deutsches forschungsnetz

Einführung in OpenID Connect

DFN-AAI Workshop, 25. Mai 2023

Silke Meyer (smeyer@dfn.de)

Überblick

- OAuth 2.0
- OIDC 1.0
- kurze Pause nach ca. 1 Std.
- SAML vs. OIDC
- OpenID Connect Federation (Draft)
- Informationen zur schrittweisen Einführung von OIDC in die DFN-AAI

OAuth 2.0

OAuth 2.0 - Eckdaten

- The OAuth 2.0 Authorization Framework (RFC 6749)
- 2010: OAuth 1.0, 2012: OAuth 2.0
- delegierte Autorisierung ohne Preisgabe des Passwortes:
  - Informationen mit ausgewählten Dritten (z.B. Websites) teilen
  - Dritte im eigenen Namen handeln lassen
  - API-Autorisierung / native Apps

OAuth 2.0 Terminologie

Resource Owner	Endnutzer*in (SAML 2.0: Principal)
Client	Server/Anwendung, die im Namen des Resource Owners handelt
Authorization Server	Authentisierung und Ausstellen von Tokens an 2 Endpunkten: • Authorization Endpoint • Token Endpoint
Resource Server	Server, der die geschützte Ressource enthält
Redirect URI, auch: Callback	URI des Clients, an die der Authorization Server antworten soll
Authorization Grant	Beweis über erfolgreiche Authentisierung holt sich der Client vom Auth. Server
Access Token	Beweis über Autorisierung, den der Client beim Resource Server vorzeigt, i.d.R. Bearer Token
Claims	Attribute
Scope	Thematische Bündelung von Claims für differenzierten Zugriff auf Nutzerdaten

Bearer Tokens

- OAuth 2.0 Access Tokens meist als Bearer Tokens implementiert
- Security Token nach RFC 6750 „Bearer Token Usage“
- Resource Server muss keine Authentifizierung implementieren, sondern die in dem Token abstrahierte Autorisierung verstehen
- Bearer = Träger → Wer Bearer Token besitzt, kann ihn ohne Identitätsprüfung benutzen.
- immer über TLS-verschlüsselte Verbindungen
- Validierung von Gültigkeit und Scope durch Resource Server beim Authorization Server
- meist Übertragung im Authorization Header Field:

```
GET /resource HTTP/1.1
Host: server.example.com
Authorization: Bearer mF_9.B5f-4.1JqM
```

