `crit` value.

policy_language_crit
: OPTIONAL. The `policy_language_crit` (critical) Entity Statement claim indicates that extensions to the policy language defined by this specification are being used that MUST be understood and processed. It is used in the same way that `crit` is used for extension JSON Web Signature (JWS) [RFC7515] header parameters that MUST be understood and processed. Its value is an array listing the policy language extensions present in the policy language statements that use those extensions. If any of the listed extension policy language extensions are not understood and supported by the recipient, then the Entity

Statement is invalid. Producers MUST NOT include policy language names defined by this specification or names that do not occur in policy language statements in the Entity Statement in the `policy_language_crit` list. Producers MUST NOT use the empty list [ ] as the `policy_language_crit` value.

trust_marks
: OPTIONAL. A JSON array of objects, each representing a Trust Mark. Each JSON object MUST contain the following two claims and MAY contain other claims if needed. All the claims in the JSON object MUST have the same values as those contained in the Trust Mark JSON Web Token.

  id
  : The Trust Mark unique identifier. It MUST be the same value as the `id` claim contained in the Trust Mark JSON Web Token.

  trust_mark
  : A signed JSON Web Token that represents a Trust Mark.

  There is more about Trust Marks in Section 5.3.

trust_marks_issuers
: OPTIONAL. A Trust Anchor MAY use this claim to tell which combination of Trust Mark identifiers and issuers are trusted by the federation. This claim MUST be ignored if present in an Entity Configuration that describes an Entity that is not a Trust Anchor. It is a JSON object with keys representing Trust Mark identifiers and values being an array of trusted Entities representing the accreditation authority. If the value list bound to a Trust Mark identifier is empty, anyone can issue Trust Marks with that identifier. There is more about Trust Marks in Section 5.3.

trust_anchor_id
: OPTIONAL. An OP MUST use this claim to tell the RP which Trust Anchor it chose to use when responding to an explicit client registration (Explicit Client Registration (Section 10.2)). The value of `trust_anchor_id` is the Entity Identifier of a Trust Anchor.

The Entity Statement is signed using the private key of the issuer Entity in the form of a JSON Web Signature (JWS) [RFC7515]. Implementations SHOULD support signature verification with the RSA SHA-256 algorithm because OpenID Connect Core requires support for it (`alg` value of `RS256`). Federations MAY also specify different mandatory-to-implement algorithms.

The following is a non-normative example of a `trust_marks_issuers` claim value:

```
{
  "https://openid.net/certification/op": ["*"],
  "https://refeds.org/wp-content/uploads/2016/01/Sirtfi-1.0.pdf":
    ["https://swamid.se"]
}
```

The following is a non-normative example of an Entity Statement before adding a signature. The example contains a critical extension `jti` (JWT ID) to the Entity Statement and one critical extension to the policy language `regexp` (Regular expression).

```
{
  "iss": "https://feide.no",
  "sub": "https://ntnu.no",
  "iat": 1516239022,
  "exp": 1516298022,
  "crit": ["jti"],
  "jti": "7l2lncFdY6SlhNia",
  "policy_language_crit": ["regexp"],
  "metadata": {
    "openid_provider": {
      "issuer": "https://ntnu.no",
      "organization_name": "NTNU",
    },
    "oauth_client": {
      "organization_name": "NTNU"
    }
  },
  "metadata_policy": {
    "openid_provider": {
      "id_token_signing_alg_values_supported":
        {"subset_of": ["RS256", "RS384", "RS512"]},
      "op_policy_uri": {
        "regexp":
          "^https:\/\/[\\w-]+\\.example\\.com\/[\\w-]+\\.html"}
    },
    "oauth_client": {
      "grant_types": {
        "subset_of": ["authorization_code", "client_credentials"]
      },
  },
  "constraints": {
    "max_path_length": 2
  },
  "jwks": {
    "keys": [
      {
        "alg": "RS256",
        "e": "AQAB",
        "key_ops": ["verify"],
        "kid": "key1",
        "kty": "RSA",
        "n": "pnXBOusEANuug6ewezb9J_...",
        "use": "sig"
      }
    ]
  }
}
```

3.2. Trust Chain

Entities whose statements build a Trust Chain are categorized as:

Trust anchor
: An Entity that represents a trusted third party.

Leaf
: In an OpenID Connect identity federation, an RP or an OP.

Intermediate
: Neither a Leaf Entity nor a Trust Anchor.

A Trust Chain begins with a Leaf Entity Configuration, and has zero or more Entity Statements issued by Intermediates about Subordinates, including the Entity Statement issued by the Trust Anchor about the top-most Intermediate (if there are Intermediates) or the Leaf Entity (if there are no Intermediates). The Trust Chain MAY end with the Entity Configuration of the Trust Anchor.

If present, the Trust Anchor's Entity Configuration at the end of the chain contains the configuration of the federation, such as the public keys of the Trust Anchor and other parameters considered within the trust evaluation. When this is the case, the Trust Chain consequentially contains the configuration of the federation at the time of the evaluation of the chain.

A simple example: If we have an RP that belongs to Organization A that is a member of Federation F, the Trust Chain for such a setup will contain the following Entity Statements:

1. an Entity Configuration about the RP published by the RP,
2. an Entity Statement about the RP published by Organization A,
3. an Entity Statement about Organization A published by Federation F.

A Trust Chain MUST always be possible to order such that: If we name the Entity Statements ES[0] (the Leaf Entity's Entity Configuration) to ES[i] (an Entity Statement issued by the Trust Anchor), i>0 then:

- The `iss` Entity in one Entity Statement is always the `sub` Entity in the next. ES[j]['iss'] == ES[j+1]['sub'], j=0,...,i-1 .
- There MUST always be a Federation Entity Key carried in the `jwks` claim in ES[j] that can be used to verify the signature of ES[j-1], j=i,...,1 .
- It MUST be possible to verify the signature of ES[0] with one of the keys in ES[0]['jwks'].

The signing key that MUST be used to verify ES[i] is distributed from the Trust Anchors to any Entity that needs to verify a Trust Chain in some secure out-of-band way not described in this document.

3.2.1. Trust Chain Header Parameter

This JWS [RFC7515] header parameter is registered in the IANA "JSON Web Signature and Encryption Header Parameters" registry established by Section 12.6.

The Section 3.2 header parameter is a JSON array containing the sequence of the statements that proves the trust relationship between the issuer of the JWS [RFC7515] and the selected Trust Anchor, sorted as shown in Section 3.2. The Trust Chain contains the public key used to verify the JWS [RFC7515]. The issuer of the JWS SHOULD select the Trust Anchor that it has in common with the audience of the JWS, otherwise the issuer is free to select the Trust Anchor it deems most significant. Use of this header parameter is OPTIONAL.

4. Metadata Type Identifiers

This specification defines metadata type identifiers for OpenID Connect and OAuth 2.0 objects. The metadata type identifier uniquely identifies the Entity Type of the participant and the format of metadata for that Entity Type. It reuses existing OpenID Connect and OAuth 2.0 metadata standards that are applicable to each metadata type. Each metadata document is a JSON object.

This specification also allows new metadata types to be defined to support use cases outside OpenID Connect and OAuth 2.0 federations.

4.1. OpenID Connect and OAuth 2.0 Metadata Extensions

For the OpenID Connect and OAuth 2.0 metadata types below, the following additional metadata properties are defined, which MAY be used with all those metadata types:

organization_name
: OPTIONAL. A human-readable name representing the organization owning the Entity. If the owner is a physical person, this MAY be, for example, the person's name. Note that this information will be publicly available.

signed_jwks_uri
: OPTIONAL. A URI pointing to a signed JWT having the Entity's JWK Set as its payload. The JWT MUST be signed using a Federation Entity Key. A signed JWT can contain the following claims, all except `keys` defined in [RFC7519]:

  keys
  : REQUIRED. List of JWK values.

  iss
  : REQUIRED. The "iss" (issuer) claim identifies the principal that issued the JWT.

  sub
  : REQUIRED. This claim identifies the owner of the keys. It SHOULD be the same as the issuer.

  iat
  : OPTIONAL. This claim identifies the time at which the JWT was issued.

  exp
  : OPTIONAL. This claim identifies the time at which the JWT is no longer valid.

  More claims are defined in [RFC7519]; of these, `aud` SHOULD NOT be used since the issuer cannot know who the audience is. `nbf` and `jti` are deemed to not be very useful in this context and are to be omitted.

The following is a non-normative example of a signed JWKS before serialization and adding a signature.

```
{
  "keys": [
    {
      "kty": "RSA",
      "kid": "SUdtUndEWVY2cUFDeDV5NVlBWDhvOXJodVl2am1mNGNtR0pmd",
      "n": "y_Zc8rByfeRIC9fFZrDZ2MGH2ZnxLrc0ZNNwkNet5rwCPYeRF3Sv
            5nihZA9NHkDTEX97dN8hG6ACfeSo6JB2P7heJtmzM8oOBZbmQ90n
            EA_JCHszkejHaOtDDfxPH6bQLrMlItF4JSUKua301uLB7C8nzTxm
            tF3eAhGCKn8LotEseccxsmzApKRNWhfKDLpKPe9i9PZQhhJaurwD
            kMwbWTAeZbqCScU1o09piuK1JDf2PaDFevioHncZcQO74Obe4nN3
            oNPNAxrMClkZ9s9GMEd5vMqOD4huXlRpHwm9V3oJ3LRutOTxqQLV
            yPucu7eHA7her4FOFAiUk-5SieXL9Q",
      "e": "AQAB"
    },
    {
      "kty": "EC",
      "kid": "MFYycG1raTI4SkZvVDBIMF9CNGw3VEZYUmxQLVN2T21nSWlkd3",
      "crv": "P-256",
      "x": "qAOdPQROkHfZY1daGofOmSNQWpYK8c9G2m2Rbkpbd4c",
      "y": "G_7fF-T8n2vONKM15Mzj4KR_shvHBxKGjMosF6FdoPY"
    }
  ],
  "iss": "https://example.org/op",
  "iat": 1618410883
}
```

The following is a non-normative example of an RP's Entity Configuration:

```
{
  "iss": "https://openid.sunet.se",
  "sub": "https://openid.sunet.se",
  "iat": 1516239022,
  "exp": 1516298022,
  "metadata": {
    "openid_relying_party": {
      "application_type": "web",
      "redirect_uris": [
        "https://openid.sunet.se/rp/callback"
      ],
      "organization_name": "SUNET",
      "logo_uri": "https://www.sunet.se/sunet/images/32x32.png",
      "grant_types": [
        "authorization_code",
        "implicit"
      ],
      "signed_jwks_uri":"https://openid.sunet.se/rp/signed_jwks.jose",
      "jwks_uri": "https://openid.sunet.se/rp/jwks.json"
    }
  },
  "jwks": {
    "keys": [
      {
        "alg": "RS256",
        "e": "AQAB",
        "key_ops": [
          "verify"
        ],
        "kid": "key1",
        "kty": "RSA",
        "n": "pnXBOusEANuug6ewezb9J_...",
        "use": "sig"
      }
    ]
  },
  "authority_hints": [
    "https://edugain.org/federation"
  ]
}
```

jwks
: OPTIONAL. JSON Web Key Set document, passed by value, as defined in [RFC7517]. This parameter is intended to be used by participants that, for some reason, cannot use the `signed_jwks_uri` parameter. One significant downside of `jwks` is that it does not enable key rotation (which `signed_jwks_uri` and `jwks_uri` do).

4.1.1. Usage of jwks, jwks_uri, and signed_jwks_uri

It is RECOMMENDED that an Entity Configuration use only one of `jwks`, `jwks_uri`, and `signed_jwks_uri` in its OpenID Connect or OAuth 2.0 metadata. However, there may be circumstances in which is it desirable to use multiple JWK Set representations, such as when an Entity is in multiple federations and the federations have different policies about the JWK Set representation to be used. When multiple JWK Set representations are

used, the keys present in each representation SHOULD be the same. Even if they are not completely the same at a given instant in time (which may be the case during key rollover operations), implementations MUST make them consistent in a timely manner.

4.2. OpenID Connect Relying Party

The metadata type identifier is `openid_relying_party`.

All parameters defined in Section 2 of OpenID Connect Dynamic Client Registration 1.0 [OpenID.Registration] and Section 4.1 are applicable, as well as additional parameters registered in the IANA "OAuth Dynamic Client Registration Metadata" registry [IANA.OAuth.Parameters].

In addition, the following RP metadata parameter is defined:

client_registration_types
: REQUIRED. An array of strings specifying the client registration types the RP wants to use. Values defined by this specification are `automatic` and `explicit`.

4.3. OpenID Provider

The metadata type identifier is `openid_provider`.

All parameters defined in Section 3 of OpenID Connect Discovery 1.0 [OpenID.Discovery] and Section 4.1 are applicable, as well as additional parameters registered in the IANA "OAuth Authorization Server Metadata" registry [IANA.OAuth.Parameters].

In addition, the following OP metadata parameters are defined:

client_registration_types_supported
: REQUIRED. Array specifying the federation types supported. Federation-type values defined by this specification are `automatic` and `explicit`.

federation_registration_endpoint
: OPTIONAL. URL of the OP's federation-specific Dynamic Client Registration Endpoint. If the OP supports explicit client registration as described in Section 10.2, then this claim is REQUIRED.

request_authentication_methods_supported
: OPTIONAL. A JSON Object defining the client authentications supported for each endpoint. The endpoint names are defined in the IANA "OAuth Authorization Server Metadata" registry [IANA.OAuth.Parameters]. Other endpoints and authentication methods are possible if made recognizable according to established standards and not in conflict with the operating principles of this specification. In OpenID Connect Core, no client authentication is performed at the authentication endpoint. Instead, the request itself is authenticated. The OP maps information in the request (like the redirect_uri) to information it has gained on the client through static or dynamic registration. If the mapping is successful, the request can be processed. If the RP uses Automatic Registration, as defined in Section 10.1, the OP has no prior knowledge of the RP. Therefore, the OP must start by gathering information about the RP using the process outlined in Section 6. Once it has the RP's metadata, the OP can verify the request in the same way as if it had known the RP's metadata beforehand. To make the request verification more secure, we demand the use of a client authentication or verification method that proves that the RP is in possession of a key that appears in the RP's metadata.

  Client authentication methods that MAY be used include:

private_key_jwt
: `private_key_jwt`. This authentication process is described in Section 9 of OpenID Connect Core 1.0 [OpenID.Core]. Note that if `private_key_jwt` is used, the audience of the signed JWT MUST be either the URL of the Authorization Server's Authorization Endpoint or the Authorization Server's Entity Identifier.

tls_client_auth
: Section 2.1 of [RFC8705].

self_signed_tls_client_auth
: Section 2.2 of [RFC8705].

A client verification method, on the other hand, is something like:

request_object
: A Request Object, as described in OpenID Connect Core 1.0 [OpenID.Core]. There is more about this in Section 10.1. Here the verification is based on the fact that the request object is signed with a key the RP controls.

The claim value of this parameter is a JSON object with members representing authentication request methods and as values lists of client authentication/verification methods that can be used together with the authentication request method.

Examples of authentication request methods are

- Authentication Request (AR), as described in Section 3 of OpenID Connect Core 1.0 [OpenID.Core], and
- Pushed Authorization Request (PAR), as described in [RFC9126].

If AR is used, then a client verification method like `request_object` can be used. If PAR is used, then client authentication methods like `private_key_jwt`, `tls_client_auth` and `self_signed_tls_client_auth` can be used.

request_authentication_signing_alg_values_supported
: OPTIONAL. JSON array containing a list of the supported JWS [RFC7515] algorithms (`alg` values) for signing the JWT [RFC7519] used in the Request Object contained in the request parameter of an authorization request or in the `private_key_jwt` of a pushed authorization request. This entry MUST be present if either of these authentication methods are specified in the `request_authentication_methods_supported` entry. No default algorithms are implied if this entry is omitted. Servers SHOULD support RS256. The value none MUST NOT be used.

The following is a non-normative example of an OP's Entity Configuration:

```
{
   "iss":"https://op.umu.se",
   "sub":"https://op.umu.se",
   "exp":1568397247,
   "iat":1568310847,
   "metadata":{
      "openid_provider":{
         "issuer":"https://op.umu.se/openid",
         "signed_jwks_uri":"https://op.umu.se/openid/signed_jwks.jose",
         "authorization_endpoint":"https://op.umu.se/openid/authorization",
         "client_registration_types_supported":[
            "automatic",
            "explicit"
         ],
         "grant_types_supported":[
            "authorization_code",
            "implicit",
            "urn:ietf:params:oauth:grant-type:jwt-bearer"
         ],
         "id_token_signing_alg_values_supported":[
            "ES256",
            "RS256"
         ],
         "logo_uri":"https://www.umu.se/img/umu-logo-left-neg-SE.svg",
         "op_policy_uri":"https://www.umu.se/en/legal-information/",
         "response_types_supported":[
            "code",
            "code id_token",
            "token"
         ],
         "subject_types_supported":[
            "pairwise",
            "public"
         ],
         "token_endpoint":"https://op.umu.se/openid/token",
         "federation_registration_endpoint":"https://op.umu.se/openid/fedreg",
         "token_endpoint_auth_methods_supported":[
            "client_secret_post",
            "client_secret_basic",
            "client_secret_jwt",
            "private_key_jwt"
         ],
         "pushed_authorization_request_endpoint":"https://op.umu.se/openid/par",
         "request_authentication_methods_supported": {
            "authorization_endpoint": [
                "request_object"
            ],
            "pushed_authorization_request_endpoint": [
                "request_object",
                "private_key_jwt",
                "tls_client_auth",
                "self_signed_tls_client_auth"
            ]
        }
      }
   },
   "authority_hints":[
      "https://umu.se"
   ],
   "jwks":{
      "keys":[
         {
            "e":"AQAB",
```

```
            "kid":"dEEtRjlzY3djcENuT01wOGxrZlkxb3RIQVJlMTY0...",
            "kty":"RSA",
            "n":"x97YKqc9Cs-DNtFrQ7_vhXoH9bwkDWW6En2jJ044yH..."
         }
      ]
   },
}
```

4.4. OAuth Authorization Server

The metadata type identifier is `oauth_authorization_server`.

All parameters defined in Section 2 of RFC 8414 [RFC8414] and Section 4.1 are applicable, as well as additional parameters registered in the IANA "OAuth Dynamic Client Registration Metadata" registry [IANA.OAuth.Parameters].

4.5. OAuth Client

The metadata type identifier is `oauth_client`.

All parameters defined in Section 2 of OAuth 2.0 Dynamic Client Registration Protocol [RFC7591] and Section 4.1 are applicable, as well as additional parameters registered in the IANA "OAuth Dynamic Client Registration Metadata" registry [IANA.OAuth.Parameters].

4.6. OAuth Protected Resource

The metadata type identifier is `oauth_resource`. The parameters defined in Section 4.1 are applicable. In addition, while it is not presently a standard, some deployments are using the protected resource metadata parameters defined in [I-D.jones-oauth-resource-metadata].

4.7. Federation Entity

The metadata type identifier is `federation_entity`.

All Entities in a federation MAY expose this metadata type. The Entities that expose federation API endpoints MUST expose this metadata type.

