# Authentication and Authorization

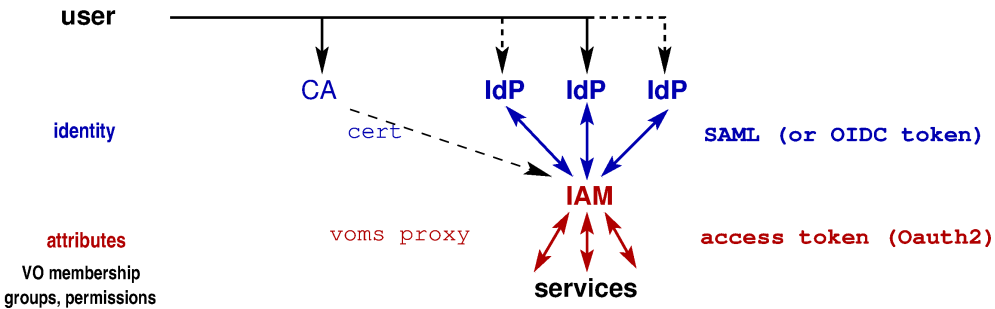

ILDG Middleware Working Group

July, 2025

# Overview

1. IAM and Single Sign-On
2. Access Control
3. Tokens

# Identity and Access Management (IAM) / Single Sign-On (SSO)




- **identity** information provided by trusted Identity Provider (IdP, e.g. home institution) transported via SAML or OIDC token (or X.509 certificate)
- additional user **attributes** provided by IAM (acting also as Attribute Service) e.g. VO membership, group membership, roles and/or permissions transported via access (or ID) tokens (or VOMS proxy certificate)

# AARC Blueprint Architecture

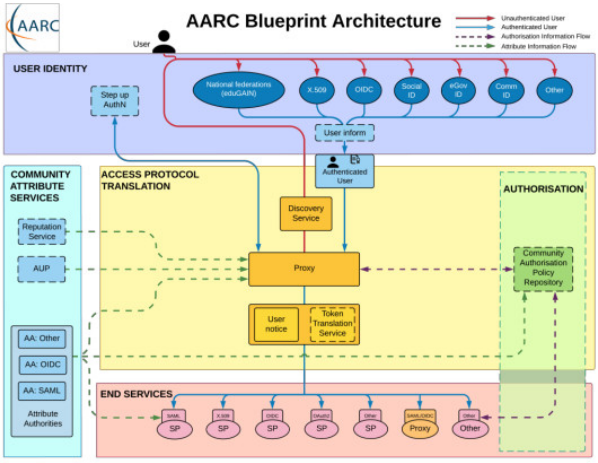


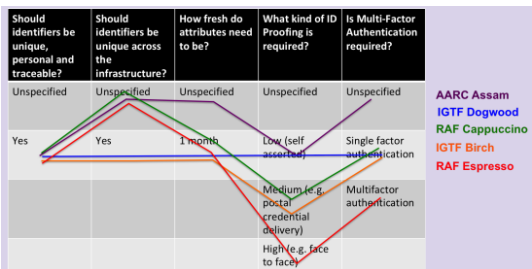

*Figure 1: Component layers of the AARC Blueprint Architecture (AARC-BPA-2019)*

# Mutual Trust Relation 1: IdP ↔ IAM

- IdP (and user) needs to trust IAM before releasing attributes (=personal data)
- ILDG (IAM, services) needs to trust identity vetting of IdP

➜ Federations of IdPs/CAs which can guarantee a well-defined Level of Assurance (LoA)

- CA: IGTF
- IdP: eduGAIN

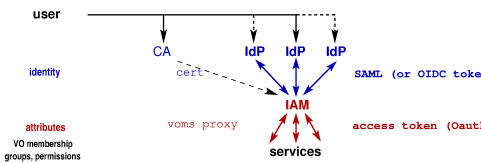



AARC Acceptable Authentication Assurance Policy

# Mutual Trust Relation 2: IAM ↔ SP/RS

Service Providers (SP) / Resource Servers (RS = MDC, FC, SE, . . . ) used by ILDG

- ❒ require (the possibility to) reliably identify users (authentication)
  e.g. for access to expensive resources
  - storage (even for public read-only access!)
  - fast (!) network connections
- ❒ **enforcement** of access restrictions / permissions (authorization)
  must be trusted by resource owners (= you!)
- ❒ **decision** of access policies
  should be controlled by the resource owners



# (Attribute-Based) Access Control Model

**Challenge:** Huge many-to-many relation (for each action: R, W, ...)
between users and resources (files, metadata, ...)

$$\{ \text{ action } \} \times \{ \text{ user } \} \times \{ \text{ resource } \} \longrightarrow \{ \text{ true, false } \}$$


- ❒ **Attributes** of
  - subject (user)
  - action (R/W)
  - object ([meta]data)
  - context
- ❒ Policy **Enforcement** Points → **distributed** Resource Servers (RS = MDC, FC, SE)
- ❒ Policy **Decision** Point → **???**

# Identity-Based Access Control

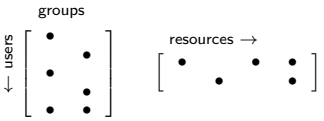
- Attribute: **user identity**
- Policy **decision** point = Policy **enforcement** point



- **Problem:**
  - ✘ GDPR
  - ✘ Consistency / synchronization of policies on each RS
  - ✘ Size

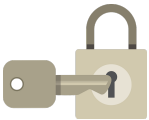
# Group-Based Access Control

- Attribute: **group membership**
- Policy **decision** point = Policy **enforcement** point


- **Problem:**
  - ✔ GDPR
  - ✘ Consistency / synchronization of policies on each RS
  - ✘ Size?