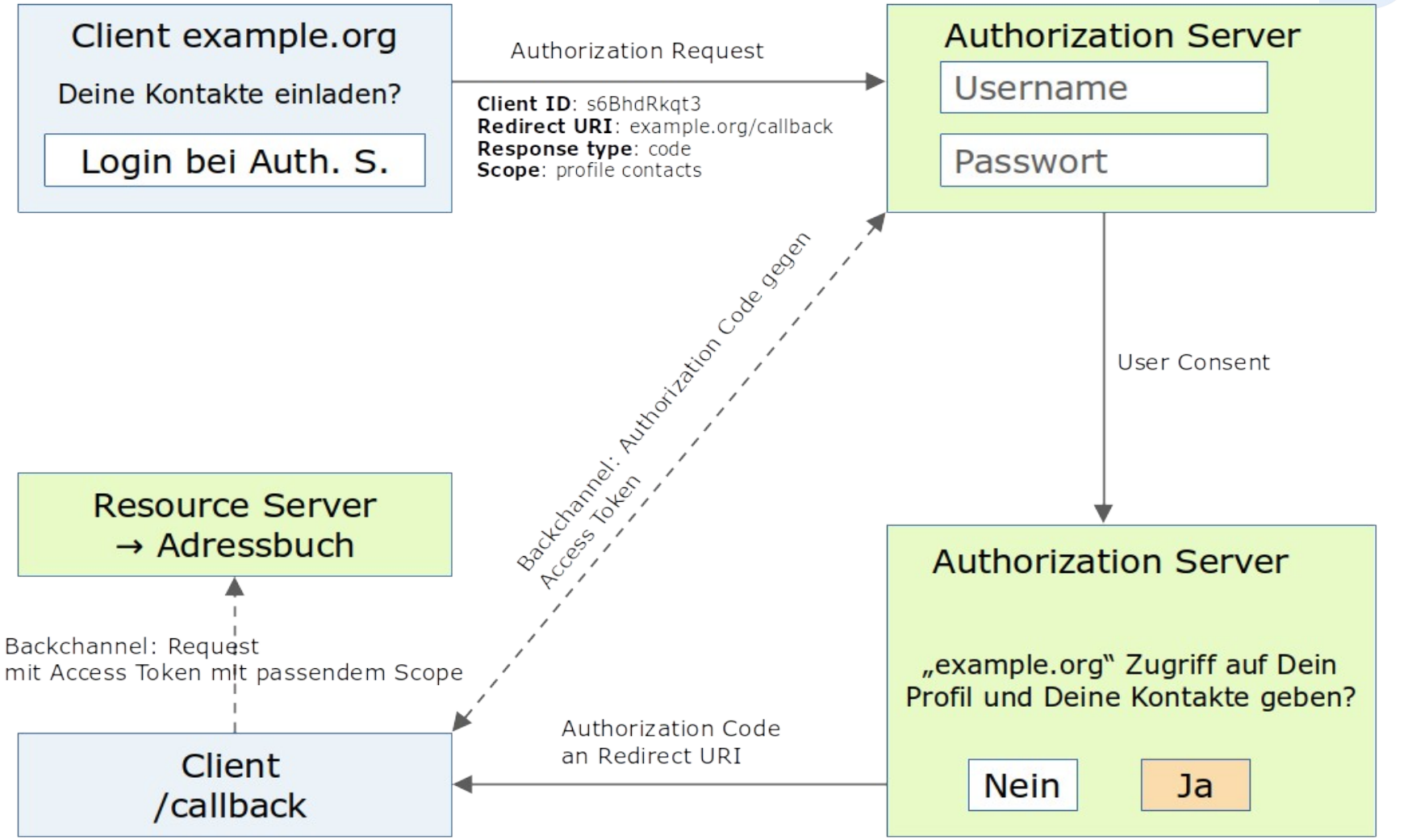
Client-Registrierung in OAuth 2.0

- Woher kennen sich Client und Authorization Server?
- RFC 6749 (2012): Client-Registrierung ist nicht Teil der Spezifikation.
- klassisch: manueller Vorab-Austausch von Client ID, Client Secret, Callback URI (vergleichbar mit WAYFless URLs → SP spricht nur mit einem IdP)
- 2014: OpenID Connect Dynamic Client Registration 1.0
- 2015: OAuth 2.0 Dynamic Registration (RFC 7591) → Erweiterung von OAuth 2.0

Client-Registrierung in OAuth 2.0/OIDC 1.0

- Dynamische Client-Registrierung
  - Authorization Server bietet Registration Endpoint an
  - Client sendet Registration Request mit Metadaten
  - Authorization Server speichert Client-Metadaten
  - Authorization Server weist Client ID und ggf. Client Secret zu

OAuth 2.0 Authorization Code Flow



Quelle: Nate Barbettini, Okta https://www.youtube.com/watch?v=996OiexHze0

OAuth 2.0 Authorization Code Flow

- Authorization Request (User Agent):

```
GET /authorize?response_type=code&client_id=s6BhdRkqt3&state=xyz
&redirect_uri=https%3A%2F%2Fexample%2Eorg%2Fcallback HTTP/1.1
Host: beispiel.org
```

- Authorization Response mit Grant an Redirection URI (User Agent):

```
HTTP/1.1 302 Found
Location: https://example.org/callback?code=SplxlOBeZQQYbYS6WxSbIA
&state=xyz
```