The following properties are allowed:

federation_fetch_endpoint
: OPTIONAL. The fetch endpoint described in Section 7.1. Intermediate Entities and Trust Anchors MUST publish a `federation_fetch_endpoint`. Leaf Entities MUST NOT.

federation_list_endpoint
: OPTIONAL. The list endpoint described in Section 7.3. Intermediate Entities and Trust Anchors MUST publish a `federation_list_endpoint`. Leaf Entities MUST NOT.

federation_resolve_endpoint
: OPTIONAL. The resolve endpoint described in Section 7.2. Any federation Entity MAY publish a `federation_resolve_endpoint`.

federation_trust_mark_status_endpoint
: OPTIONAL. The Trust Mark status endpoint described in Section 7.4. Trust mark issuers SHOULD publish a `federation_trust_mark_status_endpoint`.

federation_trust_mark_list_endpoint
: OPTIONAL. The endpoint described in Section 7.5. Trust Mark issuers MAY publish a `federation_trust_mark_list_endpoint`.

organization_name
: OPTIONAL. A human-readable name representing the organization owning the Entity. If the owner is a physical person this MAY be, for example, the person's name. Note that this information will be publicly available. It is RECOMMENDED that each federation Entity expose this claim.

contacts
: OPTIONAL. JSON array with one or more strings representing contact persons at the Entity. These MAY contain names, e-mail addresses, descriptions, phone numbers, etc.

logo_uri
: OPTIONAL. String. A URL that points to the logo of this Entity. The file containing the logo SHOULD be published in a format that can be viewed via the web.

policy_uri
: OPTIONAL. URL to the documentation of conditions and policies relevant to this Entity.

homepage_uri
: OPTIONAL. URL to a generic home page representing this Entity.

Example

```
"federation_entity": {
  "federation_fetch_endpoint":
    "https://example.com/federation_fetch",
  "federation_list_endpoint":
    "https://example.com/federation_list",
  "federation_trust_mark_status_endpoint": "https://example.com/status",
  "federation_trust_mark_list_endpoint": "https://example.com/trust_marked_list",
  "organization_name": "The example cooperation",
  "homepage_uri": "https://www.example.com"
}
```

5. Federation Policy

5.1. Metadata Policy

An Entity MAY publish metadata policies pertaining to Subordinate Entities of a specific type. Entity Type identifiers specified in this document can be found in Section 4.

5.1.1. metadata_policy

A `metadata_policy` for a specific Entity type has the following structure:

- It consists of one or more policy entries.
- Each policy entry applies to one metadata parameter, such as `id_token_signed_response_alg`.
- Each policy entry consists of one or more operators, which can be value modifiers or value checks.
- An operator can only appear once in a policy entry.

It SHOULD be noted that claim names without language tags are different from the same claim but with language tags.

An example of a policy entry:

```
"id_token_signed_response_alg": {
  "default": "ES256",
  "one_of" : ["ES256", "ES384", "ES512"]
}
```

Which fits into a metadata policy like this:

```
"metadata_policy" : {
  "openid_relying_party": {
    "id_token_signed_response_alg": {
      "default": "ES256",
      "one_of" : ["ES256", "ES384", "ES512"]
    }
  }
}
```

5.1.2. Operators

Value modifiers are:

value
: Disregarding what value the parameter has, if any, the metadata parameter MUST be set to the specified value.

add
: The specified value or values MUST be added to the metadata parameter. If any of the specified values are already present as values of the parameter, they MUST NOT be added. If the parameter has no value, then the parameter MUST be initialized with the specified value(s).

default
: If the metadata parameter has no value, then it MUST be set to the specified default value(s).

Value checks are:

essential
: If `true`, then the parameter MUST have a value. If `false`, the parameter is voluntary and is not required to have a value. If essential is missing as an operator, it is treated as if set to `false`.

one_of
: If the metadata parameter has no value and is voluntary (`essential` policy is missing or set to `false`), then the `one_of` check MUST be ignored. Otherwise the value of the metadata parameter MUST be one of those listed in the directive.

subset_of
: If the metadata parameter has no value and is voluntary (`essential` policy is missing or set to `false`), then the `subset_of` check MUST be ignored. Otherwise, this operator MUST compute the intersection of the values in the directive and the values of the parameter. If the resulting intersection is non-empty the parameter MUST be set to the values in the intersection. Note that this behavior makes `subset_of` a potential value modifier and thus it is not a pure value check. If the intersection is empty the outcome depends on the `essential` parameter : when the value is defined as essential, the parameter MUST be rejected with a policy error, else the parameter MUST be removed (implying a `null` value).

superset_of
: If the metadata parameter has no value and is voluntary (`essential` policy is missing or set to `false`), then the `superset_of` check MUST be ignored. Otherwise the values of the metadata parameter MUST contain those specified in the directive. By mathematically defining supersets, equality is included.

A "set equals" value check can be expressed by combining a `subset_of` and a `superset_of` with identical string list values.

Note that when a parameter is defined as a space-separated list of strings like for `scope` in [RFC7591] `subset_of`, `superset_of` and `default` are still expressed as lists of strings. This language from [RFC6749] also applies to such space-separated lists of strings: "If the value contains multiple space-delimited strings, their order does not matter, and each string adds an additional access range to the requested scope."

Policy		Metadata parameter	
essential	subset_of	input	output
true	[a,b,c]	[a,e]	[a]
false	[a,b,c]	[a,e]	[a]
true	[a,b,c]	[d,e]	error
false	[a,b,c]	[d,e]	no parameter
true	[a,b,c]	null or no parameter	error
false	[a,b,c]	null or no parameter	no parameter

Table 1: Map of the outputs produced by combining metadata policies with different input values.

5.1.3. Restrictions on Policy Entries

As stated, a policy entry can contain one or more operators. Not all operators are allowed to appear together in a policy entry.

subset_of/superset_of and one_of
: `subset_of` and `superset_of` apply to parameters that can have more than one value (for instance, `contacts`), while `one_of` applies to parameters that can only have one value (for instance, `id_token_signed_response_alg`). Therefore, `one_of` cannot appear beside `subset_of`/ `superset_of` in a policy entry.

value
: `value` overrides everything else. So having `value` together with any other operator (except for `essential`) does not make sense.

Other restrictions are:

- If `subset_of` and `superset_of` both appear as operators, then the list of values in `subset_of` MUST be a superset of the values in `superset_of`.
- If `add` appears in a policy entry together with `subset_of` then the value/values of `add` MUST be a subset of `subset_of`.
- If `add` appears in a policy entry together with `superset_of` then the values of `add` MUST be a superset of `superset_of`.

- If `default` appears in a policy entry together with `subset_of` then the values of `default` MUST be a subset of `subset_of`.
- If `default` appears in a policy entry together with `superset_of` then the values of `default` MUST be a superset of `superset_of`.
- If `add` appears in a policy entry together with `one_of` then the value of `add` MUST be a member of `one_of`.
- If `default` appears in a policy entry together with `one_of` then the value `default` MUST be a member of `one_of`.

5.1.4. Combining Policies

If there is more than one metadata policy in a Trust Chain, then the policies MUST be combined before they are applied to the metadata statement.

Using the notation we have defined in Section 3.2, policies are combined starting with ES[i] and then adding the policies from ES[j] j=i-1,..,1 before applying the combined policy to the Entity's metadata.

After each combination, the policy for each parameter MUST adhere to the rules defined in Section 5.1.3.

5.1.4.1. Merging Operators

subset_of
: The result of merging the values of two `subset_of` operators is the intersection of the operator values.

one_of
: The result of merging the values of two `one_of` operators is the intersection of the operator values.

superset_of
: The result of merging the values of two `superset_of` operators is the union of the operator values.

add
: The result of merging the values of two `add` operators is the union of the values.

value
: Merging two `value` operators is NOT allowed unless the two operator values are equal.

default
: Merging two `default` operators is NOT allowed unless the two operator values are equal.

essential
: If a superior has specified `essential=true`, then an Intermediate cannot change that. If a superior has specified `essential=false`, then an Intermediate is allowed to change that to `essential=true`. If a superior has not specified `essential`, then an Intermediate can set `essential` to `true` or `false`.

5.1.5. Applying Policies

Once combining the Metadata policies has been accomplished, the next step is to apply the combined policy to the metadata.

Doing that, one follows these steps for each parameter in the policy.

1. If there is a `value` operator in the policy, its value MUST be applied to the parameter and further action stopped.
2. Add whatever value is specified in an `add` operator.
3. If the parameter still has no value, apply the `default` if there is one.

4. Do the essential check. If `essential` is `true` the parameter MUST have a value, else the operator MUST fail. If the parameter is voluntary and has no value any other value checks MUST be skipped.
5. Do the other checks. Verify that the value is `one_of` or that the values are `subset_of`/`superset_of`. If the parameter values do not fall within the allowed boundaries, applying the operator MUST fail.
6. If more than one value is specified in `one_of` and the Leaf Entity does not specify a metadata parameter value, it is an implementation choice which value in the `one_of` array to use.

5.1.6. Policy Combination Example

A federation's policy for OAuth 2.0 clients:

```
{
  "grant_types": {

    "subset_of": ["authorization_code", "implicit", "refresh_token"],

    "superset_of": ["authorization_code"],

    "default": ["authorization_code", "refresh_token"]
  },
  "token_endpoint_auth_method": {
    "one_of": [
      "client_secret_post",
      "client_secret_basic"
    ]
  },
  "contacts": {
    "add": "helpdesk@federation.example.org"
  }
}
```

An organization's policy for OAuth 2.0 clients:

```
{
  "grant_types": {
    "subset_of": ["authorization_code", "refresh_token", "implicit"],
    "default": ["authorization_code", "refresh_token"]
  },
  "token_endpoint_auth_method": {
    "one_of": [
      "client_secret_post",
      "client_secret_basic"
    ],
    "default": "client_secret_basic"
  },
  "contacts": {
    "add": "helpdesk@org.example.org"
  }
}
```

The combined metadata policy then becomes:

```
{
  "grant_types": {
    "subset_of": ["authorization_code", "refresh_token"],
    "superset_of": ["authorization_code"],
    "default": ["authorization_code", "refresh_token"]
  },
  "token_endpoint_auth_method": {
    "one_of": [
      "client_secret_post",
      "client_secret_basic"
    ],
    "default": "client_secret_basic"
  },
  "contacts": {
    "add": [
      "helpdesk@federation.example.org",
      "helpdesk@org.example.org"
    ]
  }
}
```

5.1.7. Enforcing Policy

If applying policies to a metadata statement results in incorrect metadata, then such a metadata statement MUST be regarded as broken and MUST NOT be used.

5.1.8. Extending the Policy Language

There might be parties that want to extend the policy language defined here. If that happens, then the rule is that if software compliant with this specification encounters a keyword it does not understand, it MUST ignore it unless it is listed in a `policy_language_crit` list, as is done for JWS [RFC7515] header parameters with the `crit` parameter. If the policy language extension keyword is listed in the `policy_language_crit` list and not understood, then the metadata MUST be rejected.

5.1.9. Policy Example

The following is a non-normative example of a set of policies being applied to an RP's metadata.

The RP's metadata:

```
{
  "contacts": [
    "rp_admins@cs.example.com"
  ],
  "redirect_uris": [
    "https://cs.example.com/rp1"
  ],
  "response_types": [
    "code"
  ]
}
```

The federation's policy for RPs:

```
{
  "id_token_signed_response_alg": {
    "one_of": [
      "ES256",
      "ES384"
    ],
    "default": "ES256",
  },
  "response_types": {
    "subset_of": [
      "code",
      "code id_token"
    ]
  }
}
```

The organization's policy for RPs:

```
{
  "metadata_policy": {
    "openid_relying_party": {
      "contacts": {
        "add": "helpdesk@example.com"
      },
      "logo_uri": {
        "one_of": [
          "https://example.com/logo_small.svg",
          "https://example.com/logo_big.svg"
        ],
        "default": "https://example.com/logo_small.svg"
      }
    }
  },
  "metadata": {
    "openid_relying_party": {
      "policy_uri": "https://example.com/policy.html",
      "tos_uri": "https://example.com/tos.html"
    }
  }
}
```

After applying the policies above, the metadata for the Entity in question would become:

```
{
  "contacts": [
    "rp_admins@cs.example.com",
    "helpdesk@example.com"
  ],
  "logo_uri": "https://example.com/logo_small.svg",
  "policy_uri": "https://example.com/policy.html",
  "tos_uri": "https://example.com/tos.html",
  "id_token_signed_response_alg": "ES256",
  "response_types": [
    "code"
  ],
  "redirect_uris": [
    "https://cs.example.com/rp1"
  ]
}
```

5.2. Applying Constraints

A constraint specification can contain the following claims:

max_path_length
: OPTIONAL. Integer. The maximum number of Entity Statements between this Entity Statement and the last Entity Statement in the Trust Chain.

naming_constraints
: OPTIONAL. JSON object. Restriction on the Entity Identifiers of the Entities below this Entity. The behavior of this claim mimics what is defined in Section 4.2.1.10 of [RFC5280]. Restrictions are defined in terms of permitted or excluded name subtrees.

allowed_leaf_entity_types
: OPTIONAL. An array of strings. Each string is an Entity Type. Entity Type identifiers specified in this document can be found in Section 4. This constraint restricts what types of Leaf Entities MAY appear beneath the Entity described in this Entity Statement.

The following is a non-normative example of such a specification:

```
{
  "naming_constraints": {
    "permitted": [
      "https://.example.com"
    ],
    "excluded": [
      "https://east.example.com"
    ]
  },
  "max_path_length": 2,
  "allowed_leaf_entity_types": ["openid_provider", "openid_relying_party"]
}
```

If a Subordinate Entity Statement contains a constraint specification that is more restrictive than the one in effect, then the more restrictive constraint is in effect from here on.

If a Subordinate Entity Statement contains a constraint specification that is less restrictive than the one in effect, then it MUST be ignored.

5.2.1. Max Path Length

The `max_path_length` constraint specifies the maximum number of Entity Statements a Trust Chain can have between the Entity Statement that contains the constraint specification and the Leaf's Entity Statement.

A `max_path_length` constraint of zero indicates that no Intermediates MAY appear between this Entity and the Leaf Entity. Where it appears, the `max_path_length` constraint MUST have a value greater than or equal to zero.

Not adding a max_path_length constraint just means that there are no additional constraints apart from those already in effect.

Assuming that we have a Trust Chain with four Entity Statements:

1. Entity Configuration of the Leaf Entity (LE)
2. Entity Statement by Intermediate 1 (I1) about LE
3. Entity Statement by Intermediate 2 (I2) about I1
4. Entity Statement by Trust Anchor (TA) about I2

Then the Trust Chain fulfills the constraints if, for instance:

- The TA specifies a `max_path_length` that is equal to or bigger than 2.
- TA specifies a `max_path_length` of 2, I2 specifies a `max_path_length` of 1, and I1 sets no `max_path_length` constraint.
- Neither TA nor I2 specifies any `max_path_length` constraint while I1 sets `max_path_length` to 0.

The Trust Chain does not fulfill the constraints if, for instance, the:

- TA's Entity Statement has set the `max_path_length` to 1.

5.2.2. Naming Constraints

The `naming_constraints` member specifies a namespace within which all subject Entity Identifiers in Subordinate Entity Statements in a Trust Chain MUST be located.

Restrictions are defined in terms of permitted or excluded name subtrees. Any name matching a restriction in the excluded claim is invalid regardless of the information appearing in the permitted claim.

Domain name constraints are as specified in Section 4.2.1.10 of [RFC5280]. As stated there, a domain name constraint MUST be specified as a fully qualified domain name and MAY specify a host or a domain. Examples are "host.example.com" and ".example.com". When the domain name constraint begins with a period, it MAY be expanded with one or more labels. That is, the domain name constraint ".example.com" is satisfied by both host.example.com and my.host.example.com. However, the domain name constraint ".example.com" is not satisfied by "example.com". When the domain name constraint does not begin with a period, it specifies a host.

5.2.3. Leaf Entity Type Constraints

The `allowed_leaf_entity_types` constraint specifies what Entity Types are possible for Leaf Entities beneath an Entity. If there is no `allowed_leaf_entity_types` defined it means that any type is allowed.

5.3. Trust Marks

The Trust Marks used by this specification are signed JWTs.

The Trust Marks are signed by a federation-accredited authority.

The key used by the Trust Mark Issuer to sign its Trust Marks MUST be one of the private keys related to its `Federation Entity Keys`.

Note that a federation MAY allow an Entity to self-sign some Trust Marks.

Trust Mark JWTs MUST be explicitly typed by setting the `typ` header parameter to `trust-mark+jwt`. This is done to prevent cross-JWT confusion (see [RFC8725], section 3.11).

5.3.1. Trust Mark Claims

These are the Trust Mark Claims:

iss
: REQUIRED. String. The issuer of the Trust Mark.

sub
: REQUIRED. String. The Entity this Trust Mark applies to.

id
: REQUIRED. String. An identifier of the Trust Mark. The Trust Mark `id` MUST be collision-resistant across multiple federations. It is RECOMMENDED that the `id` value is built using the unique URL that uniquely identifies the federation, or the trust framework, within which it was issued. This is required to prevent Trust Marks issued in different federations from having colliding identifiers.

iat
: REQUIRED. Number. When this Trust Mark was issued. Expressed as Seconds Since the Epoch, per [RFC7519].

logo_uri
: OPTIONAL. String. A URL that points to a mark/logo that the subject is allowed to display to a user of the Entity.

exp
: OPTIONAL. Number. When this Trust Mark is not valid anymore. Expressed as Seconds Since the Epoch, per [RFC7519]. If not present, it means that the Trust Mark is valid forever.

ref
: OPTIONAL. String. URL that points to information connected to the issuance of this Trust Mark.

Other claims MAY be used in conjunction with the claims outlined above. The recommendations for claim naming described in Section 5.1.2 of OpenID Connect Core 1.0 [OpenID.Core] apply.

5.3.2. Validating a Trust Mark

An Entity SHOULD NOT try to validate a Trust Mark until it knows which Trust Anchors it wants to use.

Validating a Trust Mark issuer follows the procedure set out in Section 8. Validating the Trust Mark itself is described in Section 7.4

Note that the Entity representing the accreditation authority SHOULD be well known and trusted for a given Trust Mark identifier. A Trust Anchor MAY publish a list of accreditation authorities of Trust Marks that SHOULD be trusted by other Entities. A Trust Anchor uses the `trust_marks_issuers` claim in its Entity Configuration to publish this information.

Trust Marks that are not recognized within a federation SHOULD be ignored when evaluating the trust in the Entity that presented them. Such Trust Marks can appear for various reasons, such as the Entity Configuration including Trust Marks associated with another federation, or Trust Marks intended for specific purposes or Entity audiences.

An Entity MAY choose, at its own discretion, to utilize Trust Marks presented to it that are not recognized within the federation, and where the accreditation authority is established by some out-of-band mechanism.

