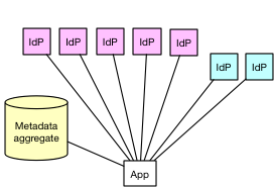
Andreas Solberg
Senior Technical Architect at Uninett

The future of Identity Federations

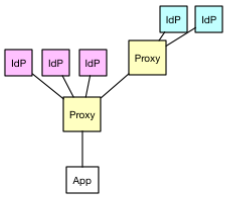
Published on May 24, 2018

Until now a few different approaches has been used to build Identity Federations.

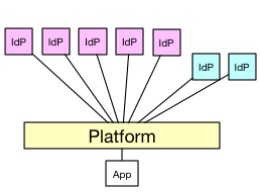
- **Hierarchical routing** and static configuration. eIDAS connects all service providers through a national node.
- **Central SAML metadata aggregate** distributed dynamically. eduGAIN distributes a giant aggregated metadata file for all service providers and identity providers in higher educuation in Europe.
- **Hubs or platforms** for a specific group of service providers hides the complexity from service providers. Dataporten offers a single point of entry for service providers in Norway connecting multiple Identity providers and attribute sources. Also, for specific sectors or environments separate platforms are built, such as ORCID and Elixir AAI.



Central metadata aggregate



Hierarchical routing



Platform

These models **do not scale very well**, and they fail to meet new requirements and expectations. To solve new more advanced use cases we often introduce new hubs. Now as

Andreas Solberg
Senior Technical Architect at Uninett

authentication that dynamically can establish and delegate trust. The future of identity federation also need to cover more than just regular authentication. It is important to build a trust infrastructure that also can be used for authorization and access to data and APIs.

While the timing is perfect now, these ideas are not new. I tried to sketch some ideas for OpenID Connect metadata in 2011, and for dynamic trust in OpenID Connect Federations in 2012. However a lot has changed since then and the OpenID Connect suite of protcols are much more mature and complete.

The entities

A **client** should be able to authenticate end-users by the use of an external provider. The client should be able to authenticate itslef as a client, and it should be able to communicate with an API provider on behalf of the authenticated end-user that is interacting with the client.

An **identity provider** may authenticate an end-user and provide some basic personal attributes.

An **API provider** is offering a backend to third party services, and may not authenticate users themselves but rely on identity providers for that. A client typically wants to interact with the API provider on behalf of an authenticated user, where the identity provider assert the identity of the user.

Basic principles

Both trust and configuration between two entities should be resolved dynamically, rather than prepopulated.

All an entity should need to preconfigure is the trust root. Something like cacerts.

All entities must have a unique globally unique persistent identifier.

Here is a generic pattern for two entities to communicate:

- Given some input discover one or more entities. The important information about an entity is the identifier.
- Given the identifier of the other entity, dynamically obtain configuration of the other party.

Andreas Solberg
Senior Technical Architect at Uninett

Asymmetric keys FTW

The standard OAuth client registration process includes manual registration and a single provider issuing an client id and client secret. This is very convenient for all use cases where there is one single platform, and only one provider, like more or less all cloud providers and most enterprise use cases.

However static client id and secrets becomes a mess for distributed use cases. This approach also does not satisfy the basic principle of all entities have globally unique persistent identifiers.

The current version of OpenID Connect Federation promotes a distributed version of using dynamic client registration.

Luckily the OAuth and **OpenID Connect** protocol **fully support using asymmetric keys** for both the client and the provider.

We've tried to describe this approach in more detail in the *OpenID Connect Federation Asymmetric variant spec*. This work is incomplete, and not updated with all the information in this article. Please help me promote the use of asymmetric keys for clients when building federations with OpenID Connect.

Some of the benefits with using asymmetric keys incudes:

- No need to perform client registration in advance. Configuration and trust can be dynamically established.
- Client identifiers that are globally unique
- Client identifiers that are persistent over time
- Client identifiers that are shared across multiple providers
- No need for handling the complexity of reregistration for credentials that expires. And they need to expire often in order to be able to revoke clients.
- No need for handling the complex state with client id and secret pairs for all providers.
- For the more advanced use cases that we need to solve later on, the possibility of using Proof-of-Possession Key Semantics for JSON Web Tokens is useful.

Basic authentication - the fundamentals

Solving the regular use case with a given client being able to authenticate a user from any

Andreas Solberg
Senior Technical Architect at Uninett

1. Pre-configuration of the client

The client generates a key pair. The public key may look like this:

```
{
  "kty": "RSA",
  "kid": "https://serviceprovider.org/application#1",
  "n": "l7rt1yRvbiOKg8XeP_ICo0yDif-kOLWkUL5FAWKVWhWWAdnN2o1t_otuBX1xLeItE24he4qGH
  "e": "AQAB"
}
```

The client may re-use this key-pair with all providers. The client will choose a *client_id* that includes the hostname of the client application. The hostname used in the *client_id* will be used for discovery by the OpenID Connect Provider.