OAuth 2.0 Authorization Code Flow

- Access Token Request (Backchannel):

```
POST /token HTTP/1.1
Host: beispiel.org
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded
client_id=s6BhdRkqt3
&grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fexample%2Eorg%2Fcallback
```

- Basic Authentication am Token Endpunkt mit Client Secret (nach RFC 2617 HTTP Authentication: Basic and Digest Access Authentication)

OAuth 2.0 Authorization Code Flow

- Access Token Response (Backchannel):

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

{
    "access_token":"2YotnFZFEjr1zCsicMWpAA",
    "token_type":"Bearer",
    "expires_in":3600,
    "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
}
```

OAuth 2.0 Authorization Code Flow

- Zugriff auf geschützte Ressource (Backchannel):

```
GET /resource/1 HTTP/1.1
Host: example.org
Authorization: Bearer mF_9.B5f-4.1JqM
```

OAuth 2.0 Zusammenfassung

- OAuth 2.0 wurde für *Autorisierung* spezifiziert
- häufig so verwendet, als ginge es um Authentifizierung
- um die Identitäten der Resource Owner geht es aber genau genommen gar nicht
- OIDC 1.0 ergänzt OAuth 2.0 um diese Informationen

DFN

OpenID Connect 1.0

OpenID Connect Core 1.0

- Spezifikation durch OpenID Foundation 2014
- Nachfolge von OpenID (und nicht damit zu verwechseln)
- Authentifizierung in Form von **ID Token**, aufsetzend auf OAuth 2.0
- drei mögliche Flows (Authorization Code, Implicit, Hybrid), hier Vorstellung des Authorization Code Flows

OIDC: zusätzliche Terminologie

OpenID Connect Provider (OP)	Authentication und Authorization Server (~ IdP)
OpenID Connect Relying Party (RP)	Client, der Identitätsinformationen abrufen möchte (~ SP)
Authentication Request	OAuth 2.0 Authorization Request, der um einen OIDC-spezifischen Scope erweitert ist, damit ein OP (als Authorization Server) eine*n Endnutzer*in gegenüber einer RP (als Client) authentifiziert
ID Token	Token, der Informationen über die Identität des Resource Owners enthält, Kernstück von OIDC 1.0
UserInfo Endpoint	zusätzlicher Endpunkt am Authorization Server, Abfrage von Informationen über Endnutzer*in

JSON Web Tokens

- RFC 7519 (JWT, häufig: „jot“)
- können mit JSON Web Signature (JWS, RFC 7515) signiert sein
- Stateless Sessions statt klass. Session ID
  - kein serverseitiges Speichern von Session-Infos nötig
  - Client-Anfragen bringen nötige Informationen mit
  - kurze Gültigkeitsdauer, ggf. kombiniert mit revoked Tokens-DB

JSON Web Token-Beispiel

- Header: gewählter Signatur-Algorithmus und Typ „JWT“

```
{"typ":"JWT", "alg":"HS256"}
```

- Payload: Standard- und ggf. Custom-Claims

```
{"sub": "johndoe", "iss":"op", "exp":1300819380}
```

- Signatur zur Token-Validierung (JWS): Header und Payload Base64url encoded, durch Punkt getrennt aneinander gehängt, shared secret oder public/private key)