5.3.3. Trust Mark Examples

A non-normative example of a Trust Mark claim inside an Entity Configuration is:

```
{
  "iss": "https://rp.example.it/spid/",
  "sub": "https://rp.example.it/spid/",
  "iat": 1516239022,
  "exp": 1516298022,
  "trust_marks": [
    {
     "id": "https://www.spid.gov.it/certification/rp/",
     "trust_mark":
        "eyJraWQiOiJmdWtDdUtTS3hwWWJjN09lZUk3Ynlya3N5a0E1bDhPb2RFSXVyOH"
        "JoNFlBIiwidHlwIjoidHJ1c3QtbWFyaytqd3QiLCJhbGciOiJSUzI1NiJ9.eyJ"
        "pc3MiOiJodHRwczovL3d3dy5hZ2lkLmdvdi5pdCIsInN1YiI6Imh0dHBzOi8vc"
        "nAuZXhhbXBsZS5pdC9zcGlkIiwiaWF0IjoxNTc5NjIxMTYwLCJpZCI6Imh0dHB"
        "zOi8vd3d3LnNwaWQuZ292Lml0L2NlcnRpZmljYXRpb24vcnAiLCJsb2dvX3Vya"
        "SI6Imh0dHBzOi8vd3d3LmFnaWQuZ292Lml0L3RoZW1lcy9jdXN0b20vYWdpZC9"
        "sb2dvLnN2ZyIsInJlZiI6Imh0dHBzOi8vZG9jcy5pdGFsaWEuaXQvZG9jcy9zc"
        "GlkLWNpZS1vaWRjLWRvY3MvaXQvdmVyc2lvbmUtY29ycmVudGUvIn0.AGf5Y4M"
        "oJt22rznH4i7Wqpb2EF2LzE6BFEkTzY1dCBMCK-8P_vj4Boz7335pUF45XXr2j"
        "x5_waDRgDoS5vOO-wfc0NWb4Zb_T1RCwcryrzV0z3jJICePMPM_1hZnBZjTNQd"
        "4EsFNvKmUo_teR2yzAZjguR2Rid30O5PO8kJtGaXDmz-rWaHbmfLhlNGJnqcp9"
        "Lo1bhkU_4Cjpn2bdX7RN0JyfHVY5IJXwdxUMENxZd-VtA5QYiw7kPExT53XcJO"
        "89ebe_ik4D0dl-vINwYhrIz2RPnqgA1OdbK7jg0vm8Tb3aemRLG7oLntHwqLO-"
        "gGYr6evM2_SgqwA0lQ9mB9yhw"
    }
  ],
  "metadata": {
    "openid_relying_party": {
      "application_type": "web",
      "client_registration_types": ["automatic"],
      "client_name": "https://rp.example.it/spid/",
      "contacts": [
        "ops@rp.example.it"
      ],

       ... follows other claims ...

   ... follows other claims ...
```

An example of a decoded Trust Mark issued to an RP, attesting the conformance to a national public service profile:

```
{
  "id":"https://federation.id/openid_relying_party/public/",
  "iss": "https://trust-anchor.gov.id",
  "sub": "https://rp.cie.id",
  "iat": 1579621160,
  "organization_name": "Organization name",
  "policy_uri": "https://rp.cie.id/privacy_policy",
  "tos_uri": "https://rp.cie.id/info_policy",
  "service_documentation": "https://rp.cie.id/api/v1/get/services",
  "ref": "https://rp.cie.id/documentation/manuale_operativo.pdf"
}
```

An example of a decoded Trust Mark issued to an RP, attesting its conformance to the rules for data management of underage users:

```
{
  "id":"https://federation.id/openid_relying_party/private/under-age",
  "iss": "https://trust-anchor.gov.id",
  "sub": "https://rp.cie.id",
  "iat": 1579621160,
  "organization_name": "Organization name",
  "policy_uri": "https://rp.cie.id/privacy_policy",
  "tos_uri": "https://rp.cie.id/info_policy"
}
```

An example of a Trust Mark attesting a stipulation of an agreement between an RP and an Attribute Authority:

```
{
  "id": "https://deleghedigitali.gov.it/openid_relying_party/sgd/",
  "iss": "https://deleghedigitali.gov.it",
  "sub": "https://rp.cie.id",
  "iat": 1579621160,
  "logo_uri": "https://deleghedigitali.gov.it/sgd-cmyk-150dpi-90mm.svg",
  "organization_type": "public",
  "id_code": "123456",
  "email": "info@rp.cie.id",
  "organization_name#it": "Nome dell'organizazzione",
  "policy_uri#it": "https://rp.cie.id/privacy_policy",
  "tos_uri#it": "https://rp.cie.id/info_policy",
  "service_documentation": "https://rp.cie.id/api/v1/get/services",
  "ref": "https://deleghedigitali.gov.it/documentation/manuale_operativo.pdf"
}
```

An example of a Trust Mark asserting conformance to a security profile:

```
{
  "iss": "https://secusign.org",
  "sub": "https://example.com/op",
  "iat": 1579621160,
  "id": "https://secusign.org/level/A",
  "logo_uri": "https://secusign.org/static/levels/
    certification-level-A-150dpi-90mm.svg",
  "ref": "https://secusign.org/conformances/"
}
```

An example of a decoded self-signed Trust Mark:

```
{
  "iss": "https://example.com/op",
  "sub": "https://example.com/op",
  "iat": 1579621160,
  "id": "https://openid.net/certification/op",
  "logo_uri": "http://openid.net/wordpress-content/uploads/2016/
    05/oid-l-certification-mark-l-cmyk-150dpi-90mm.svg",
  "ref": "https://openid.net/wordpress-content/uploads/2015/
    09/RolandHedberg-pyoidc-0.7.7-Basic-26-Sept-2015.zip"
}
```

An example of a third-party accreditation authority:

```
{
  "iss": "https://swamid.se",
  "sub": "https://umu.se/op",
  "iat": 1577833200,
  "exp": 1609369200,
  "id": "https://refeds.org/wp-content/uploads/2016/01/Sirtfi-1.0.pdf"
}
```

6. Obtaining Federation Entity Configuration Information

The Entity Configuration of every participant SHOULD be exposed at a well-known endpoint. The configuration endpoint is found using the Well-Known URIs [RFC8615] specification, with the suffix `openid-federation`. The scheme, host, and port are taken directly from the Entity Identifier combined with the following path: `/.well-known/openid-federation`.

If the Entity Identifier contains a path, it is concatenated after `/.well-known/openid-federation` in the same manner that path components are concatenated to the well-known identifier in the OAuth 2.0 Authorization Server Metadata [RFC8414] specification. Of course, in real multi-tenant deployments, in which the Entity Identifier might be of the form `https://multi-tenant-service.example.com/my-tenant-identifier` the tenant is very likely to not have control over the path `https://multi-tenant-service.example.com/.well-known/openid-federation/my-tenant-identifier` whereas it is very likely to have control over the path `https://multi-tenant-service.example.com/my-tenant-identifier/.well-known/openid-federation`. Therefore, if using the configuration endpoint at the URL with the tenant path after the well-known part fails, it is RECOMMENDED that callers retry at the URL with the tenant path before the well-known part (even though this violates [RFC8615]).

Entities SHOULD make an Entity Configuration document available at the configuration endpoint. There is only one exception to this rule and that is for an RP that only does Explicit Registration. Since it posts the Entity Configuration to the OP during client registration, the OP has everything it needs from the RP.

6.1. Federation Entity Configuration Request

An Entity Configuration document MUST be queried using an HTTP GET request at the previously specified path. The requesting party would make the following request to the Entity `https://openid.sunet.se` to obtain its configuration information:

```
GET /.well-known/openid-federation HTTP/1.1
Host: openid.sunet.se
```

6.2. Federation Entity Configuration Response

The response is an Entity Configuration, as described in Section 3.1. If the Entity is an Intermediate Entity or a Trust Anchor, the response MUST contain metadata for a federation Entity (`federation_entity`).

A successful response MUST use the HTTP status code 200 and the content type set to `application/entity-statement+jwt`, to make it clear that the response contains a signed Entity Statement. In case of an error, the response SHOULD be produced in accordance with what is defined in Section 7.7.

The following is a non-normative example response from an Intermediate Entity, before serialization and adding a signature:

```
200 OK
Content-Type: application/entity-statement+jwt

{
  "iss": "https://openid.sunet.se",
  "sub": "https://openid.sunet.se",
  "iat": 1516239022,
  "exp": 1516298022,
  "metadata": {
    "openid_provider": {
      "issuer": "https://openid.sunet.se",
      "signed_jwks_uri": "https://openid.sunet.se/jwks.jose",
      "authorization_endpoint":
        "https://openid.sunet.se/authorization",
      "client_registration_types_supported": [
        "automatic",
        "explicit"
      ],
      "grant_types_supported": [
        "authorization_code"
      ],
      "id_token_signing_alg_values_supported": [
        "ES256", "RS256"
      ],
      "logo_uri":
        "https://www.umu.se/img/umu-logo-left-neg-SE.svg",
      "op_policy_uri":
        "https://www.umu.se/en/website/legal-information/",
      "response_types_supported": [
        "code"
      ],
      "subject_types_supported": [
        "pairwise",
        "public"
      ],
      "token_endpoint": "https://openid.sunet.se/token",
      "federation_registration_endpoint":
        "https://op.umu.se/openid/fedreg",
      "token_endpoint_auth_methods_supported": [
        "private_key_jwt"
      ]

    }
  },
  "jwks": {
    "keys": [
      {
        "alg": "RS256",
        "e": "AQAB",
        "key_ops": [
          "verify"
        ],
        "kid": "key1",
        "kty": "RSA",
        "n": "pnXBOusEANuug6ewezb9J_...",
        "use": "sig"
      }
    ]
  },
  "authority_hints": [
    "https://edugain.org/federation"
```

```
    ]
}
```

7. Federation Endpoints

The federation endpoints of an Entity can be found in the configuration response as described in Section 6 or by other means.

7.1. Fetching Entity Statements

All Entities that are expected to publish Entity Statements about other Entities MUST expose a Fetch endpoint.

Fetching Entity Statements is performed to collect Entity Statements one by one to gather Trust Chains.

To fetch an Entity Statement, an Entity needs to know the identifier of the Entity to ask (the issuer), the fetch endpoint of that Entity, and the identifier of the Entity, that is, the subject of the statement.

7.1.1. Fetch Entity Statements Request

The request MUST be an HTTP request using the GET method and the `https` scheme to a resolved federation endpoint with the following query string parameters, encoded in `application/x-www-form-urlencoded` format.

iss
: OPTIONAL. The Entity Identifier of the issuer from which the Entity Statement issued. Because of the normalization of the URL, multiple issuers MAY resolve to a shared fetch endpoint. This parameter makes it explicit exactly which issuer the Entity Statements must come from. Without this parameter, the issuer matches to the Entity at which the request was made.

sub
: OPTIONAL. The Entity Identifier of the subject for which the Entity Statement is issued. If this parameter is left out, it is considered to be the same as the issuer and would indicate a request for a self-signed statement.

The following is a non-normative example of an API request for an Entity Statement:

```
GET /federation_fetch_endpoint?
iss=https%3A%2F%2Fedugain.org%2Ffederation&
sub=https%3A%2F%2Fopenid%2Esunet%2Ese HTTP/1.1
Host: edugain.org
```

7.1.2. Fetch Entity Statements Response

A successful response MUST use the HTTP status code 200 and the content type set to `application/entity-statement+jwt`, to make it clear that the response contains a signed Entity Statement. If it is a negative response, it will be a JSON object and the content type MUST be set to `application/json`. See more about error responses in Section 7.7.

The following is a non-normative example of a response, before serialization and adding a signature:

```
200 OK
Content-Type: application/entity-statement+jwt

{
  "iss": "https://edugain.org/federation",
  "sub": "https://openid.sunet.se"
  "exp": 1568397247,
  "iat": 1568310847,
  "jwks": {
    "keys": [
      {
        "e": "AQAB",
        "kid": "dEEtRjlzY3djcENuT01wOGxrZlkxb3RIQVJlMTY0...",
        "kty": "RSA",
        "n": "x97YKqc9Cs-DNtFrQ7_vhXoH9bwkDWW6En2jJ044yH..."
      }
    ]
  },
  "metadata":{
    "federation_entity": {
        "organization_name":"SUNET"
    }
  }
  "metadata_policy": {
    "openid_provider": {
      "subject_types_supported": {
        "value": [
          "pairwise"
        ]
      },
      "token_endpoint_auth_methods_supported": {
        "default": [
          "private_key_jwt"
        ],
        "subset_of": [
          "private_key_jwt",
          "client_secret_jwt"
        ],
        "superset_of": [
          "private_key_jwt"
        ]
      }
    }
  }
}
```

7.2. Resolve Entity Statement

An Entity MAY use the resolve endpoint to fetch resolved metadata and Trust Marks for an Entity as seen/trusted by the resolver. The resolver is supposed to fetch the subject's Entity Configuration, collect a Trust Chain that starts with the subject's Entity Statement and ends with the specified Trust Anchor, verify the Trust Chain, and then apply all the policies present in the Trust Chain to the Entity Statements metadata. The resolver is also expected to verify that the present Trust Marks are active. If it finds Trust Marks that are not active, then those should be left out of the response set.

7.2.1. Resolve Request

The request MUST be an HTTP request using the GET method and the `https` scheme to a resolve endpoint with the following query string parameters, encoded in `application/x-www-form-urlencoded` format.

sub
: REQUIRED. The Entity Identifier of the Entity whose resolved data is requested.

anchor
: REQUIRED. The Trust Anchor that the remote peer MUST use when resolving the metadata. The value is an Entity identifier.

type
: OPTIONAL. A specific metadata type to resolve. In this document, we use the metadata types listed in Section 4. If no value is given, then all metadata types are expected to be returned.

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

```
GET /resolve?
sub=https%3A%2F%2Fop.example.it%2Fspid&
type=openid_provider&
anchor=https%3A%2F%2Fswamid.se HTTP/1.1
Host: openid.sunet.se
```

7.2.2. Resolve Response

A successful response MUST use the HTTP status code 200 and the content type set to `application/resolve-response+jwt`, containing resolved metadata and verified Trust Marks.

The response is a signed JWT, explicitly typed by setting the `typ` header parameter to `resolve-response+jwt`. This is done to prevent cross-JWT confusion (see [RFC8725], section 3.11).

The resolve response JWT MAY also contain the sequence of the Entity Statements that compose the Trust Chain, sorted as shown in Section 3.2. The key used by the resolver to sign the response MUST be one of its Federation Entity Keys. The response SHOULD contain the aud claim only if the requesting client is authenticated.

These are the resolve response Claims:

iss
: REQUIRED. String. URL corresponding to the federation's Entity Identifier of the issuer of the resolve response.

sub
: REQUIRED. String. URL corresponding to the federation's Entity Identifier of the subject for which the resolve response refers to.

iat
: REQUIRED. Number. When this resolution was issued. Expressed as Seconds Since the Epoch, as defined in [RFC7519].

exp
: REQUIRED. Number. When this resolution is not valid anymore. Expressed as Seconds Since the Epoch, as defined in [RFC7519]. It MUST correspond to the `exp` value of the Trust Chain from which the resolve response was derived.

metadata
: REQUIRED. JSON object containing the resolved subject metadata, according to the requested `type` and expressed in the `metadata` format defined in Section 3.1.

trust_marks
: OPTIONAL. A JSON array of objects, each representing a Trust Mark, as defined in Section 3.1.

trust_chain
: OPTIONAL. Array containing the sequence of the statements that compose the Trust Chain starting with the subject and ending with the selected Trust Anchor, sorted as shown in Section 3.2.

Other claims MAY be used in conjunction with the claims outlined above. The claim naming recommendations outlined in Section 5.1.2 of OpenID Connect Core 1.0 [OpenID.Core] apply.

If the response is negative, the response SHOULD be produced in accordance with what is defined in Section 7.7.

The following is a non-normative example of a response, before serialization and adding a signature:


```
{
  "iss": "https://resolver.spid.gov.it/",
  "sub": "https://op.example.it/spid/",
  "iat": 1516239022,
  "exp": 1516298022,
  "metadata": {
    "openid_provider": {
      "contacts": ["legal@example.it", "technical@example.it"],
      "logo_uri":
        "https://op.example.it/static/img/op-logo.svg",
      "op_policy_uri":
        "https://op.example.it/en/about-the-website/legal-information/",
      "federation_registration_endpoint":"https://op.example.it/spid/fedreg/",
      "authorization_endpoint":
        "https://op.example.it/spid/authorization/",
      "token_endpoint": "https://op.example.it/spid/token/",
      "response_types_supported": [
        "code",
        "code id_token",
        "token"
      ],
      "grant_types_supported": [
        "authorization_code",
        "implicit",
        "urn:ietf:params:oauth:grant-type:jwt-bearer"
      ],
      "subject_types_supported": ["pairwise"],
      "id_token_signing_alg_values_supported": ["RS256"],
      "issuer": "https://op.example.it/spid/",
      "jwks": {
        "keys": [
          {
            "kty": "RSA",
            "use": "sig",
            "n": "1Ta-sE ...",
            "e": "AQAB",
            "kid": "FANFS3YnC9tjiCaivhWLVUJ3AxwGGz_98uRFaqMEEs"
          }
        ]
      }
    }
  },
  "trust_marks": [
    {"id": "https://www.spid.gov.it/certification/op/",
     "trust_mark":
       "eyJhbGciOiJSUzI1NiIsImtpZCI6ImRGRTFjMFF4UzBFdFFrWmxNRXR3ZWxOQl"
       "eyJhbGciOiJSUzI1NiIsImtpZCI6Ijh4c3VLV2lWZndTZ0hvZjFUZTRPVWRjeT"
       "RxN2RKcktmRlJsTzV4aEhJYTAifQ.eyJpc3MiOiJodHRwczovL3d3dy5hZ2lkL"
       "mdvdi5pdCIsInN1YiI6Imh0dHBzOi8vb3AuZXhhbXBsZS5pdC9zcGlkLyIsIml"
       "hdCI6MTU3OTYyMTE2MCwiaWQiOiJodHRwczovL3d3dy5zcGlkLmdvdi5pdC9jZ"
       "XJ0aWZpY2F0aW9uL29wLyIsImxvZ29fdXJpIjoiaHR0cHM6Ly93d3cuYWdpZC5"
       "nb3YuaXQvdGhlbWVzL2N1c3RvbS9hZ2lkL2xvZ28uc3ZnIiwicmVmIjoiaHR0c"
       "HM6Ly9kb2NzLml0YWxpYS5pdC9pdGFsaWEvc3BpZC9zcGlkLXJlZ29sZS10ZWN"
       "uaWNoZS1vaWRjL2l0L3N0YWJpbGUvaW5kZXguaHRtbCJ9"
    }
  ],
  "trust_chain" : [
    "eyJhbGciOiJSUzI1NiIsImtpZCI6Ims1NEhRdERpYnlHY3M5WldWTWZ2aUhm ...",
    "eyJhbGciOiJSUzI1NiIsImtpZCI6IkJYdmZybG5oQU11SFIwN2FqVW1BY0JS ...",
    "eyJhbGciOiJSUzI1NiIsImtpZCI6IkJYdmZybG5oQU11SFIwN2FqVW1BY0JS ..."
```

```
    ]
}
```

7.2.3. Considerations

The basic assumption of this specification is that an Entity should have direct trust in no one except the Trust Anchor and its own capabilities. However, Entities may establish a kind of transistive trust in other Entities. For example, the Trust Anchor states who its Subordinates are and Entities may choose to trust these. If a party uses the resolve service of another Entity to obtain federation data, it is trusting the resolver to perform the validation of the cryptographically protected metadata correctly and to provide it with authentic results.