# Capability-Based Access Control

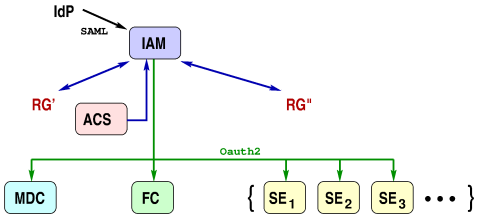


- **Attributes:** capabilities assigned to user (e.g. "scope" claim of token)
  explicit or implicit "Access Control Attributes" of resources
- Policy **enforcement** points → **distributed** Resource Servers (RS)
- Policy **decision** point → **central** Access Control Service (ACS)
- Break-up of huge user-resource relation into smaller many-to-many relations

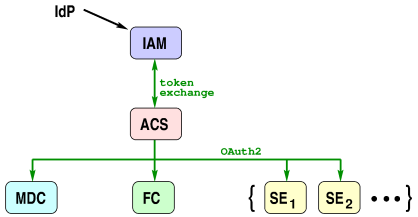

☛ Closely following WLCG specifications

# Access Control Service (ACS)

- 3rd "catalogue" for administrative metadata
  - hierarchical delegation (admin → project → groups/users)
  - optional group management (GDPR concerns!)
- ACS beside/behind of IAM




or in front of IAM




☛ Setup of ILDG is in certain aspects ahead of (and interesting for) other communities

# Representation of “claims” by JSON Web Tokens (JWT)

A JWT consists of several parts (base64 encoded, separated by “.”), e.g. rfc7519

- ❒ cryptographic info
- ❒ payload = claim set (= JSON object made of name-value pairs)
- ❒ cryptographic signature (offline and online verification)

```
echo $BEARER_TOKEN | cut -d .  -f 2 | base64 -d | jq .
```

Registered claim names:

| | |
|---|---|
| `iss` | (Issuer) |
| `sub` | (Subject) |
| `aud` | (Audience) |
| `exp` | (Expiration Time) |
| `iat` | (Issued At) |
| `jti` | (JWT ID) |

Other (public/private) claim names:

- `scope` (authorization info)
- `wlcg.groups` (identity attributes)
- ...

# Use-cases of Tokens

❒ **OpenID Connect:** Identity layer on top of OAuth2 protocol — OIDC Core 1.0

Roles:

- OpenID Provider (OP)
- Relying Party (RP)

ID-Token

- format: always JWT
- claim: "user has been authenticated"

❒ **OAuth2:** Authorization framework — rfc6749

Roles:

- Resource Owner
- Resource Server
- Client
- Authorization Server

Access-Token, Refresh-Token

- format: string, possibly JWT
- claim: "app has been authorized" (expressed through claim name `scope`)

# Authorization Grants: Code Flow

(1) A registered client requests token and receives code from IAM
(2) Resource owner authenticates to IAM and approves request
(3) Client presents code and receives token from IAM

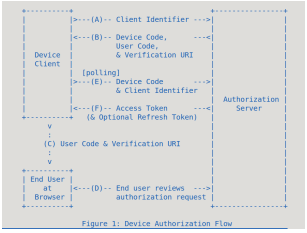


Figure 1: Device Authorization Flow

Two variants of code flow:

- Device Code (client runs on a device without browser) rfc8628
- Authorization Code (using redirect URL) rfc6749

Other authorization flows:

- Refresh Token
- Token Exchange rfc8693
- ...

# Scopes used in ILDG

$$\texttt{storage.stage:/}\langle path\rangle \Rightarrow \texttt{storage.read:/}\langle path\rangle$$
$$\texttt{storage.modify:/}\langle path\rangle \Rightarrow \texttt{storage.create:/}\langle path\rangle$$
$$\texttt{metadata.write:/}\langle path\rangle \Rightarrow \texttt{metadata.read:/}\langle path\rangle$$

**Path matching:** [WLCG Common JWT Profiles 1.0]
Access to a resource with associated path (Access Control Attribute) is allowed or denied depending on $\langle path\rangle$ parameter of scope in token.

e.g. SURL = `https://dcache.somewhere.net:2880/a/b` **/c/d**

OAuth2 token:

| scope (capability) | access |
|---|---|
| `storage.read:/` | permit |
| `storage.read:/c` | permit |
| `storage.read:/c/d` | permit |
| `storage.read:/x` | deny |
| `storage.read:/c/y` | deny |