```
HMAC_SHA256(
  secret,
  base64urlEncoding(header) + '.' +
  base64urlEncoding(payload)
  )
```

- Resultat (gekürzt): eyJ0eXAiOiJKiJIUzI1NiJ9.EyJpc3MiOiJA4MTkzOD.dBjft4CVP-mhb1p1r_wWOEjXk

ID Tokens

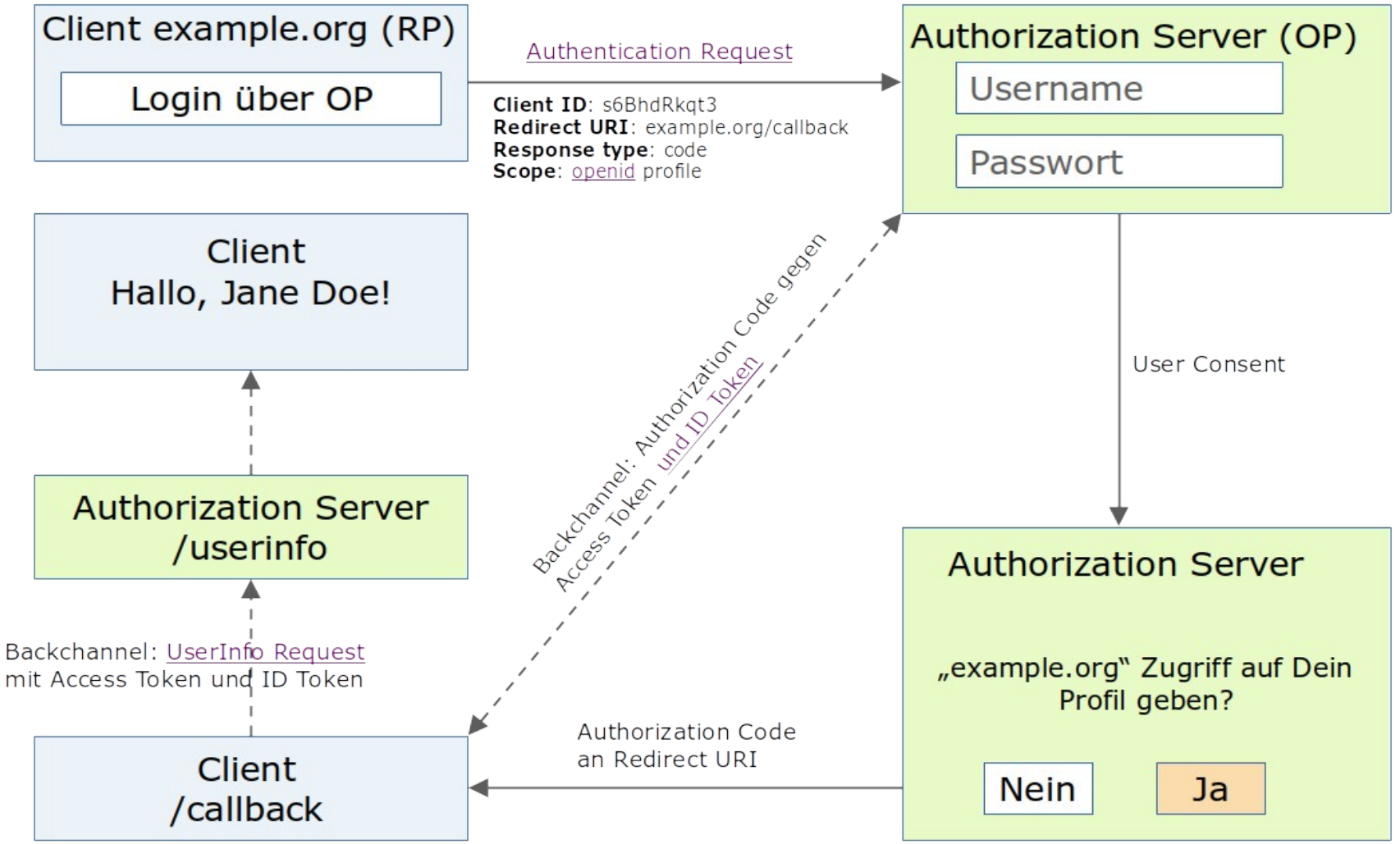
- sind JSON Web Tokens
- Herausgabe durch OP zusätzlich zu Access Token und ggf. Refresh Token
- Inhalt: Claims (Infos über Nutzer*in/Resource Owner)
- Übermittlung an Client im Authorization-Feld des HTTP-Headers, kein Caching erlaubt
- mit JSON Web Signature signiert, optional dann noch verschlüsselt
- Client prüft bei Erhalt:Empfänger, Aussteller, ggf. Entschlüsselbarkeit, Algorithmus, alle Zeitstempel, Nonce

ID Token Response im HTTP-Header (gekürzt)

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

  {
      "access_token": "SlAV32hkKG",
      "token_type": "Bearer",
      "refresh_token": "8xLOxBtZp8",
      "expires_in": 3600,
      "id_token": "eyJ0eXAiOiJKiJIUzI1NiJ9.EyJpc3MiOiJA4MTkzOD.dBjft4CVP-mhb1p1r_wWOEjXk"
  }
```