7.3. Subordinate Listings

The listing endpoint is exposed by Federation Entities acting as Trust Anchor, Intermediates or Trust Mark issuers.

The endpoint lists the Subordinates, about which the Trust Anchor or Intermediate issues Entity Statements.

As Trust Mark issuer, the endpoint MAY list the Subordinates for which Trust Marks have been issued and are still valid, if the issuer exposing this endpoint supports Trust Mark filtering, as defined below.

In both cases, the list contained in the result MAY be a very large list.

7.3.1. Subordinate Listing Request

The request MUST be an HTTP request using the GET method and the `https` scheme to a list endpoint with the following query parameters, encoded in `application/x-www-form-urlencoded` format.

entity_type
: OPTIONAL. If the responder knows the Entity Type of all its Subordinates, the result MUST be filtered accordingly. If the responder does not support this feature it MUST use the HTTP status code 400 and set the content type to `application/json`, with the error code set to `unsupported_parameter`.

trust_marked
: OPTIONAL. Boolean. If the parameter `trust_marked` is present and set to `true`, the result contains only the Subordinates for which at least one Trust Mark have been issued and is still valid. If the responder does not support this feature it MUST use the HTTP status code 400 and set the content type to `application/json`, with the error code set to `unsupported_parameter`.

trust_mark_id
: OPTIONAL. If the responder has issued Trust Marks with the `trust_mark_id`, the list obtained in the response MUST be filtered with the Subordinates for which that Trust Mark identifier has been issued and is still valid. If the responder does not support this feature it MUST use the HTTP status code 400 and set the content type to `application/json`, with the error code set to `unsupported_parameter`.

The following is a non-normative example of an API request for a list of Subordinates:

```
GET /list HTTP/1.1
Host: openid.sunet.se
```

7.3.2. Subordinate Listing Response

A successful response MUST use the HTTP status code 200 and the content type set to `application/json`, containing a JSON array with the known Entity Identifiers.

If the response is negative, the response should be produced in accordance with what is defined in Section 7.7.

The following is a non-normative example of a response, containing the Subordinate Entities:

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

[
  "https://ntnu.andreas.labs.uninett.no/",
  "https://blackboard.ntnu.no/openid/callback",
  "https://serviceprovider.andreas.labs.uninett.no/application17"
]
```

7.4. Trust Mark Status

This enables an Entity to check whether a Trust Mark is still active or not. The query MUST be sent to the Trust Mark issuer.

7.4.1. Status Request

The request MUST be an HTTP request using the POST method and the `https` scheme to a status endpoint with the following parameters, encoded in `application/x-www-form-urlencoded` format.

sub
: OPTIONAL. The ID of the Entity to which the Trust Mark was issued.

id
: OPTIONAL. Identifier of the Trust Mark.

iat
: OPTIONAL. When the Trust Mark was issued. If `iat` is not specified and the Trust Mark Issuer has issued several Trust Marks with the `id` specified in the request to the Entity identified by `sub`, the most recent one is assumed.

trust_mark
: OPTIONAL. The whole Trust Mark.

If `trust_mark` is used, then `sub` and `id` are not needed. If `trust_mark` is not used, then `sub` and `id` are required.

The following is a non-normative example of a request using `sub` and `id`:

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

sub=https%3A%2F%2Fopenid.sunet.se%2FRP
&id=https%3A%2F%2Frefeds.org%2Fsirtfi
```

7.4.2. Status Response

A successful response MUST use the HTTP status code 200 and the content type set to `application/json`, to make it clear that the response contains a JSON object containing the claim below.

If the response is negative, the response SHOULD be produced in accordance with what is defined in Section 7.7.

active
: REQUIRED. Boolean. Whether the Trust Mark is active or not.

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

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

{
  "active": true
}
```

7.5. Trust Marked Entities Listing

The Trust Marked Entities listing endpoint is exposed by Trust Mark issuers and lists all the Entities for which Trust Marks have been issued and are still valid.

7.5.1. Trust Marked Entities Listing Request

The request MUST be an HTTP request using the GET method and the `https` scheme to a list endpoint with the following query parameters, encoded in `application/x-www-form-urlencoded` format.

trust_mark_id
: REQUIRED. If the responder has issued Trust Marks with the `trust_mark_id`, the list obtained in the response MUST be filtered with the Entities for which that Trust Mark identifier has been issued and is still valid.

The following is a non-normative example of an API request for a list of Trust Marked Entities:

```
GET /trust_marked_list?
trust_mark_id=https%3A%2F%2Ffederation.example.org%2Fopenid_relying_party%2Fprivate%2Fu
nder-age HTTP/1.1
Host: trust-mark-issuer.example.org
```

7.5.2. Trust Marked Entities Listing Response

A successful response MUST use the HTTP status code 200 and the content type set to `application/json`, containing a JSON array with Entity Identifiers.

If the response is negative, the response should be produced in accordance with what is defined in Section 7.7.

The following is a non-normative example of a response, containing the Trust Marked Entities:

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

[
  "https://blackboard.ntnu.no/openid/rp",
  "https://that-rp.example.org"
]
```

7.6. Federation Historical Keys endpoint

Each Federation Entity MAY publish its previously used federation public keys at the endpoint `/.well-known/openid-federation-historical-jwks` . The purpose of this endpoint is to provide the list of keys previously used by the Federation Entity in order to provide non-repudiation of statements signed by it after key rotation. This endpoint also discloses the reason for the retraction of the keys, whether they were expired or revoked, including the reason for the revocation.

Note that an expired key can be later additionally marked as revoked, to indicate a key compromise event discovered after the expiration of the key.

The publishing of the historical keys guarantees that Trust Chains will remain verifiable and usable as inputs to trust decisions after the key expiration, unless the key becomes revoked for a security reason.

7.6.1. Federation Historical Keys Request

The request MUST be an HTTP request using the GET method and the `https` scheme to the well known endpoint `openid-federation-historical-jwks`.

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

```
GET /.well-known/openid-federation-historical-jwks HTTP/1.1
Host: trust-anchor.example.com
```

7.6.2. Federation Historical Keys Response

A successful response MUST use the HTTP status code 200 and the content type set to `application/jwk-set+jwt`. The response is a signed JWT with the JWK Set [RFC7517] as its payload. The JWT is explicitly typed by setting the `typ` header parameter to `jwk-set+jwt`. This is done to prevent cross-JWT confusion (see [RFC8725], section 3.11). The response JWT contains the following claims:

iss
: REQUIRED. String. URL corresponding to the Federation Entity Identifier.

iat
: REQUIRED. Number. When the signed JWT was issued, using the time format defined for the `iat` claim in [RFC7519].

keys
: REQUIRED. An array of JSON objects representing the public Federation Entity signing keys.

  Every JWK in the `keys` claim includes the following claims:

  kid
  : REQUIRED. Parameter is used to match a specific key. It is RECOMMENDED that the Key ID be the JWK Thumbprint [RFC7638] using the SHA-256 hash function of the key.

  iat
  : REQUIRED. Time at which the key was issued, using the time format defined for the `iat` claim in [RFC7519].

exp
: REQUIRED. Expiration time, using the time format defined for the exp claim in [RFC7519], on or after which the key MUST NOT be considered as valid.

revoked
: OPTIONAL. JSON object that contains the properties of the revocation, defined below.

  revoked_at
  : REQUIRED. Time at which the key was revoked or must be considered revoked, using the time format defined for the iat claim in [RFC7519].

  reason
  : REQUIRED. Human readable value that identifies the reason for the key revocation, as defined in Section 5.3.1 of [RFC5280].

  reason_code
  : REQUIRED. Integer that identifies the reason for the key revocation, as defined in Section 5.3.1 of [RFC5280].

The following is a non-normative example of a response, before serialization and adding a signature:

```
HTTP/1.1 200 OK
Content-Type: application/jwk-set+jwt

{
    "iss": "https://trust-anchor.federation.example.com",
    "iat": 123972394272,
    "keys":
        [
            {
                "kty":"RSA",
                "n":"5s4qi …",
                "e":"AQAB",
                "kid":"2HnoFS3YnC9tjiCaivhWLVUJ3AxwGGz_98uRFaqMEEs",
                "iat": 123972394872,
                "exp": 123974395972
            },
            {
                "kty":"RSA",
                "n":"ng5jr …",
                "e":"AQAB",
                "kid":"8KnoFS3YnC9tjiCaivhWLVUJ3AxwGGz_98uRFaqMJJr",
                "iat": 123972394872,
                "exp": 123974394972
                "revoked": {
                  "revoked_at": 123972495172,
                  "reason": "keyCompromise",
                  "reason_code": 1
                }
            }
        ]
}
```

7.6.3. Rationale for the Federation Historical Keys endpoint

The Federation Historical Keys endpoint solves the problem of verifying historical Trust Chains when the Federation Entity Keys are changed, due to expiry or revocation.

The Federation Historical Keys endpoint publishes the list of public keys used in the past by a Federation Entity. The given public keys are needed by all the Entities to verify the Trust Chains evaluated in the past with the Federation Entity Keys not published anymore in the Federation Entity's Entity Configuration.

Federation Historical Keys endpoint response contains a signed JWT that attests all the expired and revoked Federation Entity Keys.

On the basis of the attributes contained in the Entity Statements that form a Trust Chain, it MAY be also possible to verify the non-federation public keys used in the past by a Leaf for the signature operations related to the OpenID Connect requests and responses. For example an Entity Statement issued for a Leaf MAY also include the `jwks` claim, respectively in the claims `metadata` or `metadata_policy`.

A simple example: In the following Trust Chain the Federation Intermediate attests the Leaf's OpenID Connect `jwks` in the Entity Statement issued for the Leaf. The result is a Trust Chain that contains the Leaf's OIDC Core jwks, needed to verify historical signature on Request Objects, ID Tokens and any other signed JWT issued by the Leaf, if it is an RP or an OP.

1. an Entity Configuration about the RP published by the RP,
2. an Entity Statement about the RP published by Organization A, with the claim `jwks` contained in `metadata` or `metadata_policy` attesting the Leaf's OpenID Core `jwks`.
3. an Entity Statement about Organization A published by Federation F.

7.7. Generic Error Response

If the request was malformed, or some error occurred during the processing of the request, the response body SHOULD be a JSON object with the content type set to `application/json`.

In compliance with [RFC6749], the following standardized error format SHOULD be used:

error
: REQUIRED. One of the following error codes SHOULD be used.

  invalid_request
  : The request is incomplete or does not comply with current specifications. The HTTP response status code SHOULD be 400 (Bad Request).

  invalid_client
  : The Client can not be authorized or is not a valid participant of the federation. The HTTP response status code SHOULD be 401 (Unauthorized).

  not_found
  : The requested Entity Identifier is "not found". The HTTP response status code SHOULD be 404 (Not Found).

  server_error
  : The server encountered an unexpected condition that prevented it from fulfilling the request. The HTTP response status code SHOULD be one in the 5xx range, like 500 (Internal Server Error).

  temporarily_unavailable
  : The server hosting the federation endpoint is currently unable to handle the request due to a temporary overloading or maintenance. The HTTP response status code SHOULD be 503 (Service Unavailable).

  unsupported_parameter
  : The server does not support the requested parameter.

error_description
: REQUIRED. A human-readable short text describing the error.

The following is a non-normative example of an error response:

```
400 Bad request
Content-Type: application/json

{
  "error": "invalid_request",
  "error_description":
    "Required request parameter [sub] was missing."
}
```

8. Resolving Trust Chain and Metadata

An Entity (e.g., the consumer) that wants to establish trust with a remote peer MUST have the remote peer's Entity Identifier and a list of Entity Identifiers of Trust Anchors together with the public version of their signing keys. The consumer will first have to fetch sufficient Entity Statements to establish at least one chain of trust from the remote peer to one or more of the configured Trust Anchors. After that, the Entity MUST validate the Trust Chains independently, and if there are multiple valid Trust Chains and if the application demands it, choose one.

8.1. Fetching Entity Statements to Establish a Trust Chain

Depending on the circumstances, the consumer MAY be handed the remote peer's Entity Configuration, or it may have to fetch it by itself. If it needs to fetch it, it will use the process described in Section 6 based on the Entity Identifier of the remote peer.

The next step is to iterate through the list of Intermediates listed in `authority_hints`, ignoring the authority hints that end in an unknown Trust Anchor, requesting an Entity Configuration from each of the Intermediates. If the received Entity Configuration contains an authority hint, this process is repeated.

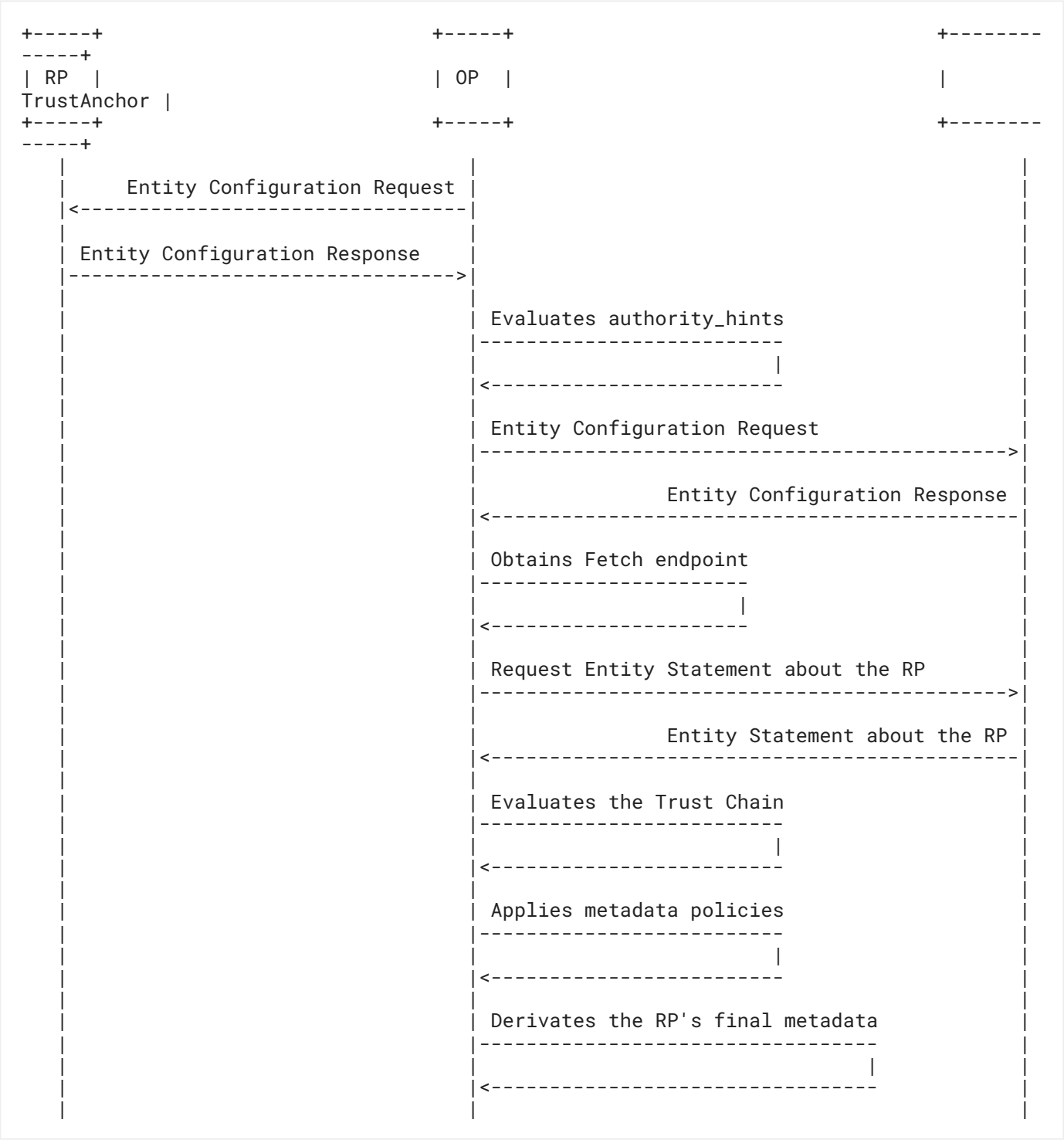
With the list of all Intermediates and the Trust Anchor, the respective federation API (see Section 7) is used to fetch Entity Statements about the Intermediates and the remote peer.

Note: The consumer SHOULD NOT attempt to fetch Entity Statements it already has fetched during this process (loop prevention).

A successful operation will return one or more lists of Entity Statements. Each of the lists terminating in a self-signed Entity Statement is issued by a Trust Anchor.

If there is no path from the remote peer to at least one of the trusted Trust Anchors, then the list will be empty and there is no way of establishing trust in the remote peer's information. How the consumer deals with this is out of scope for this specification.

Resolving Trust Chain and Metadata: the perspective of an OP.



8.2. Validating a Trust Chain

As described in Section 3.2, a Trust Chain consists of an ordered list of Entity Statements. So whichever way the consumer has acquired the set of Entity Statements, it MUST now verify that it is a proper Trust Chain using the rules laid out in that section.

To validate the chain, the following MUST be done:

- For each Entity Statement ES[j] j=i,..,0:
  - Verify that the statement contains all the required claims.
  - Verify that `iat` has a value in the past.
  - Verify that `exp` has a value that is in the future.
- For j=0 verify that `iss` == `sub`.
- For j=0,...,i-1: Verify that ES[j]['iss'] == ES[j+1]['sub']
- For j=0,...,i-1: Verify the signature of ES[j] using a public key carried in ES[j+1]['jwks'].
- For j == 0 verify the signature of ES[0] using a public key carried in ES[0]['jwks'].
- For j == i: verify that a) the issuer matches the configured identifier of a Trust Anchor and b) its signature is valid with the likewise configured public key of said Trust Anchor.

Verifying the signature is a much more expensive operation than verifying the correctness of the statement and the timestamps. An implementer MAY therefore choose not to verify the signature until all the other checks have been done.

Consumers MAY cache Entity Statements or signature verification results for a given time until they expire, per Section 8.4.

Note that the second bullet point means that, at each step in the Trust Chain resolution, it MUST be verified that the signing JWK is also present in the `jwks` statement claim issued by the superior.

8.3. Choosing One of the Valid Trust Chains

If multiple valid Trust Chains are found, the consumer will need to decide on which one to use.

One simple rule would be to prefer a shorter chain over a longer one.

Consumers MAY follow other rules according to local policy.

8.4. Calculating the Expiration Time of a Trust Chain

Each Entity Statement in a Trust Chain is signed and MUST have an expiration time (`exp`) set. The expiration time of the whole Trust Chain is set to the minimum value of (`exp`) within the chain.

9. Updating Metadata, Key Rollover, and Revocation

This specification allows for a smooth process of updating metadata and public keys.

As described above in Section 8.4, each Trust Chain has an expiration time. A consumer of metadata using this specification MUST support refreshing a Trust Chain when it expires. How often a consumer SHOULD re-evaluate the Trust Chain depends on how quickly the consumer wants to find out that something has changed in the Trust Chain.

9.1. Protocol Key Rollover

If a Leaf Entity publishes its public keys in the metadata part using `jwks`, setting an expiration time on the self-signed Entity Statement can be used to control how often the receiving Entity is fetching an updated version of the public key.