# Mechanisms to control Scopes in IAM

❒ **Client configuration** IAM documentation

- system scopes (enabled by admin if restricted, or by user)
- custom scopes (enabled by owner, unless interfering with a system scope)

❒ **Scope policies** IAM documentation

- deny (by default)
- permit to specific groups (users)

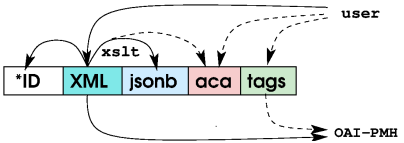

Current IAM setup (will change):

- rely on scope policies
- use dedicated client(s) for which protected (custom) scopes are enabled by admin
  → disadvantage: shared client "secret", no path hierarchy for custom scopes

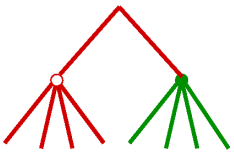
# Identifiers and Access Control

Access Control Attribute is

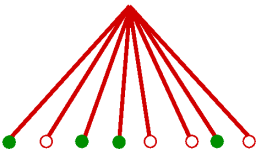
- in SE identical to path (after baseURL)
- in FC derived from SURL (baseURL/baseACA)
- in MDC derived from XML (markovChainURI)




☛ Well chosen hierarchy of identifiers can simplify/complicate handling of embargo situations!



1 policy



4 policies

# Client Registration and Configuration in IAM

- ❒ Main:
  - name (irrelevant)
  - redirect URLs (e.g. for `oidc-agent`)
- ❒ Credentials:
  - authentication method (HTTP basic)
  - client secret
- ❒ Scopes
  - system (only by admin if protected)
  - custom (unless in conflict with system scope)
- ❒ Grant types
  - authorization code
  - device code
  - refresh token
- ❒ [Owners]

# Get (access) token by manual Device-Code Flow

| | | |
|---|---|---|
| (D1) | client requests device code and displays user code<br>(error if any requested scope is not enabled for client) | (curl) |
| (D2) | authenticated user approves request<br>(not neccessarily owner of client) | (browser) |
| (D3) | client presents code and receives access token<br>(scopes are silently removed if denied) | (curl) |

PRO:

- No further configuration steps or setup needed (see script `try-token`)

CON:

- No handling of refresh tokens (possible, but requires security precaution)
- Uses shared client (no customization by normal user)

# Step (D1)

**Request:**

```
curl -s -X POST -d client_id=$CI -d "scope=$OPT_S" $URL1
```

- `CI` = client ID
- `OPT_S` = space-separated list of requested scopes
- `URL1` = https://iam-ildg.cloud.cnaf.infn.it/devicecode

**Response:** JSON object with members

- `device_code`
- `verification_uri`
- `user_code`
- ...

# Step (D3)

**Request:**

```
curl -s -X POST -u $CI:$CS
-d grant_type=urn:ietf:params:oauth:grant-type:device_code
-d device_code=$DC -d audience=$AUD -d "scope=$OPT_S" $URL2
```

- `CS` = client secret
- `DC` = device code from (D1)
- `AUD` = optional audience claim
- `URL2` = https://iam-ildg.cloud.cnaf.infn.it/token

**Response:** JSON object with members

- `access_token`
- `id_token` (if `openid` enabled and requested)
- `refresh_token` (if `offline_access` enabled and requested)
- ...

# Using oidc-agent

Provides book-keeping and encrypted storage of refresh tokens and client credentials

- store client credentials and configuration as "account" ⟨*name*⟩

```
oidc-gen ⟨name⟩ --client-id=1d636a1d-2f8a-41b4-83b6-058ab080af61
      --client-secret=$CLIENT_SECRET --scope="$OPT_S"
      --iss=https://iam-ildg.cloud.cnaf.infn.it/
```

- show account details

```
oidc-gen -p ⟨name⟩
```

- activate specific account

```
oidc-add ⟨name⟩
```

- use cached access token or renew through refresh token

```
oidc-token ⟨name⟩ [-s ⟨scope⟩]
```

# Using Tokens

- Display token

```
echo ⟨token⟩
| awk -F . 'A=$2; while(length(A)%4) A=A "="; print A'
| base64 -d | jq .
```

- Use token with `curl`

```
curl -H "Authorization:  Bearer ⟨token⟩" ...
```

- Use token with `gfal`

```
export BEARER_TOKEN=⟨token⟩
gfal-...
```

# Summary

- INDIGO IAM developed and hosted by INFN-CNAF serves multiple purposes
  - IdP federation through eduGAIN
  - user and group mangement
  - enforcement of AUP and VO policy
  - token issuer
  - client registration
  - (access-)policy engine
  - missing support for hierarchical delegation can be handled by ACS
- Complete transition to tokens has enabled fine-grained access control
  - central policy decision point
  - capability-based and GDPR compliant
  - exploiting advanced features of standard specifications and of IAM
  - ahead of other communities

Many thanks to IAM developers and support team at INFN-CNAF
and to Basavaraja BS (now CTAO)