OIDC 1.0 Authorization Code Flow



Quelle: Nate Barbettini, Okta https://www.youtube.com/watch?v=996OiexHze0

OIDC 1.0 Authorization Code Flow

- Authentication Request (User Agent):

```
GET /authorize?response_type=code&client_id=s6BhdRkqt3&state=xyz&scope=openid
%20profile&redirect_uri=https%3A%2F%2Fexample%2Eorg%2Fcallback HTTP/1.1
Host: op.beispiel.org
```

- Authentication Response = OAuth 2.0 Authorization Response
- Access Token Request = OAuth 2.0 Access Token Request
- Validierung des Token Request durch OP: analog zu OAuth 2.0 plus Prüfung, ob der Authorization Code als Antwort auf einen OIDC Authentication Request ausgestellt wurde. Nur dann Ausgabe eines ID Tokens.

OIDC 1.0 Authorization Code Flow

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

  {
      "access_token": "SlAV32hkKG",
      "token_type": "Bearer",
      "refresh_token": "8xLOxBtZp8",
      "expires_in": 3600,
      "id_token": "eyJ0eXAiOiJKiJIUzI1NiJ9.EyJpc3MiOiJA4MTkzOD.dBjft4CVP-mhb1p1r_wWOEjXk"
  }
```

OIDC 1.0 Authorization Code Flow

- ID Token-Validierung durch RP:
  - Entschlüsselbarkeit mit Key und Alg. aus Client Registrierung
  - Abgleich von Aussteller und Empfänger
  - Zeitstempel-Prüfung, z.B. Ausstellungszeitpunkt und Ablaufdatum des ID Tokens, Speicherdauer des Nonce
  - optionales Nonce muss mit Nonce aus AuthnReq übereinstimmen

OIDC 1.0 Authorization Code Flow

- UserInfo Request: Client-Anfrage am UserInfo-Endpunkt des OP mit Access Token

```
GET /userinfo HTTP/1.1
Host: op.beispiel.de
Authorization: Bearer SlAV32hkKG
```

- UserInfo Response: JSON Objekt im Body der HTTP Response

```
HTTP/1.1 200 OK
Content-Type: application/json
{
    "sub": "248289761001",
    "name": "Jane Doe",
    "given_name": "Jane",
    "family_name": "Doe"
}
```

OIDC 1.0 Authorization Code Flow

- ähnliche Prüfungen wie oben durch Client und zusätzlich:
  - Ist ein Sub Claim in der UserInfo Response?
  - Stimmt der Sub Claim überein mit Sub Claim im ID Token?

SAML versus OIDC

SAML 2.0 und OIDC 1.0 im Vergleich

SAML 2.0	OIDC 1.0
Browser	Browser, native Apps, Geräteauthentifizierung, API
Authentication und Authorization	OAuth 2.0: Autorisierung OIDC 1.0: Identity Layer zur Authentifizierung
xml-basiert	JSON-basiert
recht verbos → höhere Transaktionskosten	kleine Payload → geringe Transaktionskosten
HTTP / SOAP	HTTP / REST-API Kommunikation
Vertrauen über Austausch von xml-Metadaten (1:1 oder Föderationsmetadaten)	Vertrauen über Client ID, Client Secret + Callback URI oder dyn. Client-Registrierung, momentan nur 1:1

SAML Assertion