9.2. Key Rollover for a Trust Anchor

A Trust Anchor MUST publish an Entity Configuration about itself. The Trust Anchor SHOULD set a reasonable expiration time on that Statement, such that the consumers will re-fetch the Entity Configuration at reasonable intervals. If the Trust Anchor wants to roll over its signing keys, it would have to:

1. Add the new keys to the `jwks` representing the Trust Anchor's signing keys.
2. Keep signing the Entity Configuration and the Entity Statements using the old keys for a long enough time period to allow all Subordinates to have gotten access to the new keys.
3. Switch to signing with the new keys.
4. After a reasonable time period, remove the old keys. What is regarded as a reasonable time is dependent on the security profile and risk assessment of the Trust Anchor.

9.3. Redundant Retrieval of Trust Anchor Keys

It is RECOMMENDED that Federation Operators provide a means of retrieving the public keys for the Trust Anchors it administers that is independent of those Trust Anchors' Entity Configurations. This is intended to provide redundancy in the eventuality of the compromise of the Web PKI infrastructure underlying retrieval of public keys from Entity Configurations.

The keys retrieved via the independent mechanism specified by the Federation Operator SHOULD be compared to those retrieved via the Trust Anchor's Entity Configuration. If they do not match, both SHOULD be retrieved again. If they still do not match, it is indicative of a security or configuration problem. The appropriate remediation steps in that eventuality SHOULD be specified by the Federation Operator.

9.4. Revocation

Since the consumers are expected to check the Trust Chain at regular, reasonably frequent times, this specification does not specify a standard revocation process. Specific federations MAY make a different choice and will then have to add such a process.

10. OpenID Connect Communication

This section describes how the trust framework in this specification is used to establish trust between an RP and an OP that have no explicit configuration or registration in advance.

There are two alternative approaches to establish trust between an RP and an OP, which we call Automatic and Explicit Registration. Members of a federation or a community SHOULD agree upon which one to use. While implementations should support both methods, deployments MAY choose to disable the use of one of them.

Independent of whether the RP uses Automatic or Explicit Registration, the way that the RP learns about the OP is the same. It will use the procedure that is described in Section 8.

10.1. Automatic Registration

Automatic Registration enables an RP to make Authentication Requests without a prior registration step with the OP. The OP resolves the RP's Entity Configuration from the Client ID in the Authentication Request, following the process defined in Section 8.

Automatic Registration has the following characteristics:

- In all interactions with the OP, the RP employs its Entity Identifier as the Client ID. The OP retrieves the RP's Entity Configuration from a URL derived from the Entity Identifier, as described in Section 6.
- Since there is no registration step prior to the Authentication Request, the RP MUST NOT be supplied with a Client Secret. Instead, the Automatic Registration model requires the RP to use asymmetric cryptography to authenticate its requests.
- The OP MUST publish that it supports a request authentication method using the metadata claim `request_authentication_methods_supported`.

10.1.1. Authentication Request

The Authentication Request is performed by passing a Request Object by value as described in Section 6.1 in OpenID Connect Core 1.0 [OpenID.Core] or using pushed authorization as described in Pushed Authorization Requests [RFC9126].

10.1.1.1. Using a Request Object

In the case where a Request Object is used, the value of the `request` parameter is a JWT whose Claims are the request parameters specified in Section 3.1.2 in OpenID Connect Core 1.0 [OpenID.Core]. The JWT MUST be signed and MAY be encrypted. The following parameters apply to the JWT:

aud
: REQUIRED. The Audience (aud) value MUST be or include the OP's Issuer Identifier URL.

iss
: REQUIRED. The claim `iss` MUST contain the Client Identifier.

sub
: MUST NOT be present. This, together with the value of `aud`, SHOULD make reuse of the statement for `private_key_jwt` client authentication not feasible.

jti
: REQUIRED. JWT ID. A unique identifier for the JWT, which can be used to prevent reuse of the token. These tokens MUST only be used once, unless conditions for reuse were negotiated between the parties; any such negotiation is beyond the scope of this specification.

exp
: REQUIRED. Expiration time on or after which the JWT MUST NOT be accepted for processing.

iat
: OPTIONAL. Time at which the JWT was issued.

trust_chain
: OPTIONAL. Array containing the sequence of the statements that compose the Trust Chain between the RP that makes the request and the selected Trust Anchor, sorted as shown in Section 3.2. When the RP and the OP are part of the same federation the RP MUST select the Trust Anchor that it has in common with the OP, otherwise the RP is free to select the Trust Anchor it deems most significant.

10.1.1.1.1. Authorization Request with a Trust Chain

When the `trust_chain` JWS [RFC7515] header parameter is used in conjunction with the `trust_chain` request parameter, both contents MUST be identical, otherwise the request MUST be rejected.

When the `trust_chain` JWS [RFC7515] header parameter or the `trust_chain` request parameter is used in the Authorization Request Object, the Relying Party informs the OP about the sequence of the statements that proves the trust relationship between it and the selected Trust Anchor.

Due to the large size of a Trust Chain, it could be necessary to use the HTTP POST method, `request_uri` or Pushed Authorization Request [RFC9126] for the request.

The following is a non-normative example of the Claims in a Request Object before base64url encoding and signing:

```
{
  "alg": "RS256",
  "kid": "that-kid-which-points-to-a-jwk-contained-in-the-trust-chain",
  "trust_chain" : [
    "eyJhbGciOiJSUzI1NiIsImtpZCI6Ims1NEhRdERpYnlHY3M5WldWTWZ2aUhm ...",
    "eyJhbGciOiJSUzI1NiIsImtpZCI6IkJYdmZybG5oQU11SFIwN2FqVW1BY0JS ...",
    "eyJhbGciOiJSUzI1NiIsImtpZCI6IkJYdmZybG5oQU11SFIwN2FqVW1BY0JS ..."
  ]
}
.
{
  "aud": "https://op.example.org",
  "client_id": "https://rp.example.com",
  "exp": 1589699162,
  "iat": 1589699102,
  "iss": "https://rp.example.com",
  "jti": "4d3ec0f81f134ee9a97e0449be6d32be",
  "nonce": "4LX0mFMxdBjkGmtx7a8WIOnB",
  "redirect_uri": "https://rp.example.com/authz_cb",
  "response_type": "code",
  "scope": "openid profile email address phone",
  "state": "YmX8PM9I7WbNoMnnieKKBiptVW0sP2OZ"
}
```

The following is a non-normative example of an Authentication Request using the request parameter (with line wraps within values for display purposes only):

```
https://server.example.com/authorize?
    redirect_uri=https%3A%2F%2Frp.example.com%2Fauthz_cb
    &scope=openid+profile+email+address+phone
    &response_type=code
    &client_id=https%3A%2F%2Frp.example.com
    &request=eyJ0cnVzdF9jaGFpbiI6WyJleUpoYkdjaU9pSlNVekkxTmlJc0ltdHBaQ0k2SW1z
      MU5FaFJkRVJwWW5sSFkzTTVXbGRXVFdaMmFVaG0gLi4uIiwiZXlKaGJHY2lPaUpT
      VXpJMU5pSXNJbXRwWkNJNklrSllkbVp5Ykc1b1FVMTFTRkl3TjJGcVZXMUJZMEpT
      IC4uLiIsImV5SmhiR2NpT2lKU1V6STFOaUlzSW10cFpDSTZJa0pZZG1aeWJHNW9R
      VTExU0ZJd04yRnFWVzFCWTBKUyAuLi4iXSwiYWxnIjoiUlMyNTYiLCJraWQiOiI2
      X2VGcGNoNXpTYm1QT3hMdGRGLXlrM1dqVFJvUGpBMll6UTd5YnJmV2dvIn0.eyJh
      dWQiOiJodHRwczovL29wLmV4YW1wbGUub3JnIiwiY2xpZW50X2lkIjoiaHR0cHM6
      Ly9ycC5leGFtcGxlLmNvbSIsImV4cCI6MTU4OTY5OTE2MiwiaWF0IjoxNTg5Njk5
      MTAyLCJpc3MiOiJodHRwczovL3JwLmV4YW1wbGUuY29tIiwianRpIjoiNGQzZWMw
      ZjgxZjEzNGVlOWE5N2UwNDQ5YmU2ZDMyYmUiLCJub25jZSI6IjRMWDBtRk14ZEJq
      a0dtdHg3YThXSU9uQiIsInJlZGlyZWN0X3VyaSI6Imh0dHBzOi8vcnAuZXhhbXBs
      ZS5jb20vYXV0aHpfY2IiLCJyZXNwb25zZV90eXBlIjoiY29kZSIsInNjb3BlIjoi
      b3BlbmlkIHByb2ZpbGUgZW1haWwgYWRkcmVzcyBwaG9uZSIsInN0YXRlIjoiWW1Y
      OFBNOUk3V2JOb01ubmllS0tCaXB0Vlcwc1AyT1oiLCJzdWIiOiJodHRwczovL3Jw
      LmV4YW1wbGUuY29tIn0.Gjo1NSYAx5PIllnUJhRCzZT-ezqyofU95pnGsgzclTfj
      cYCwSef_g2cniIWX4-35cAYR-NcAGEzaDIvQgzQ90O_24HlCtZ6yvUlb65uhZGGt
      O1TvsI7bl-92yrYCKD8fmaWH73R7qXZ8uLNspRy0L4emGXdUrFJ8RozE5asEdY_L
      _1orhot6uwWWrYE5cSyxJqCk_G1ackqKRmOlHB3EX3pNmVZodz6DQyONLeBqiMId
      xpvVALEkmpAQavEwrfpA-s4K3QIJrKAbEVQ1AfyQR0cGDd7fF4bju-wigYhBura0
      Pv4PrEFSNYG22b5ZPoubTPoFe-7W5Ypec_Io1aXNDA
```

10.1.1.1.2. Processing the Authentication Request

When the OP receives an incoming Authentication Request, the OP supports OpenID Connect Federation, the incoming Client ID is a valid URL, and the OP does not have the Client ID registered as a known client, then the OP SHOULD resolve the Trust Chains related to the requestor.

An RP MAY present to the OP a Trust Chain related to itself, using the trust_chain (Section 10.1.1.1) request parameter or the trust_chain (Section 3.2.1) JWS [RFC7515] header parameter in the Request Object. If the OP doesn't have a valid registration for the RP or its registration has expired, the OP MAY use the received Trust Chain as a hint for which path to take from the Leaf Entity to the Trust Anchor. The OP MAY evaluate the statements in the `trust_chain` to make its Federation Entity Discovery procedure more efficient, especially if the RP shows more than a single authority hint in its Entity Configuration. If the OP already has a valid registration for the RP, it MAY use the received Trust Chain to update the RP's registration. Whenever the OP uses a Trust Chain submitted by an RP, the OP MUST fully verify it, with every statement contained in it. A Trust Chain may be relied upon by the OP because it has validated all of its statements. This is true whether these statements are retrieved from their URLs or whether they are provided via the `trust_chain` request parameter or the JWS [RFC7515] header parameter in the Request Object.

If the RP doesn't include the trust_chain (Section 10.1.1.1) request parameter or the JWS [RFC7515] header parameter in the Request Object, or the OP doesn't support this feature, it then MUST validate the possible Trust Chains, starting with the RP's Entity Configuration as described in Section 8.1, and resolve the RP metadata with type `openid_relying_party`.

The OP SHOULD furthermore consider the resolved metadata of the RP and verify that it complies with the client metadata specification in OpenID Connect Dynamic Client Registration 1.0 [OpenID.Registration].

Once the OP has the RP's metadata, it can verify that the client was actually the one sending the Authentication Request by verifying the signature of the Request Object using the key material the client published through its metadata (underneath `metadata/openid_relying_party`).

10.1.1.2. Using Pushed Authorization

Pushed Authorization Requests [RFC9126] provide an interoperable way to push the payload of a Request Object directly to the AS in exchange for a `request_uri`.

When it comes to request authentication, the applicable methods are four:

- Using a JWT for Client authentication as described for `private_key_jwt` in Section 9 of OpenID Connect Core 1.0 [OpenID.Core].
- mTLS as described in Section 2.2 of [RFC8705] based on self-signed certificates. In this case, the self-signed certificate MUST be present as the value of an `x5c` claim for one key in the JWK Set describing the RP's keys.
- mTLS as described in Section 2.1 of [RFC8705] based on public key infrastructure (PKI).
- A Request Object as described in Section 6 of OpenID Connect Core 1.0 [OpenID.Core]

Note that if mTLS is used, TLS client authentication MUST be configured, and, in the case of self-signed certificates, the server must omit the certificate chain validation.

Using the example above, a request could look like this:

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

redirect_uri=https%3A%2F%2Frp.example.com%2Fauthz_cb
&scope=openid+profile+email+address+phone
&response_type=code
&nonce=4LX0mFMxdBjkGmtx7a8WIOnB
&state=YmX8PM9I7WbNoMnnieKKBiptVW0sP2OZ
&client_id=https%3A%2F%2Frp.example.com
&client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3A
  client-assertion-type%3Ajwt-bearer
&client_assertion=eyJhbGciOiJSUzI1NiIsImtpZCI6ImRVTjJ
  hMDF3Umtoa1NXcGxRVGh2Y1ZCSU5VSXdUVWRPVUZVMlRtVnJTbW
  hFUVhnelpYbHBUemRRTkEifQ.eyJzdWIiOiAiaHR0cHM6Ly9ycC
  5leGFtcGxlLmNvbSIsICJpc3MiOiAiaHR0cHM6Ly9ycC5leGFtc
  GxlLmNvbSIsICJpYXQiOiAxNTg5NzA0NzAxLCAiZXhwIjogMTU4
  OTcwNDc2MSwgImF1ZCI6ICJodHRwczovL29wLmV4YW1wbGUub3J
  nL2F1dGhvcml6YXRpb24iLCAianRpIjogIjM5ZDVhZTU1MmQ5Yz
  Q4ZjBiOTEyZGM1NTY4ZWQ1MGQ2In0.oUt9Knx_lxb4V2S0tyNFH
  CNZeP7sImBy5XDsFxv1cUpGkAojNXSy2dnU5HEzscMgNW4wguz6
  KDkC01aq5OfN04SuVItS66bsx0h4Gs7grKAp_51bClzreBVzU4g
  _-dFTgF15T9VLIgM_juFNPA_g4Lx7Eb5r37rWTUrzXdmfxeou0X
  FC2p9BIqItU3m9gmH0ojdBCUX5Up0iDsys6_npYomqitAcvaBRD
  PiuUBa5Iar9HVR-H7FMAr7aq7s-dH5gx2CHIfM3-qlc2-_Apsy0
  BrQl6VePR6j-3q6JCWvNw7l4_F2UpHeanHb31fLKQbK-1yoXDNz
  DwA7B0ZqmuSmMFQ
```

10.1.1.2.1. Processing the Authentication Request

All the assumptions and requirements already defined in Section 10.1.1.1.2 also apply to Pushed Authorization Requests [RFC9126].

Once the OP has the RP's metadata, it can verify the client using the keys published underneath the `metadata/openid_relying_party` element. This is where it diverges depending on which client authentication method was used.

private_key_jwt
: If this method is used, then the OP will try to verify the signature of the signed JWT using the key material published by the RP in its metadata. If the authentication is successful, then the registration is regarded as correct.

tls_client_auth
: If mTLS is used and the certificate used was not self-signed, then the Subject Alternative Name of the certificate MUST match the Entity Identifier of the RP.

self_signed_tls_client_auth
: If mTLS is using a self-signed certificate, then the certificate MUST be present as the value of an `x5c` claim for one key in the JWK Set describing the RP's keys.

10.1.2. Authentication Error Response

If the OP fails to establish trust with the RP, it SHOULD use an appropriate error code and an `error_description` that aids the RP in understanding what is wrong.

In addition to the error codes defined in Section 3.1.2.6 of OpenID Connect Core, this specification also defines the following error codes:

missing_trust_anchor
: No trusted Trust Anchor could be found.

validation_failed
: Trust chain validation failed.

The following is a non-normative example error response:

```
HTTP/1.1 302 Found
  Location: https://client.example.org/cb?
    error=missing_trust_anchor
    &error_description=
      Could%20not%20find%20a%20trusted%20anchor
    &state=af0ifjsldkj
```

10.1.3. Possible Other Uses of Automatic Registration

Note that we anticipate that Automatic Registration could be employed for use cases beyond OpenID Connect Federation. If used with pure OAuth 2.0, for instance, Automatic Registration would occur during Authorization Requests. If for a particular such use case, the client does not need to be securely identified, then the requirement for signed requests could be relaxed.

10.2. Explicit Registration

This method involves performing an explicit registration step for a new client before the RP interacts with an OP for the first time, similar to the process specified by OpenID Connect Dynamic Client Registration 1.0 [OpenID.Registration], but where the client registration request contains the Entity Configuration or an entire Trust Chain. Appendix A.3.2 contains an example using explicit client registration.

10.2.1. Client Registration

10.2.1.1. Client Registration Request

An OP that supports OpenID Dynamic Client Registration as extended by this specification, signals this by having the claim `federation_registration_endpoint` in the OP's metadata.

Given that the OP supports Explicit Registration, the RP progresses as follows:

1. Once it has a list of acceptable Trust Anchors to the OP, it MUST choose the subset it wants to progress with. This subset can be as small as one Trust Anchor, but it can also contain more than one.
2. Using this subset of Trust Anchors, the RP will choose a set of `authority_hints` from the hints that are available to it. Each hint MUST, when used as a starting point for Trust Chain collection, lead to at least one of the Trust Anchors in the subset.
3. The RP will now construct its Entity Configuration where the metadata statement chosen is influenced by the OP's metadata and where the `authority_hints` included are picked by the process described above.
4. The RP MAY include its Entity Configuration in a Trust Chain regarding itself. The Registration Request will contain an array containing the sequence of the statements that compose the Trust Chain between the RP that makes the request and the selected Trust Anchor.
5. The Entity Configuration, or the entire Trust Chain, is sent, using POST, to the `federation_registration_endpoint` defined in this document. There are no parameters in the POST; the Entity Configuration or Trust Chain is the entire POST body.
6. The content type of the Registration Request MUST be set to `application/entity-statement+jwt` if it contains only the Entity Configuration of the requestor. Otherwise, if it contains the Trust Chain, the content type of the Registration Request MUST be set to `application/trust-chain+json`.

10.2.1.2. Client Registration Response

10.2.1.2.1. OP Constructing the Response

1. After the OP receives the registration request, it checks if this contains an Entity Configuration or an entire Trust Chain.
2. If the request contains an Entity Configuration, the OP collects and evaluates the Trust Chains starting with the `authority_hints` in the Entity Configuration of the requestor. After it has verified at least one Trust Chain, it MUST verify that the signature on the received registration request is correct. If the OP finds more than one acceptable Trust Chain, it MUST choose one Trust Anchor from those chains as the one it will proceed with.
3. If the request contains a Trust Chain, the OP MAY evaluate the statements in the Trust Chain to make its Federation Entity Discovery procedure more efficient, especially if the RP shows more than a single authority hint in its Entity Configuration.
4. At this point, if there already exists a client registration under the same Entity Identifier, then that registration MUST be regarded as invalid. Note that key material from the previous registration SHOULD be kept to enable verifying signatures or decrypting archived data.
5. The OP will now construct an Entity Statement with metadata policies and assertions that when applied to the RP's metadata statement will result in metadata that the OP finds acceptable. Note that the Client ID the OP chooses does not have to be the Entity Identifier of the RP. The Entity Statement MUST include a `trust_anchor_id` claim and a single-valued `authority_hints` claim that correspond to the Trust Chain chosen in step 2.
6. The OP will sign the Entity Statement and return it as the registration response to the RP.
7. A successful response MUST have an HTTP status code 200 and a content type set to `application/entity-statement+jwt`.
8. If the response is negative, the response SHOULD be produced in accordance with what is defined in Section 7.7.

10.2.1.2.2. RP Parsing the Response

1. If the response is positive, the RP verifies the correctness of the received Entity Statement, making sure that one of the `authority_hints` it added to the registration request will lead to the Trust Anchor the OP named using the claim `trust_anchor_id`.
2. The RP MUST NOT apply metadata policies and assertions from the Trust Chains that the OP provides because those are not valid for the RP's metadata. The RP MUST apply policies and assertions to the metadata using one of its own Trust Chains that ends in the Trust Anchor that the OP chose. Once it has applied those policies and assertions, it can then apply the policies and assertions returned from the OP. This application of policies and assertions is equivalent to adding the OP's metadata policies and assertions to the Trust Chain in between the RP's and its immediate superior's Entity Statements. When the RP has applied all the metadata policies and assertions to its metadata statement, it then stores the result and will use the agreed-upon metadata when talking to the OP.
3. At this point, the RP also knows which Trust Chain it should use when evaluating the OP's metadata. It can therefore apply the metadata policies and assertions on the OP's metadata using the relevant Trust Chain and store the result as the OP's metadata.
4. If the RP does not accept the received Entity Statement for some reason, then it has the choice to restart the registration process or to give up.

10.2.2. After Explicit Client Registration

A client registration using the explicit method is not expected to be valid forever. The Entity Statements exchanged all have expiration times, which means that the registration will eventually time out. An OP can also, for administrative reasons, decide that a client registration is not valid anymore. An example of this

could be the OP leaving the federation used to register an RP.

The temporary nature of explicit registration means that an RP must expect its registration to become invalidated at any time, causing RP requests to the OP, such as authorization, token or UserInfo requests, to fail. An RP MAY devise appropriate strategies to re-register with the OP and restart the transaction when such a condition occurs.

10.2.2.1. What the RP MUST Do

At regular intervals, the RP MUST:

1. Starting with the OP's Entity Statement, resolve and verify the Trust Chains it chooses to use when constructing the registration request. If those Trust Chains do not exist anymore or do not verify, then the registration SHOULD be regarded as invalid, and a new registration process SHOULD be started.
2. If the OP's Entity Statement was properly formed, the RP must now verify that the Entity Statement it received about itself from the OP is still valid. Again, if that is not the case, the registration SHOULD be regarded as invalid and a new registration process SHOULD be started.

What is regarded as reasonable intervals will depend on federation policies and risk assessment by the maintainer of the RP.

10.2.2.2. What the OP MUST Do

At regular intervals, the OP MUST:

1. If the signature on the registration request has expired, it MUST mark the registration as invalid and demand that the RP MUST re-register. Else
2. starting with the RP's client registration request, the OP MUST verify that there still is a valid Trust Chain terminating in the Trust Anchor the OP chose during the registration process.

10.2.3. Expiration Times

An OP MUST NOT assign an expiration time to an RP's registration that is later than the trust chain's expiration time.

10.3. Differences between Automatic Registration and Explicit Registration

The primary differences between Automatic Registration and Explicit Registration are:

- With Automatic Registration, there is no registration step prior to the Authentication Request, whereas with Explicit Registration, there is. (OpenID Connect Dynamic Client Registration 1.0 [OpenID.Registration] and OAuth 2.0 Dynamic Client Registration [RFC7591]) also employ a prior registration step.)
- With Automatic Registration, the Client ID value is the RP's Entity Identifier and is supplied to the OP by the RP, whereas with Explicit Registration, a Client ID is assigned by the OP and supplied to the RP.
- Instead of using a Client Secret to authenticate the client, with Automatic Registration, the client is authenticated by means of the RP proving that it controls a private key corresponding to one of its Entity Configuration's public keys.

10.4. Rationale for the Trust Chain in the Request

Both Automatic and Explicit Client Registration support the submission of the Trust Chain embedded in the Request, calculated by the requestor and related to itself. This enables the following benefits in a federation:

It solves the problem of OPs using RP metadata that has become stale. This stale data may occur when the OP uses cached RP metadata from a Trust Chain that hasn't reached its expiration time yet. The RP MAY notify the OP that a change has taken place by passing in the request the trust_chain (Section 3.2.1) header parameter or the `trust_chain` request parameter , thus letting the OP update its client registration and prevent potential temporary faults due to stale metadata.

It enables the RP to pass a verifiable hint for which trust path to take in order to build the Trust Chain. This can reduce the costs of RP Federation Entity Discovery for OPs in complex federations where the RP has multiple Trust Anchors or the Trust Chain resolution may result in dead-ends.

It enables direct passing of the Entity Configuration, including any present Trust Marks, thus saving the OP from having to make an HTTP request to the RP `/.well-known/openid-federation` endpoint.

11. Claims Languages and Scripts

Human-readable Claim Values and Claim Values that reference human-readable values MAY be represented in multiple languages and scripts. This specification enables such representations in the same manner as defined in Section 5.2 of OpenID Connect Core 1.0 [OpenID.Core]. The paragraphs that follow are excerpted from there.

To specify the languages and scripts, BCP47 [RFC5646] language tags are added to member names, delimited by a # character. For example, `family_name#ja-Kana-JP` expresses the Family Name in Katakana in Japanese, which is commonly used to index and represent the phonetics of the Kanji representation of the same represented as `family_name#ja-Hani-JP`. As another example, both `website` and `website#de` Claim Values might be returned, referencing a Web site in an unspecified language and a Web site in German.

Since Claim Names are case sensitive, it is strongly RECOMMENDED that language tag values used in Claim Names be spelled using the character case with which they are registered in the IANA "Language Subtag Registry" [IANA.Language]. In particular, language names are often spelled with lowercase characters, region names are spelled with uppercase characters, and scripts are spelled with mixed-case characters. However, since BCP47 language tag values are case-insensitive, implementations SHOULD interpret the language tag values supplied in a case-insensitive manner.

Per the recommendations in BCP47, language tag values for Claims SHOULD only be as specific as necessary. For instance, using `fr` might be sufficient in many contexts, rather than `fr-CA` or `fr-FR`. Where possible, OPs SHOULD try to match requested Claim locales with Claims it has. For instance, if the Client asks for a Claim with a `de` (German) language tag and the OP has a value tagged with `de-CH` (Swiss German) and no generic German value, it would be appropriate for the OP to return the Swiss German value to the Client. (This intentionally moves as much of the complexity of language tag matching to the OP as possible to simplify Clients.)

12. IANA Considerations

12.1. OAuth Authorization Server Metadata Registry

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

12.1.1. Registry Contents

- Metadata Name: `client_registration_types_supported`
- Metadata Description: Client Registration Types Supported

- Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
- Specification Document(s): Section Section 4.3 of [[this specification]]

- Metadata Name: `organization_name`
- Metadata Description: Human-readable name representing the organization owning the OP
- Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
- Specification Document(s): Section 4.3 of [[this specification]]

- Metadata Name: `federation_registration_endpoint`
- Metadata Description: Federation Registration Endpoint
- Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
- Specification Document(s): Section 4.3 of [[this specification]]

- Metadata Name: `request_authentication_methods_supported`
- Metadata Description: Authentication request authentication methods supported
- Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
- Specification Document(s): Section 4.3 of [[this specification]]

- Metadata Name: `request_authentication_signing_alg_values_supported`
- Metadata Description: JSON array containing the JWS signing algorithms supported for the signature on the JWT used to authenticate the request
- Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
- Specification Document(s): Section 4.3 of [[this specification]]

- Metadata Name: `signed_jwks_uri`
- Metadata Description: URI pointing to a signed JWT having the Entity's JWK Set as its payload
- Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
- Specification Document(s): Section 4.3 of [[this specification]]

- Metadata Name: `jwks`
- Metadata Description: JSON Web Key Set document, passed by value
- Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
- Specification Document(s): Section 4.3 of [[this specification]]

12.2. OAuth Dynamic Client Registration Metadata Registration

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

12.2.1. Registry Contents

- Client Metadata Name: `client_registration_types`

- Client Metadata Description: An array of strings specifying the client registration types the RP wants to use
- Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
- Specification Document(s): Section Section 4.2 of [[this specification]]

- Client Metadata Name: `organization_name`
- Client Metadata Description: Human-readable name representing the organization owning the RP
- Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
- Specification Document(s): Section Section 4.2 of [[this specification]]

- Client Metadata Name: `signed_jwks_uri`
- Client Metadata Description: URI pointing to a signed JWT having the Entity's JWK Set as its payload
- Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
- Specification Document(s): Section Section 4.2 of [[this specification]]

12.3. OAuth Extensions Error Registration

This section registers the following values in the IANA "OAuth Extensions Error Registry" registry [IANA.OAuth.Parameters] established by [RFC6749].

12.3.1. Registry Contents

- Name: missing_trust_anchor
- Usage Location: Authorization Request
- Protocol Extension: OpenID Connect Federation
- Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
- Reference: Section 10.1.2 of [[this specification]]

- Name: validation_failed
- Usage Location: Authorization Request
- Protocol Extension: OpenID Connect Federation
- Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
- Reference: Section 10.1.2 of [[this specification]]

12.4. Media Type Registration

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

12.4.1. Registry Contents

- Type name: application
- Subtype name: entity-statement+jwt
- Required parameters: n/a
- Optional parameters: n/a

- Encoding considerations: binary; An Entity Statement is a JWT; JWT values are encoded as a series of base64url-encoded values (some of which may be the empty string) separated by period ('.') characters.
- Security considerations: See Section 13 of [[this specification]]
- Interoperability considerations: n/a
- Published specification: [[this specification]]
- Applications that use this media type: Applications that use [[this specification]]
- Fragment identifier considerations: n/a
- 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:
  Michael B. Jones, mbj@microsoft.com
- Intended usage: COMMON
- Restrictions on usage: none
- Author: Michael B. Jones, mbj@microsoft.com
- Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
- Provisional registration? No

- Type name: application
- Subtype name: trust-mark+jwt
- Required parameters: n/a
- Optional parameters: n/a
- Encoding considerations: binary; A Trust Mark is a JWT; JWT values are encoded as a series of base64url-encoded values (some of which may be the empty string) separated by period ('.') characters.
- Security considerations: See Section 13 of [[this specification]]
- Interoperability considerations: n/a
- Published specification: [[this specification]]
- Applications that use this media type: Applications that use [[this specification]]
- Fragment identifier considerations: n/a
- 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:
  Michael B. Jones, mbj@microsoft.com
- Intended usage: COMMON
- Restrictions on usage: none
- Author: Michael B. Jones, mbj@microsoft.com
- Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net

- Provisional registration? No

- Type name: application
- Subtype name: resolve-response+jwt
- Required parameters: n/a
- Optional parameters: n/a
- Encoding considerations: binary; An Entity Resolve Response is a signed JWT; JWT values are encoded as a series of base64url-encoded values (some of which may be the empty string) separated by period ('.') characters.
- Security considerations: See Section 13 of [[this specification]]
- Interoperability considerations: n/a
- Published specification: [[this specification]]
- Applications that use this media type: Applications that use [[this specification]]
- Fragment identifier considerations: n/a
- 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:
  Michael B. Jones, mbj@microsoft.com
- Intended usage: COMMON
- Restrictions on usage: none
- Author: Michael B. Jones, mbj@microsoft.com
- Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
- Provisional registration? No

- Type name: application
- Subtype name: trust-chain+json
- Required parameters: n/a
- Optional parameters: n/a
- Encoding considerations: binary; A Trust Chain is a JSON array of JWTs; JWT values are encoded as a series of base64url-encoded values (some of which may be the empty string) separated by period ('.') characters.
- Security considerations: See Section 13 of [[this specification]]
- Interoperability considerations: n/a
- Published specification: [[this specification]]
- Applications that use this media type: Applications that use [[this specification]]
- Fragment identifier considerations: n/a
- 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:
  Michael B. Jones, mbj@microsoft.com
- Intended usage: COMMON
- Restrictions on usage: none
- Author: Michael B. Jones, mbj@microsoft.com
- Change controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
- Provisional registration? No

12.5. OAuth Parameter Registry

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

12.5.1. Registry Contents

- Parameter Name: `trust_chain`
- Parameter Usage Location: authorization request
- Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
- Specification Document(s): Section Section 10.1.1.1.2 of [[this specification]]

12.6. JWS Header Registry

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

12.6.1. Registry Contents

- Header Parameter Name: `trust_chain`
- Header Parameter Description: OpenID Connect Federation Trust Chain
- Header Parameter Usage Location: JWS
- Change Controller: OpenID Foundation Artifact Binding Working Group - openid-specs-ab@lists.openid.net
- Specification Document(s): Section Section 3.2 of [[this specification]]

13. Security Considerations

13.1. Denial-of-Service Attack Prevention

Some of the interfaces defined in this specification could be used for Denial-of-Service attacks (DoS), most notably, the Resolve endpoint (Section 7.3), Explicit Client Registration (Section 10.2), and Automatic Client Registration (Section 10.1) can be exploited as vectors of HTTP propagation attacks. If these endpoints are provided, some adequate defense methods are required, such as those described below and in [RFC4732].

A Trust Mark can be statically validated using the public key of its issuer. The static validation of the Trust Marks represents a filter against propagation attacks. An attacker could exploit the Federation Entity Discovery mechanism and use an OIDC Federation to propagate many HTTP requests. For each authorization request crafted by an anonymous client, the OP would produce approximately three HTTP requests to third parties in the absence of Intermediates, and at least five HTTP requests with at least one Intermediate. If an OP doesn't find at least one valid Trust Mark in an Entity Configuration, it should reject the request and temporary ban the requestor.

If client authentication is not demanded at the Resolve endpoint then incoming requests should not result in the immediate collection (Federation Entity Discovery process) and evaluation of Trust Chains by default.

13.2. Unsigned Error Messages

One of the fundamental design goals of this protocol is to protect messages end-to-end. This can not be accomplished by demanding TLS since TLS, in lots of cases, is not end-to-end but ends in a HTTPS to HTTP Reverse Proxy. Allowing unsigned error messages therefore opens up an attack vector for someone who wants to run a Denial of Service attack. This is not specific to OpenID Connect Federation but equally valid for other protocols when HTTPS to HTTP reverse proxies are used.

14. References

14.1. Normative References

[OpenID.Core] Sakimura, N., Bradley, J., Jones, M.B., de Medeiros, B., and C. Mortimore, "OpenID Connect Discovery 1.0", 3 August 2015, <http://openid.net/specs/openid-connect-core-1_0.html>.

[OpenID.Discovery] Sakimura, N., Bradley, J., Jones, M.B., and E. Jay, "OpenID Connect Discovery 1.0", 3 August 2015, <http://openid.net/specs/openid-connect-discovery-1_0.html>.

[OpenID.Registration] Sakimura, N., Bradley, J., and M.B. Jones, "OpenID Connect Dynamic Client Registration 1.0", 3 August 2015, <http://openid.net/specs/openid-connect-registration-1_0.html>.

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

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/info/rfc3339>.

[RFC4732] Handley, M., Ed., Rescorla, E., Ed., and IAB, "Internet Denial-of-Service Considerations", RFC 4732, DOI 10.17487/RFC4732, December 2006, <https://www.rfc-editor.org/info/rfc4732>.

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

[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>.

[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>.

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

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

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

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

[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>.

[RFC8705] Campbell, B., Bradley, J., Sakimura, N., and T. Lodderstedt, "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens", RFC 8705, DOI 10.17487/RFC8705, February 2020, <https://www.rfc-editor.org/info/rfc8705>.

[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>.

14.2. Informative References

[I-D.jones-oauth-resource-metadata] Jones, M. and P. Hunt, "OAuth 2.0 Protected Resource Metadata", Work in Progress, Internet-Draft, draft-jones-oauth-resource-metadata-01, 19 January 2017, <https://datatracker.ietf.org/doc/html/draft-jones-oauth-resource-metadata-01>.

[IANA.JOSE] IANA, "JSON Web Signature and Encryption Header Parameters", <https://www.iana.org/assignments/jose/>.

[IANA.Language] IANA, "Language Subtag Registry", <https://www.iana.org/assignments/language-subtag-registry>.

[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>.

[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>.

[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>.

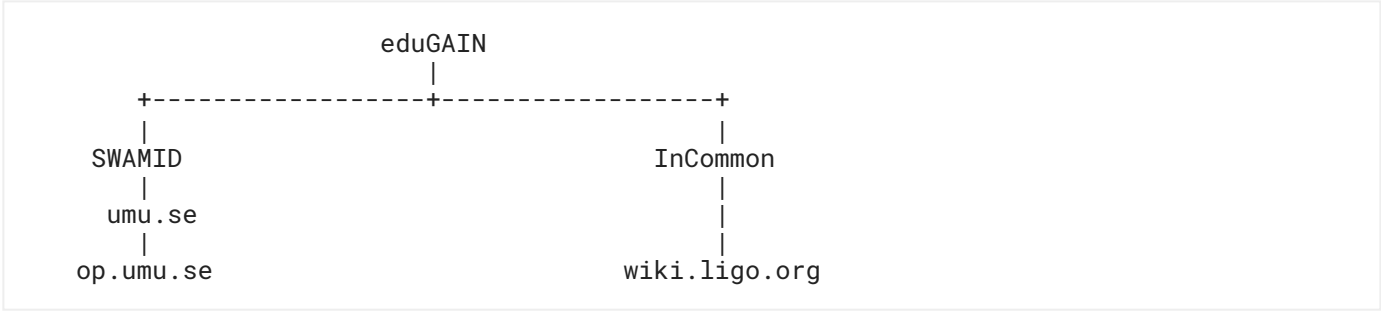
[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[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>.

Appendix A. Provider Information Discovery and Client Registration in a Federation

Let us assume the following: The project LIGO would like to offer access to its wiki to all OPs in eduGAIN. LIGO is registered with the InCommon federation.

The players



Both SWAMID and InCommon are identity federations in their own right. They also have in common that they are both members of the eduGAIN federation.

SWAMID and InCommon are different in how they register Entities. SWAMID registers organizations and lets the organizations register Entities that belong to the organization, while InCommon registers all Entities directly and not beneath any organization Entity. Hence the differences in depth in the federations.

Let us assume a researcher from Umea University would like to login to the LIGO Wiki. At the Wiki, the researcher will use some kind of discovery service to find the home identity provider (op.umu.se)

Once the RP part of the Wiki knows which OP it SHOULD talk to, it has to find out a couple of things about the OP. All of those things can be found in the metadata. But finding the metadata is not enough; the RP also has to trust the metadata.

Let us make a detour and start with what it takes to build a federation.

A.1. Setting Up a Federation

These are the steps to set up a federation infrastructure:

- Generation of signing keys. These must be public/private key pairs.
- Set up a signing service that can sign JWTs/Entity Statements using the Federation Entity Keys.
- Set up web services that can publish signed Entity Statements, one for the URL corresponding to the federation's Entity Identifier returning an Entity Configuration and the other one providing the fetch Entity Statement endpoint, as described in Section 7.1.1.

Once the previous requirements have been satisfied, a Federation Operator can add Entities to the federation. Adding an Entity comes down to:

- Providing the Entity with the federation's Entity Identifier and the public part of the key pairs used by the federation operator for signing Entity Statements.
- Getting the Entity's Entity Identifier and the JWKS that the Entity plans to publish in its Entity Configuration.

Before the federation operator starts adding Entities, there must be policies on who can be part of the federation and the layout of the federation. Is it supposed to be a one-layer federation like InCommon, a two-layer one like the SWAMID federation, or a multi-layer federation? The federation may also want to think about implementing other policies using the federation policy framework, as described in Section 5.

With the federation in place, things can start happening.

A.2. The LIGO Wiki Discovers the OP's Metadata

Federation Entity Discovery is a sequence of steps that starts with the RP fetching the Entity Configuration of the Leaf Entity (in this case, https://op.umu.se) using the process defined in Section 6. What follows after that is this sequence of steps:

1. Pick out the immediate superior Entities using the authority hints
2. Fetch the configuration for each such Entity. This uses the process defined in Section 6
3. Using the fetch endpoint of the superiors to ask for information about the Subordinate Entity Section 7.1.1.

How many times this has to be repeated depends on the depth of the federation. What follows below is the result of each step the RP has to take to find the OP's metadata using the federation setup described above.

When building the Trust Chain, the Entity Statements issued by a superior about its Subordinate are used together with the Entity Configuration issued by the Leaf.

The Entity Configuration concerning Intermediates is not part of the Trust Chain.

A.2.1. Configuration Information for op.umu.se

The LIGO WIKI RP fetches the Entity Configuration from the OP (op.umu.se) using the process defined in Section 6.

The result is this Entity Configuration.