In example the client can use a *client_id* like this:

```
https://serviceprovider.org/application
```

2. End-user discovery

First the end users navigates to the application (hereby refered to as client), and clicks the login button. In order for the client to know where to send the user for authentication we need some input from the user.

OpenID Connect Discovery defines a protocol to obtain the identifier of the provider from a user hint such as the user e-mail. What is great about this is the completely distributed nature using WebFinger.

> **OpenID Connect Discovery 1.0 incorporating errata set 1**
> http://openid.net
> OpenID Connect 1.0 is a simple identity layer on top of the OAuth 2.0 protocol. It enables Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information...

It is likely that for use cases within higher education, one would like to still rely on Discovery Services presenting the user with lists of providers. What is missing though is a protocol for this to support the OpenID Connect suite. For SAML we have Identity Provider Discovery Service Protocol and Profile.

Andreas Solberg
Senior Technical Architect at Uninett

provider.

The provider configuration will be extended with metadata statements as defined in OpenID Connect Federation section 3.

The provider should sign its own metadata/configuration using its own key pair. The metadata could contain metadata statements from trusted third parties, such as Feide, eduGAIN or eIDAS. The metadata may also contain URLs to assertions from third party as an alternative to embedded ones. The client may also obtain assertions about providers from other sources.

Signed metadata statements can contain limitations on what the provider can be trusted for. The signed metadata statements MUST contain the public key of the provider using Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs).

4. Client sends an authentication request

The client MUST send a signed authentication request as described in OpenID Connect Core section 6. It MUST sign the request using the same key as described in section 1 in this document.

The client does not need to be pre-registered at the provider. The request contains the client_id of the client, which will allow the provider to dynamically obtain metadata for the client. A request may look like this:

```
GET /authorize?request=eyJhbGciOiJSUzI1NiIsImtpZCI6ImsyYmRjIn0.ew0KICJpc3Mi...
  &response_type=code
  &client_id=https%3A%2F%2Fserviceprovider.org%2Fapplication
  &redirect_uri=https%3A%2F%serviceprovider.org/application%2Fcallback HTTP/1.1
Host: provider.org
```

5. Provider dynamically discovers configuration of the client

If the provider has no preconfigured trust with the incoming request *client_id*, and if the *client_id* is a valid URL, and if the provider support OpenID Connect Federation it will proceed with discovery of the client.

By the simple use of WebFinger the provider will get the metadata / configuration of the client. The metadata SHOULD be signed using the client key pair. The metadata may contain metadata statements or references to such from third parties.

In essence this is more or less the same as the provider discovers and trusts the client.

end user and send the code back to the client (authorization code flow).

7. Client sends authenticated request to the token endpoint

In order for the client to obtain the access token at the token endpoint, the client would need to authenticate using its key pair rather than using basic authentication with id/secret pair – as is the most common approach. The client MUST authenticate the request by including the *private_key_jwt* parameter described in OpenID Connect Core Section 9

The *aud* parameter of the generated JWT MUST equal the OP *issuer*.

Here is an example:

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

grant_type=authorization_code&
  code=791222e0-bd67-40f9-8c41-4dd65a9ea33d&
  client_id=https%3A%2F%2Fserviceprovider.org%2Fapplication&
  client_assertion_type=
  urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-beare
  client_assertion=PHNhbWxwOl ... ZT
```

The client will then obtain the access token as in a normal OAuth flow.

More advanced use cases

Next we need to work on more advanced use cases. Here we are missing some pieces. I'll get back to it later.

One important use case is as follow:

> *API provider X would like to offer some personal information to authenticated clients that are tagged in a certain way through the eduGAIN trusted third party (federation), but it requires the end-users to be authenticated using eID (eIDAS).*

The solution will build on top of the basic authentication fundamentals, but require more building blocks.

OAuth 2.0 Token Exchange will be an essential part of it.

And when this is done, it is probably the time for someone to build a new UMA. UMA Connect.

References

Tom Barton from University of Chicago and Internet2 wrote this great article that inspired me to write this update on the use of OpenID Connect in federations

Federation 2.0, Tom Barton

There is ongoing work on OpenID Connect Federation. There is a GEANT project implementing the specification in several languages.

OpenID Connect Federation

Andreas Solberg

Andreas Solberg
Senior Technical Architect at Uninett

28 4 5

More from Andreas Solberg See all 5 articles

Dataporten for Developers
Andreas Solberg on LinkedIn

Office365 Authentication in Higher education in Norway
Andreas Solberg on LinkedIn

Adopting OpenID Connect in Web Applications
Andreas Solberg on LinkedIn

Dataporten.n Platform for e
Andreas Solberg