```
2021-09-08 13:36:22,449 - 127.0.0.2 - DEBUG [org.opensaml.saml.saml2.encryption.Encrypter:339] - Assertion before encryption:
<?xml version="1.0" encoding="UTF-8"?><saml2:Assertion xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion" ID="_cf4228f862a4b997bbbd7b5c3b395299" IssueInstant="2021-09-08T11:36:21.671Z" Version="2.0">
   <saml2:Issuer>https://idp.local/idp/shibboleth</saml2:Issuer>
   <saml2:Subject>
      <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent" NameQualifier="https://idp.local/idp/shibboleth" SPNameQualifier="https://sp1.local/shibboleth"
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">ZrUq2jdeUVo5sO9Z3WnuwKEUGTw=</saml2:NameID>
      <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
         <saml2:SubjectConfirmationData Address="127.0.0.2" InResponseTo="_1565b5838dd009ff423216c6fc6646c7" NotOnOrAfter="2021-09-08T11:41:21.902Z" Recipient="https://sp1.local/Shibboleth.sso/SAML2/POST"/>
      </saml2:SubjectConfirmation>
   </saml2:Subject>
   <saml2:Conditions NotBefore="2021-09-08T11:36:21.671Z" NotOnOrAfter="2021-09-08T11:41:21.671Z">
      <saml2:AudienceRestriction>
         <saml2:Audience>https://sp1.local/shibboleth</saml2:Audience>
      </saml2:AudienceRestriction>
   </saml2:Conditions>
   <saml2:AuthnStatement AuthnInstant="2021-09-08T11:36:14.437Z" SessionIndex="_e30db63df76c093a938fa393916fb621">
      <saml2:SubjectLocality Address="127.0.0.2"/>
      <saml2:AuthnContext>
         <saml2:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport</saml2:AuthnContextClassRef>
      </saml2:AuthnContext>
   </saml2:AuthnStatement>
   <saml2:AttributeStatement>
      <saml2:Attribute FriendlyName="eduPersonScopedAffiliation" Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.9" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
         <saml2:AttributeValue>member@local</saml2:AttributeValue>
         <saml2:AttributeValue>employee@local</saml2:AttributeValue>
         <saml2:AttributeValue>staff@local</saml2:AttributeValue>
      </saml2:Attribute>
      <saml2:Attribute FriendlyName="eduPersonEntitlement" Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.7" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
         <saml2:AttributeValue>urn:mace:dir:entitlement:common-lib-terms</saml2:AttributeValue>
      </saml2:Attribute>
   </saml2:AttributeStatement>
</saml2:Assertion>
```

OIDC Token Response

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

  {
    "access_token": "SlAV32hkKG",
    "token_type": "Bearer",
    "refresh_token": "8xLOxBtZp8",
    "expires_in": 3600,
    "id_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ.ewogImlzc
      yI6ICJodHRwOi8vc2VydmVyLmV4YW1wbGUuY29tIiwKICJzdWIiOiAiMjQ4Mjg5
      NzYxMDAxIiwKICJhdWQiOiAiczZCaGRSa3F0MyIsCiAibm9uY2UiOiAibi0wUzZ
      fV3pBMk1qIiwKICJleHAiOiAxMzExMjgxOTcwLAogImlhdCI6IDEzMTEyODA5Nz
      AkfQ.ggW8hZ1EuVLuxNuuIJKX_V8a_OMXzR0EHR9R6jgdqrOOF4daGU96Sr_P6q
      Jp6IcmD3HP99Obi1PRs-cwh3LO-p146waJ8IhehcwL7F09JdijmBqkvPeB2T9CJ
      NqeGpe-gccMg4vfKjkM8FcGvnzZUN4_KSP0aAp1tOJ1zZwgjxqGByKHiOtX7Tpd
      QyHE5lcMiKPXfEIQILVq0pc_E2DzL7emopWoaoZTF_m0_N0YzFC6g6EJbOEoRoS
      K5hoDalrcvRYLSrQAZZKflyuVCyixEoV9GfNQC3_osjzw2PAithfubEEBLuVVk4
      XUVrWOLrLl0nx7RkKU8NXNHq-rvKMzqg"
  }
```