```
{
  "authority_hints": [
    "https://umu.se"
  ],
  "exp": 1568397247,
  "iat": 1568310847,
  "iss": "https://op.umu.se",
  "sub": "https://op.umu.se",
  "jwks": {
    "keys": [
      {
        "e": "AQAB",
        "kid": "dEEtRjlzY3djcENuT01wOGxrZlkxb3RIQVJlMTY0...",
        "kty": "RSA",
        "n": "x97YKqc9Cs-DNtFrQ7_vhXoH9bwkDWW6En2jJ044yH..."
      }
    ]
  },
  "metadata": {
    "openid_provider": {
      "issuer": "https://op.umu.se/openid",
      "signed_jwks_uri": "https://op.umu.se/openid/jwks.jose",
      "authorization_endpoint":
        "https://op.umu.se/openid/authorization",
      "client_registration_types_supported": [
        "automatic",
        "explicit"
      ],
      "request_parameter_supported": true,
      "grant_types_supported": [
        "authorization_code",
        "implicit",
        "urn:ietf:params:oauth:grant-type:jwt-bearer"
      ],
      "id_token_signing_alg_values_supported": [
        "ES256", "RS256"
      ],
      "logo_uri":
        "https://www.umu.se/img/umu-logo-left-neg-SE.svg",
      "op_policy_uri":
        "https://www.umu.se/en/website/legal-information/",
      "response_types_supported": [
        "code",
        "code id_token",
        "token"
      ],
      "subject_types_supported": [
        "pairwise",
        "public"
      ],
      "token_endpoint": "https://op.umu.se/openid/token",
      "federation_registration_endpoint":
        "https://op.umu.se/openid/fedreg",
      "token_endpoint_auth_methods_supported": [
        "client_secret_post",
        "client_secret_basic",
        "client_secret_jwt",
        "private_key_jwt"
      ]
    }
```

```
    }
}
```

The `authority_hints` points to the Intermediate Entity `https://umu.se`. So that is the next step.

This Entity Configuration is the first link in the Trust Chain.

A.2.2. Configuration Information for 'https://umu.se'

The LIGO RP fetches the Entity Configuration from "https://umu.se" using the process defined in Section 6.

The request will look like this:

```
GET /.well-known/openid-federation HTTP/1.1
Host: umu.se
```

And the GET will return:

```
{
  "authority_hints": [
    "https://swamid.se"
  ],
  "exp": 1568397247,
  "iat": 1568310847,
  "iss": "https://umu.se",
  "sub": "https://umu.se",
  "jwks": {
    "keys": [
      {
        "e": "AQAB",
        "kid": "endwNUZrNTJsX2NyQlp4bjhVcTFTTVltR2gxV2RV...",
        "kty": "RSA",
        "n": "vXdXzZwQo0hxRSmZEcDIsnpg-CMEkor50SOG-1XUlM..."
      }
    ]
  },
  "metadata": {
    "federation_entity": {
      "contacts": "ops@umu.se",
      "federation_fetch_endpoint": "https://umu.se/oidc/fedapi",
      "homepage_uri": "https://www.umu.se",
      "organization_name": "UmU"
    }
  }
}
```

The only piece of information that is used from this Entity Statement is the `federation_fetch_endpoint`, which is used in the next step.

A.2.3. Entity Statement Published by 'https://umu.se' about 'https://op.umu.se'

The RP uses the fetch endpoint provided by https://umu.se as defined in Section 7.1.1 to fetch information about "https://op.umu.se".

The request will look like this:

```
GET /oidc/fedapi?sub=https%3A%2F%2Fop.umu.se&
iss=https%3A%2F%2Fumu.se HTTP/1.1
Host: umu.se
```

and the result is this:

```
{
  "exp": 1568397247,
  "iat": 1568310847,
  "iss": "https://umu.se",
  "jwks": {
    "keys": [
      {
        "e": "AQAB",
        "kid": "dEEtRjlzY3djcENuT01wOGxrZlkxb3RIQVJlMTY0...",
        "kty": "RSA",
        "n": "x97YKqc9Cs-DNtFrQ7_vhXoH9bwkDWW6En2jJ044yH..."
      }
    ]
  },
  "metadata_policy": {
    "openid_provider": {
      "contacts": {
        "add": [
          "ops@swamid.se"
        ]
      },
      "organization_name": {
        "value": "University of Ume\u00e5"
      },
      "subject_types_supported": {
        "value": [
          "pairwise"
        ]
      },
      "token_endpoint_auth_methods_supported": {
        "default": [
          "private_key_jwt"
        ],
        "subset_of": [
          "private_key_jwt",
          "client_secret_jwt"
        ],
        "superset_of": [
          "private_key_jwt"
        ]
      }
    }
  },
  "sub": "https://op.umu.se"
}
```

This is the second link in the Trust Chain.

Notable here is that this path leads to two Trust Anchors using the same next step ("https://swamid.se").

A.2.4. Configuration Information for 'https://swamid.se'

The LIGO Wiki RP fetches the Entity Configuration from "https://swamid.se" using the process defined in Section 6.

The request will look like this:

```
GET /.well-known/openid-federation HTTP/1.1
Host: swamid.se
```

And the GET will return:

```
{
  "authority_hints": [
    "https://edugain.geant.org"
  ],
  "exp": 1568397247,
  "iat": 1568310847,
  "iss": "https://swamid.se",
  "jwks": {
    "keys": [
      {
        "e": "AQAB",
        "kid": "N1pQTzFxUXZ1RXVsUkVuMG5uMnVDSURGRVdhUzdO...",
        "kty": "RSA",
        "n": "3EQc6cR_GSBq9km9-WCHY_lWJZWkcn0M05TGtH6D9S..."
      }
    ]
  },
  "metadata": {
    "federation_entity": {
      "contacts": "ops@swamid.se",
      "federation_fetch_endpoint":
        "https://swamid.se/fedapi",
      "homepage_uri": "https://www.sunet.se/swamid/",
      "organization_name": "SWAMID"
    }
  },
  "sub": "https://swamid.se"
}
```

The only piece of information that is used from this Entity Statement is the `federation_fetch_endpoint`, which is used in the next step.

A.2.5. Entity Statement Published by 'https://swamid.se' about 'https://umu.se'

The LIGO Wiki RP uses the fetch endpoint provided by "https://swamid.se" as defined in Section 7.1.1 to fetch information about "https://umu.se".

The request will look like this:

```
GET /fedapi?sub=https%3A%2F%2Fumu.se&
iss=https%3A%2F%2Fswamid.se HTTP/1.1
Host: swamid.se
```

and the result is this:

```
{
  "exp": 1568397247,
  "iat": 1568310847,
  "iss": "https://swamid.se",
  "jwks": {
    "keys": [
      {
        "e": "AQAB",
        "kid": "endwNUZrNTJsX2NyQlp4bjhVcTFTTVltR2gxV2RV...",
        "kty": "RSA",
        "n": "vXdXzZwQo0hxRSmZEcDIsnpg-CMEkor50SOG-1XUlM..."
      }
    ]
  },
  "metadata_policy": {
    "openid_provider": {
      "id_token_signing_alg_values_supported": {
        "subset_of": [
          "RS256",
          "ES256",
          "ES384",
          "ES512"
        ]
      },
      "token_endpoint_auth_methods_supported": {
        "subset_of": [
          "client_secret_jwt",
          "private_key_jwt"
        ]
      },
      "userinfo_signing_alg_values_supported": {
        "subset_of": [
          "ES256",
          "ES384",
          "ES512"
        ]
      }
    }
  },
  "sub": "https://umu.se"
}
```


This is the third link in the Trust Chain.

If we assume that the issuer of this Entity Statement is not in the list of Trust Anchors the LIGO Wiki RP has access to, we have to go one step further.

A.2.6. Configuration Information for 'https://edugain.geant.org'

RP fetches the Entity Configuration from "https://edugain.geant.org" using the process defined in Section 6.

The request will look like this:

```
GET /.well-known/openid-federation HTTP/1.1
Host: edugain.geant.org
```

And the GET will return:

```
{
  "exp": 1568397247,
  "iat": 1568310847,
  "iss": "https://edugain.geant.org",
  "sub": "https://edugain.geant.org",
  "jwks": {
    "keys": [
      {
        "e": "AQAB",
        "kid": "Sl9DcjFxR3hrRGdabUNIR21KT3dvdWMyc2VUM2Fr...",
        "kty": "RSA",
        "n": "xKlwocDXUw-mrvDSO4oRrTRrVuTwotoBFpozvlq-1q..."
      }
    ]
  },
  "metadata": {
    "federation_entity": {
      "federation_fetch_endpoint": "https://geant.org/edugain/api"
    }
  }
}
```

Within the Trust Anchor Entity Configuration, the Relying Party looks for the `federation_fetch_endpoint` and gets the updated Federation Entity Keys of the Trust Anchor. Each Entity within a Federation may change their Federation Entity Keys, or any other attributes, at any time. See Section 9.2 for futhers details.

A.2.7. Entity Statement Published by 'https://edugain.geant.org' about 'https://swamid.se'

The LIGO Wiki RP uses the fetch endpoint of https://edugain.geant.org as defined in Section 7.1.1 to fetch information about "https://swamid.se".

The request will look like this:

```
GET /edugain/api?sub=https%3A%2F%2Fswamid.se&
iss=https%3A%2F%2Fedugain.geant.org HTTP/1.1
Host: geant.org
```

and the result is this:

```
{
  "exp": 1568397247,
  "iat": 1568310847,
  "iss": "https://edugain.geant.org",
  "jwks": {
    "keys": [
      {
        "e": "AQAB",
        "kid": "N1pQTzFxUXZ1RXVsUkVuMG5uMnVDSURGRVdhUzd0...",
        "kty": "RSA",
        "n": "3EQc6cR_GSBq9km9-WCHY_lWJZWkcn0M05TGtH6D9S..."
      }
    ]
  },
  "metadata_policy": {
    "openid_provider": {
      "contacts": {
        "add": "ops@edugain.geant.org"
      }
    },
    "openid_relying_party": {
      "contacts": {
        "add": "ops@edugain.geant.org"
      }
    }
  },
  "sub": "https://swamid.se"
}
```


If we assume that the issuer of this statement appears in the list of Trust Anchors the LIGO Wiki RP has access to, this would be the fourth and final Entity Statement in the Trust Chain.

We now have the whole chain from the Entity Configuration of the Leaf Entity up until the last one that is issued by a Trust Anchor. All in all, we have:

1. Entity Configuration by the Leaf Entity (https://op.umu.se)
2. Entity Configuration by https://umu.se
3. Statement issued by https://umu.se about https://op.umu.se
4. Entity Configuration by https://swamid.se
5. Statement issued by https://swamid.se about https://umu.se
6. Entity Configuration by https://edugain.geant.org
7. Statement issued by https://edugain.geant.org about https://swamid.se

Using the public keys of the Trust Anchor that the LIGO Wiki RP has been provided within some secure out-of-band way, it can now verify the Trust Chain as described in Section 8.2.

A.2.8. Verified Metadata for op.umu.se

Having verified the chain, the LIGO Wiki RP can proceed with the next step.

Combining the metadata policies from the tree Entity Statements we have by a superior about its Subordinate and applying the combined policy to the metadata statement that the Leaf Entity presented, we get:

```
{
  "authorization_endpoint":
    "https://op.umu.se/openid/authorization",
  "claims_parameter_supported": false,
  "contacts": [
    "ops@swamid.se"
  ],
  "federation_registration_endpoint":
    "https://op.umu.se/openid/fedreg",
  "client_registration_types_supported": [
    "automatic",
    "explicit"
  ],
  "grant_types_supported": [
    "authorization_code",
    "implicit",
    "urn:ietf:params:oauth:grant-type:jwt-bearer"
  ],
  "id_token_signing_alg_values_supported": [
    "RS256",
    "ES256"
  ],
  "issuer": "https://op.umu.se/openid",
  "signed_jwks_uri": "https://op.umu.se/openid/jwks.jose",
  "logo_uri":
    "https://www.umu.se/img/umu-logo-left-neg-SE.svg",
  "organization_name": "University of Ume\u00e5",
  "op_policy_uri":
    "https://www.umu.se/en/website/legal-information/",
  "request_parameter_supported": true,
  "request_uri_parameter_supported": true,
  "require_request_uri_registration": true,
  "response_types_supported": [
    "code",
    "code id_token",
    "token"
  ],
  "subject_types_supported": [
    "pairwise"
  ],
  "token_endpoint": "https://op.umu.se/openid/token",
  "token_endpoint_auth_methods_supported": [
    "private_key_jwt",
    "client_secret_jwt"
  ]
}
```


We have now reached the end of the Provider Discovery process.

A.3. The Two Ways of Doing Client Registration

As described in Section 10, there are two ways that can be used to do client registration:

Automatic
: No negotiation between the RP and the OP is made regarding what features the client SHOULD use in future communication are done. The RP's published metadata filtered by the chosen trust chain's metadata policies defines the metadata that is to be used.

Explicit

The RP will access the `federation_registration_endpoint`, which provides the RP's metadata. The OP MAY return a metadata policy that adds restrictions over and above what the Trust Chain already has defined.

A.3.1. RP Sends Authentication Request (Automatic Registration)

The LIGO Wiki RP does not do any registration but goes directly to sending an Authentication Request.

Here is an example of such an Authentication Request:

```
GET /openid/authorization?
  request=eyJhbGciOiJSUzI1NiIsImtpZCI6ImRVTjJhMDF3Umtoa1NXc
    GxRVGh2Y1ZCSU5VSXdUVWRPVUZVMlRtVnJTbWhFUVhnelpYbHBUemRR
    TkEifQ.eyJyZXNwb25zZV90eXBlIjogImNvZGUiLCAic2NvcGUiOiAi
    b3BlbmlkIHByb2ZpbGUgZW1haWwiLCAiY2xpZW50X2lkIjogImh0dHB
    zOi8vd2lraS5saWdvLm9yZyIsICJzdGF0ZSI6ICIyZmY3ZTU4OS0zOD
    Q4LTQ2ZGEtYTNkMi05NDllMTIzNWU2NzEiLCAibm9uY2UiOiAiZjU4M
    WExODYtYWNhNC00NmIzLTk0ZmMtODA0ODQwODNlYjJjIiwgInJlZGly
    ZWN0X3VyaSI6ICJodHRwczovL3dpa2kubGlnby5vcmcvb3BlbmlkL2N
    hbGxiYWNrIiwgImlzcyI6ICIiLCAiaWF0IjogMTU5MzU4ODA4NSwgIm
    F1ZCI6ICJodHRwczovL29wLnVtdS5zZSJ9.cRwSFNcDx6VsacAQDcIx
    5OAt_Pj30I_uUKRh04N4QJd6MZ0f50sETRv8uspSt9fMa-5yV3uzthX
    _v8OtQrV33gW1vzgOSRCdHgeCN40StbzjFk102seDwtU_Uzrcsy7KrX
    YSBp8U0dBDjuxC6h18L8ExjeR-NFjcrhy0wwua7Tnb4QqtN0QCia6DD
    8QBNVTL1Ga0YPmMdT25wS26wug23IgpbZB20VUosmMGgGtS5yCI5AwK
    Bhozv-oBH5KxxHzH1Oss-RkIGiQnjRnaWwEOTITmfZWra1eHP254wFF
    2se-EnWtz1q2XwsD9NSsOEJwWJPirPPJaKso8ng6qrrOSgw
  &response_type=code
  &client_id=https%3A%2F%2Fwiki.ligo.org
  &redirect_uri=https%3A%2F%2Fwiki.ligo.org/openid/callback
  &scope=openid+profile+email
  HTTP/1.1
Host: op.umu.se
```

The OP receiving this Authentication Request will, unless the RP is already registered, start to dynamically fetch and establish trust with the RP.

A.3.1.1. OP Fetches Entity Statements

The OP needs to establish a Trust Chain for the RP (wiki.ligo.org). The OP in this example is configured with public keys of two federations:

- https://edugain.geant.org
- https://swamid.se

The OP starts to resolve metadata for the Client Identifier https://wiki.ligo.org by fetching the Entity Configuration using the process described in Section 6.

The process is the same as described in Appendix A.2 and will result in a Trust Chain with the following Entity Statements:

1. Entity Configuration by the Leaf Entity https://wiki.ligo.org
2. Statement issued by https://incommon.org about https://wiki.ligo.org
3. Statement issued by https://edugain.geant.org about https://incommon.org

A.3.1.2. OP Evaluates the RP Metadata

Using the public keys of the Trust Anchor that the LIGO Wiki RP has been provided within some secure out-of-band way, it can now verify the Trust Chain as described in Section 8.2.

We will not list the complete Entity Statements but only the `metadata` and `metadata_policy` parts. There are two metadata policies:

edugain.geant.org

```
"metadata_policy": {
  "openid_provider": {
    "contacts": {
      "add": "ops@edugain.geant.org"
    }
  },
  "openid_relying_party": {
    "contacts": {
      "add": "ops@edugain.geant.org"
    }
  }
}
```

incommon.org

```
"metadata_policy": {
  "openid_relying_party": {
    "application_type": {
      "one_of": [
        "web",
        "native"
      ]
    },
    "contacts": {
      "add": "ops@incommon.org"
    },
    "grant_types": {
      "subset_of": [
        "authorization_code",
        "refresh_token"
      ]
    }
  }
}
```

Combining these and apply them to the metadata for wiki.ligo.org:

```
"metadata": {
  "application_type": "web",
  "client_name": "LIGO Wiki",
  "contacts": [
    "ops@ligo.org"
  ],
  "grant_types": [
    "authorization_code",
    "refresh_token"
  ],
  "id_token_signed_response_alg": "RS256",
  "signed_jwks_uri": "https://wiki.ligo.org/jwks.jose",
  "redirect_uris": [
    "https://wiki.ligo.org/openid/callback"
  ],
  "response_types": [
    "code"
  ],
  "subject_type": "public"
}
```

The final result is:

```
{
  "application_type": "web",
  "client_name": "LIGO Wiki",
  "contacts": [
    "ops@ligo.org",
    "ops@edugain.geant.org",
    "ops@incommon.org"
  ],
  "grant_types": [
    "refresh_token",
    "authorization_code"
  ],
  "id_token_signed_response_alg": "RS256",
  "signed_jwks_uri": "https://wiki.ligo.org/jwks.jose",
  "redirect_uris": [
    "https://wiki.ligo.org/openid/callback"
  ],
  "response_types": [
    "code"
  ],
  "subject_type": "public"
}
```

Once the Trust Chain and the final Relying Party metadata have been obtained, the OpenID Provider has everything needed to validate the signature of the Request Object in the Authentication Request, using the public keys made available at the `signed_jwks_uri` endpoint.

A.3.2. Client Starts with Registration (Explicit Client Registration)

Here the LIGO Wiki RP sends a client registration request to the `federation_registration_endpoint` of the OP (op.umu.se). What it sends is an Entity Configuration.

The JWT Claims Set of that Entity Configuration might look like this:

```
{
  "iss": "https://wiki.ligo.org",
  "sub": "https://wiki.ligo.org",
  "iat": 1676045527,
  "exp": 1676063610,
  "aud": "https://op.umu.se",
  "metadata": {
    "openid_relying_party": {
      "application_type": "web",
      "client_name": "LIGO Wiki",
      "contacts": ["ops@ligo.org"],
      "grant_types": ["authorization_code"],
      "id_token_signed_response_alg": "RS256",
      "signed_jwks_uri": "https://wiki.ligo.org/jwks.jose",
      "redirect_uris": [
        "https://wiki.ligo.org/openid/callback"
      ],
      "response_types": ["code"],
      "subject_type": "public"
    }
  },
  "jwks": {
    "keys": [
      {
        "kty": "RSA",
        "use": "sig",
        "kid":
          "U2JTWHY0VFg0a2FEVVdTaHptVDJsNDNiSDk5MXRBVEtNSFVkeXZwb",
        "e": "AQAB",
        "n":
          "4AZjgqFwMhTVSLrpzzNcwaCyVD88C_Hb3Bmor97vH-2AzldhuVb8K..."
      }
    ]
  },
  "authority_hints": ["https://incommon.org"]
}
```

Once the OP has the Entity Configuration, it proceeds with the same sequence of steps as laid out in Appendix A.2.

The OP will end up with the same RP metadata described in Appendix A.3.1.2, but it now can return a metadata policy that it wants to be applied to the RP's metadata. This metadata policy will be combined with the Trust Chain's combined metadata policy before being applied to the RP's metadata.

If we assume that the OP does not support refresh tokens, it MAY want to add a metadata policy that says:

```
"metadata_policy": {
  "openid_relying_party": {
    "grant_types": {
      "subset_of": [
        "authorization_code"
      ]
    }
  }
}
```

Thus, the Entity Statement returned by the OP to the RP MAY look like this:

```
{
  "trust_anchor_id": "https://edugain.geant.org",
  "metadata_policy": {
    "openid_relying_party": {
      "contacts": {
        "add": [
          "ops@incommon.org",
          "ops@edugain.geant.org"
        ]
      }
    }
  },
  "metadata": {
    "openid_relying_party": {
      "client_id": "m3GyHw",
      "client_secret_expires_at": 1604049619,
      "client_secret":
        "cb44eed577f3b5edf3e08362d47a0dc44630b3dc6ea99f7a79205"
      "client_id_issued_at": 1601457619
    }
  },
  "authority_hints": [
    "https://incommon.org"
  ],
  "aud": "https://wiki.ligo.org",
  "jwks": {
    "keys": [
      {
        "kty": "RSA",
        "use": "sig",
        "kid":
          "U2JTWHY0VFg0a2FEVVdTaHptVDJsNDNiSDk5MXRBVEtNSFVkeXZwb",
        "e": "AQAB",
        "n":
          "4AZjgqFwMhTVSLrpzzNcwaCyVD88C_Hb3Bmor97vH-2AzldhuVb8K..."
      },
      {
        "kty": "EC",
        "use": "sig",
        "kid": "LWtFcklLOGdrW",
        "crv": "P-256",
        "x": "X2S1dFE7zokQDST0bfHdlOWxOc8FC1l4_sG1Kwa4l4s",
        "y": "812nU6OCKxgc2ZgSPt_dkXbYldG_smHJi4wXByDHc6g"
      }
    ]
  },
  "iss": "https://op.umu.se",
  "iat": 1601457619,
  "exp": 1601544019
}
```


And the resulting metadata used by the RP could look like:

```
{
  "application_type": "web",
  "client_name": "LIGO Wiki",
  "client_id": "m3GyHw",
  "client_secret_expires_at": 1604049619,
  "client_secret": "cb44eed577f3b5edf3e08362d47a0dc44630b3dc6ea99f7a79205"
  "client_id_issued_at": 1601457619,
  "contacts": [
    "ops@edugain.geant.org",
    "ops@incommon.org",
    "ops@ligo.org"
  ],
  "grant_types": [
    "authorization_code"
  ],
  "id_token_signed_response_alg": "RS256",
  "signed_jwks_uri": "https://wiki.ligo.org/jwks.jose",
  "redirect_uris": [
    "https://wiki.ligo.org/openid/callback"
  ],
  "response_types": [
    "code"
  ],
  "subject_type": "public"
}
```


Appendix B. Notices

Appendix C. Acknowledgements

The authors wish to acknowledge the contributions of the following individuals and organizations to this specification: Michele D'Amico, Andrii Deinega, Heather Flanagan, Samuel Gulliksson, Takahiko Kawasaki, Torsten Lodderstedt, Francesco Marino, Roberto Polli, Jouke Roorda, Mischa Sallé, Marcos Sanz, Amir Sharif, Giada Sciarretta, Peter Schober, Michael Schwartz, Kristina Yasuda, and the JRA3T3 task force of GEANT4-2.

Appendix D. Document History

[[To be removed from the final specification]]

-29

- Fixed #1921: brand new JWS Header parameter, `trust_chain`.
- Fixed #1879, #1881, #1883, #1885: introductionary text, metadata section, editorials.

-28

- Fixed #1823: Metadata policy clarification text on essential term and sub_set, one_of, superset_of.
- Listing endpoint - added the parameters `trust_marked` and `trust_mark_id`.
- Historical keys endpoint, references to rfc7517 and use of claim keys, correction in the normative text.
- Added the endpoint `federation_trust_mark_list_endpoint` for Trust Mark issuers.

-27

- Fixed #1756: Clarified that the authority_hints in the Entity Statement of an explicit registration response is single-valued.
- Added text about the usage of metadata besides metadata_policy.
- Fixed #1745 #1746: Non normative example of Trust Mark status endpoint.
- Fixed #1747: Client Registration Response content-type set to application/entity-statement+jwt.
- Fixed #1740: Metadata policy example for OIDC, removed scopes.
- Fixed #1755: clarification on the key to be used for Federation operations.
- The "essential" policy operator can be used in conjunction with one_of, subset_of, superset_of to make their presence optional.
- Fixed #1779: Entity Type as defined term.
- Added Sequence Diagram representing a Federation Entity Discovery and metadata evaluation.
- Fixed #1757: Federation Historical Keys endpoint support for revocation status. Endpoint enabled for all the Federation Entities.
- Fixed #1803: Subordinate is a defined term and other small editorial changes after Heather's revision.
- Added Cryptographic Trust Mechanism description.
- Fixed #1660: Clarified that Explicit Registration uses no parameters and added Explicit Registration example.
- Fixed #1782: Reference protected resource metadata draft.
- Fixed #1809: Replaced use of "trust issuers" terminology.
- Fixed #1661: Tightened descriptions of metadata standards used.
- Fixed #1801: Better described function of "essential" metadata operator.

-26

- Applied editorial improvements by Heather Flanagan.

-25

- Fixed #1684: disambiguation on the role federation_entity.
- Fixed #1703: Unsigned error response.
- Fixed #1693: Trust mark status endpoint HTTP method changed to POST.
- Fixed #1681: RS256 is not mandatory anymore for federation operations signatures.
- Fixed #1701: trust_chain parameter with PAR and redirect_uri.
- Fixed #1695: entity_type url paramenter for Listing endpoint.
- Fixed #1712: federation_entity claims: organization_name is recommended, logo_uri is added.
- Fixed #1702: removal of metadata type trust_mark_issuer, federation_status_endpoint moved in the federation_entity metadata and renamed to federation_trust_mark_status_endpoint.
- Fixed #1716: Clarified how to retrieve the RP's Entity Configuration when using Automatic Registration.
- Fixed #1656: Introductionary text review.

-24

- Fixed #1654: Text cleanup on how and who publishes the Entity Configuration.
- Relaxed JWK Set representations: jwks, signed_jwks_uri and jwks_uri can coexist in the same Metadata for interoperability purposes across different Federations.
- Fixed #1641: Federation Historical Keys endpoint.
- Fixed #1673: added resolve endpoint JWT claims.
- Fixed #1663: clarifications on the http status codes returning from /.well-known/openid-federation.
- Fixed #1667: serialization format for federation endpoint made explicit.
- Fixed #1662: ID Token section removed.
- Fixed #1680: removal of the claim operation in Generic Errors.
- Fixed #1669: error types in the section Generic Error Response.
- Added Vladimir Dzhuvinov as an editor.

-23

- Text on max_path_length
- Fixed #1634: Trust Chain explanatory text.
- Federation Entity Discovery as defined term.
- Fixed #1645: Federation Entity Keys as defined term.
- Fixed #1633: Explaination text about who signs the Trust Mark and how.
- Fixed #1650: typo in signed_jws_uri and jwks metadata claims.

-22

- Rewritten text about metadata processing when doing Explicit Client Registration.
- Fixed #1581: OP metadata claims enabled for AS.
- Fixed #1594: self-issued trust_chain in the Authorization Request.
- Fixed #1583: metadata policy one_of operator, explanatory text for multiple values.
- Fixed #1584: Stated that domain name constraints are as specified in Section 4.2.1.10 of [RFC5280].

- Added more text on considerations when using the resolve endpoint.
- Removal of aud parameter in the fetch endpoint request.
- General rewording with the terms Leaf Entity and Entity Configuration.
- Fixed #1588: Explicitly described the differences between Automatic and Explicit Registration.
- Fixed #1606: Described situation in which the requirement for signed requests with Automatic Registration could be relaxed.
- Fixed #1629: authz Request Object, audience explanatory text.
- Fixed #1630: Resolve endpoint response content type.
- Fixed #1608: submission of the Trust Chain in the Explicit Client Registration Request.

-21

- Fixed #1547: Metadata section restructured.
- Fixed #1548: request_authentication_signing_alg_values_supported clarification text about the usage of private_key_jwt and request_object.
- Added a new constraint, allowed_leaf_entity_types.
- Resolve endpoint: Removed iss paramenter in the request and specified the usage of the aud claim in the response.
- Fixed #1513: request_authentication_methods_supported according to IANA OAuth 2.0 AS metadata registered names.
- Fixed #1527: Defined the Trust Mark term and added non-normative examples.
- Added Security Considerations regarding the role of the Trust Marks.
- Editorial review, normative language, and typos.
- Fixed #1535: Removed the sub claim from the non-normative example of the Authorization Request Object.
- Fixed #1528: Added Claims Languages and Scripts section.
- Fixed #1456: Added language about space-delimited string parameters from RFC 6749.

-20

- Updated example for Resolve endpoint.
- Recommended that Key IDs be the JWK Thumbprint of the key.
- Fixed #1502 - Corrected usage of `id_token_signed_response_alg`.
- Fixed #1506 - Referenced RFC 9126 for Pushed Authorization Requests.
- Added Giuseppe De Marco as an editor and removed Samuel Gulliksson as an editor.
- Fixed #1508 - Editorial review.
- Fixed #1505 - Defined that `signed_jwks_uri` is preferred over `jwks_uri` and `jwks`.
- Fixed #1514 - Corrected RP metadata examples to use `id_token_signed_response_alg`.
- Fixed #1512 - Registered the error codes `missing_trust_anchor` and `validation_failed`.
- Also registered the OP Metadata parameters `organization_name`, `request_authentication_methods_supported`, `request_authentication_signing_alg_values_supported`, `signed_jwks_uri`, and `jwks` and the RP Metadata parameters `client_registration_types` (which was previously misregistered as `client_registration_type`), `organization_name`, and `signed_jwks_uri`.
- Defined the term Federation Operator and described redundant retrieval of Trust Anchor keys.

- Fixed #1519 - Corrected instances of `client_registration_type` to `client_registration_types_supported`.
- Fixed #1520 - Renamed `federation_api_endpoint` to `federation_fetch_endpoint`.
- Fixed #1521 - Changed swamid.sunet.se to swamid.se in examples.
- Fixed #1523 - Added `request_parameter_supported` to examples.
- Capitalized defined terms.

-19

- Fixed #1497 - Removed `trust_mark` claim from federation entity metadata.
- Fixed #1446 - Added `trust_chain` and removed `is_leaf`.
- Fixed #1479 - Added `jwks` claim in OP metadata.
- Fixed #1477 - Added `request_authentication_methods_supported`.
- Fixed #1474 - Added the `request_authentication_signing_alg_values_supported` OP metadata parameter.
- Rewrote `trust_marks` definition.

-18

- Added the jwks claim name for OP Metadata.
- Moved from a federation API to separate endpoints per operation.
- Added descriptions of the list, resolve and status federation endpoints.
- Registered the `client_registration_types_supported` and `federation_registration_endpoint` authorization server metadata parameters.
- Registered the `client_registration_type` dynamic client registration parameter.
- Registered and used the `application/entity-statement+jwt` media type.
- Registered the `application/trust-mark+jwt` media type.
- Trust marks: the claim `mark` has been renamed to `logo_uri`.
- New federation endpoint: Resolve Entity Statement.
- New federation endpoint: Trust Mark Status.
- Federation API, Fetch: `iss` parameter is optional.
- Trust Marks: added non-normative examples.
- Expanded Section 8.1: Included Entity configurations.
- Fixed #1373 - More clearly defined the term Entity Statement, both in the Terminology section and in the Entity Statement section.

-17

- Addressed many working group review comments. Changes included adding the Overall Architecture section, the `trust_marks_issuers` claim, and the Setting Up a Federation section.

-16

- Added Security Considerations section on Denial-of-Service attacks.

-15

- Added `signed_jwks_uri`, which had been lost somewhere along the road.

-14

- Rewrote the federation policy section.
- What previously was referred to as `client authentication` in connection with automatic client registration is now changed to be `request authentication`.
- Made a distinction between parameters and claims.
- Corrected the description of the intended behavior when `essential` is absent.

-13

- Added 'trust_marks' as a parameter in the Entity Statement.
- An RP's self-signed Entity Statement MUST have the OP's issuer identifier as the value of 'aud'.
- Added RS256 as default signing algorithm for Entity Statements.
- Specified that the value of 'aud' in the Entity Statement use in automatic client registration MUST have the ASs authorization endpoint URL as value. Also, the 'sub' claim MUST NOT be present.
- Separating the usage of merge and combine as proposed by Vladimir Dzhuvinov in Bitbucket issue #1157.
- An RP doing only explicit client registration are not required to publish and support a `/.well-known/openid-federation`.
- Every key in a JWK Set in an Entity Statement MUST have a kid.

-12

- Made JWK Set OPTIONAL in an Entity Statement returned by OP as response of an explicit client registration
- Renamed Entity type to client registration type.
- Added more text describing client_registration_auth_methods_supported.
- Made "Processing the Authentication Request" into two separate sections: one for Authentication Request and one for Pushed Authorization Request.
- Added example of URLs to some examples in the appendix.
- Changed the automatic client registration example in the appendix to use Request Object instead of a client_assertion.

-11

- Added section on trust marks.
- Clarified private_key_jwt usage in the authentication request.
- Fixed Bitbucket issues #1150 and #1155 by Vladimir Dzhuvinov.
- Fixed some examples to make them syntactically correct.

-10

- Incorporated additional review feedback from Marcos Sanz. The primary change was moving constraints to their own section of the Entity Statement.

-09

- Incorporated review feedback from Marcos Sanz. Major changes were as follows.
- Separated Entity configuration discovery from operations provided by the federation API.
- Defined new authentication error codes.

- Also incorporated review feedback from Michael B. Jones.

-08

- Incorporated review feedback from Michael B. Jones. Major changes were as follows.
- Deleted `sub_is_leaf` Entity Statement since it was redundant.
- Added `client_registration_type` RP registration metadata value and `client_registration_types_supported` OP metadata value.
- Deleted `openid_discovery` metadata type identifier since its purpose is already served by `openid_provider`.
- Entity identifier paths are now included when using the Federation API, enabling use in multi-tenant deployments sharing a common domain name.
- Renamed `sub_is_leaf` to `is_leaf` in the Entity Listings Request operation parameters.
- Added `crit` and `policy_language_crit`, enabling control over which Entity Statement and policy language extensions MUST be understood and processed.
- Renamed `openid_client` to `openid_relying_party`.
- Renamed `oauth_service` to `oauth_authorization_server`.
- Renamed `implicit` registration to `automatic` registration to avoid naming confusion with the implicit grant type.
- Renamed `op` to `operation` to avoid naming confusion with the use of "OP" as an acronym for "OpenID Provider".
- Renamed `url` to `uri` in several identifiers.
- Restored Open Issues appendix.
- Corrected document formatting issues.

-07

- Split metadata into metadata and metadata_policy
- Updated example

-06

- Some rewrites
- Added example of explicit client registration

-05

- A major rewrite.

-04

- Changed client metadata names `scopes` to `rp_scopes` and `claims` to `rp_claims`.
- Added Open Issues appendix.
- Added additional references.
- Editorial improvements.
- Added standard Notices section, which is present in all OpenID specifications.

Authors' Addresses

Roland Hedberg (EDITOR)
independent
Email: roland@catalogix.se

Michael B. Jones
Microsoft
Email: mbj@microsoft.com
URI: https://self-issued.info/

Andreas Åkre Solberg
Sikt
Email: Andreas.Solberg@sikt.no
URI: https://www.linkedin.com/in/andreassolberg/

John Bradley
Yubico
Email: ve7jtb@ve7jtb.com
URI: http://www.thread-safe.com/

Giuseppe De Marco
independent
Email: demarcog83@gmail.com
URI: https://www.linkedin.com/in/giuseppe-de-marco-bb054245/

Vladimir Dzhuvinov
Connect2id
Email: vladimir@connect2id.com
URI: https://twitter.com/dzhuvi