Open ID Connect Federation (Draft)

Vertrauen in multilateralen Föderationen

- Vermittlung durch dritte Stelle
- Sicherstellung:
  - Die Gegenstelle folgt denselben Regeln.
  - Die Gegenstelle gehört derselben Föderation an (national, internat.).
  - Die Informationen, die die Gegenstelle während der Kommunikation über sich selbst schickt, wurden nicht manipuliert.
- OpenID Connect Federation: Spezifikation im Entwurfsstadium (Draft 29, Stand Mai 2023)

Das grobe Prinzip

- Trust-Chain-Validierung
  - Jede Entität stellt ihre eigene Konfiguration bereit (eigene „Metadaten“).
  - Darin u.a.: Info, welche übergeordnete Entitäten die eigenen Informationen bestätigen können
  - Verkettung von Signaturen kann hoch bis zum Föderationsbetreiber oder bis zum Betreiber einer Interföderation (eduGAIN) verfolgt und validiert werden
- Metadata Policy-Berechnung

OIDC Federation: Terminologie (Ausschnitt)

Trust Anchor	vertrauenswürdiger Dritter, z.B. vom Föderationsbetreiber betriebenes System, das Informationen über teilnehmende Org./Systeme verteilt, z.B. Signing Keys
Entity Configuration	selbst ausgestelltes Entity Statement einer Entität, inkl. Signing Keys u. a. Infos, mit denen die Trust Chain überprüft werden kann
Entity Statement	**signierter** JWT mit festgelegten Claims über eine Entität
Leaf Entity 	Teilnehmende Entität (OP oder RP)
Trust Chain	Abfolge von Entity Statements von Leaf Entity bis Trust Anchor
Trust Marks	signierte JWTs, die ausdrücken, dass eine Entität ein bestimmtes Set von Anforderungen erfüllt (vergleichbar mit Entity Categories)

Funktionsweise I

- Entitäten stellen ihre Konfiguration bereit: `/.well-known/openid-federation`
- Konfig enthält u.a. Angaben zu übergeordneten Entitäten („Authority Hints“)
- Trust Anchors und Intermediates stellen Entity Statements über untergeordnete Entitäten aus.
- verwendetes Schlüsselmaterial: JSON Web Keys (JWK, RFC 7517), selbst ausgestellte öffentliche Schlüssel

Funktionsweise II: Endpunkte

Endpunkt	Anbieter	Zweck
Fetch	übergeordnete Entitäten	Abfrage von Entity Statements über untergeordnete Entitäten
List	übergeordnete Entitäten	Abfrage aller direkt untergeordneten Entitäten
Status	Aussteller von Trust Marks	Abfrage der Aktualität von Trust Marks
Resolve	jede Entität, optional	Abfrage von Metadaten und Trust Marks
`./well-known/openid-federation/historical-jwks`	jede Entität, optional	Veröffentlichung vorheriger öffentlicher Schlüssel

Funktionsweise III: Trust Chain-Auflösung

- Trust Anchor-Schlüsselmaterial ist RP im Vorfeld bekannt
- RP hangelt sich die Kette hoch:
  - Abfrage von Autoritäten aus Entity Configuration des OP
  - Abfrage der Entity Configuration der Autorität → Fetch-Endpunkt
  - Abfrage des signierten Entity Statements der Autorität über den OP
  - Mehrstufigkeit möglich (z.B. OP → DFN-AAI → eduGAIN)
- Überprüfung der Kette mit öff. Schlüssel des Trust Anchors

OP Entity Configuration (gekürzt)

```
{
  "iss":"https://op.umu.se",
  "sub":"https://op.umu.se",
  "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"],
      "op_policy_uri":"https://www.umu.se/en/legal-information/",
      "token_endpoint":"https://op.umu.se/openid/token",
      "federation_registration_endpoint":"https://op.umu.se/openid/fedreg"}
    },
    "authority_hints":["https://umu.se"],
    "jwks":{ "keys":[{"e":"AB...", "kid":"dER...", "kty":"RSA", "n":"xqcCs-…"} }
}
```

RP Entity Configuration (gekürzt)

```
{
  "iss": "https://openid.sunet.se",
  "sub": "https://openid.sunet.se",
  "metadata": {
    "openid_relying_party": {
      "application_type": "web",
      "redirect_uris": ["https://openid.sunet.se/rp/callback"],
      "client_registration_types": ["automatic", "explicit"],
      "organization_name": "SUNET",
      "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": "pnXBezb9J_...", "use": "sig" }]},
  "authority_hints": ["https://edugain.org/federation"]
}
```

Funktionsweise IV: Metadata Policies

- Entitäten können Policies für untergeordnete Entitäten kommunizieren
- Beispiele: zu verwendende Kontaktadressen, Grant Types, Authentifizierungsmethoden an Endpunkten, Algorithmen zur Token-Signierung, Standardwerte
- Zur Auflösung der Metadaten einer Entität gehört nach der Validierung der Trust Chain die Anwendung gültiger Metadata-Policies
- bei Abweichungen zwischen Ebenen: Kombination nach spezifizierten Verrechnungsregeln

Zum Stand von OpenID Connect in der DFN-AAI

Viele offene Fragen

- OIDC Federation noch im Entwurfsstadium
- bisher keine uns bekannte Open-Source-Implementierung verfügbar
- momentaner Plan: dreistufiges Vorgehen
  1. SAML OIDC-Proxy
  2. Relying Parties in xml-Metadaten
  3. Erweiterung der MDV für OIDC Federation

SAML OIDC-Proxy für die Föderation

- derzeit in Vorbereitung: temporärer SAML OIDC Proxy zur Anbindung von RP, die mit mehreren IdP/OP in der Föderation zusammenarbeiten
- Evaluation der Lösungen läuft
- Inbetriebnahme voraussichtlich im Herbst 2023
- Informationen auf dfn-aai-users Mailingliste

Relying Parties in xml-Metadaten

- Workaround der Shibboleth-Entwickler: Der Shibboleth OP versteht xml-Metadaten (und JSON-Metadaten)
- Aufnahme von OIDC RPs in die normalen Föderationsmetadaten, zu klären: Discovery
  - über Entity Category?
  - über Embedded Discovery Services an den RPs mit freigeschalteten OP?

Fernziel: OIDC Federation über die MDV

- längerfristig: Erweiterung der DFN-AAI Metadatenverwaltung:
  - Teilnehmende stellen OIDC Entity Configurations ein
  - Föderationsbetreiber stellt Entity Statements über teilnehmende OIDC Leaf Entities aus

OIDC Claims

- To do: Empfehlungen für OIDC-Claims bzw. für die Zusammenfassung von Claims in Scopes, die ein OP liefern sollte
- Abstimmung mit internationalem Kontext sinnvoll → REFEDS
- Vorarbeiten gibt es bereits aus Umfeld von OpenID Foundation bzw. AARC-Community → analog zu Attributen aus den Schemata eduPerson und schac

Vielen Dank! Gibt‘s Fragen?

- Kontakt
  - DFN-AAI Team

    E-Mail: hotline@aai.dfn.de
    Tel.: +49-30-884299-9124
    Fax: +49-30-884299-370

    Anschrift:
    DFN-Verein, Geschäftsstelle
    Alexanderplatz 1
    10178 Berlin