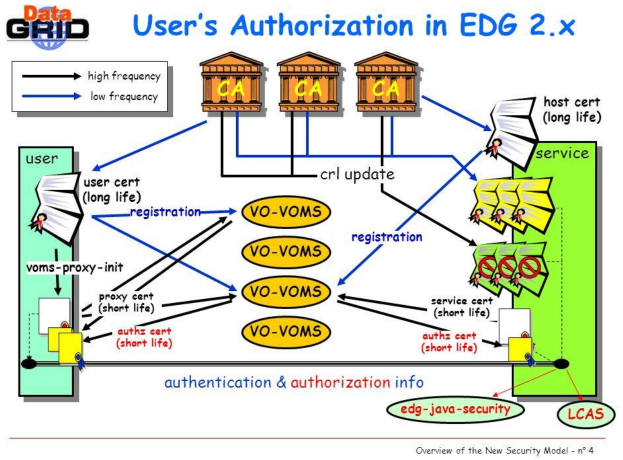
WLCG profile for JWTs

Corso di formazione “Panoramica su OAuth2/OpenID Connect e sue applicazioni tramite il servizio INDIGO IAM”, 12-14 Maggio 2025, LNF

Roberta Miccoli, INFN CNAF

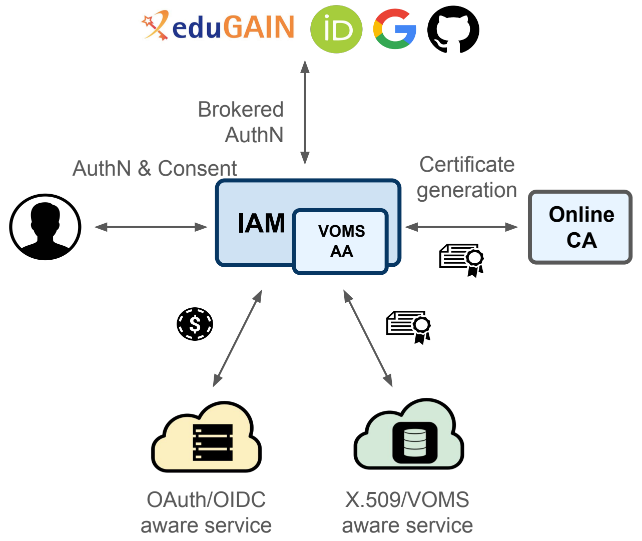
Evolution of the WLCG AAI beyond X.509

Current, X.509-based AAI



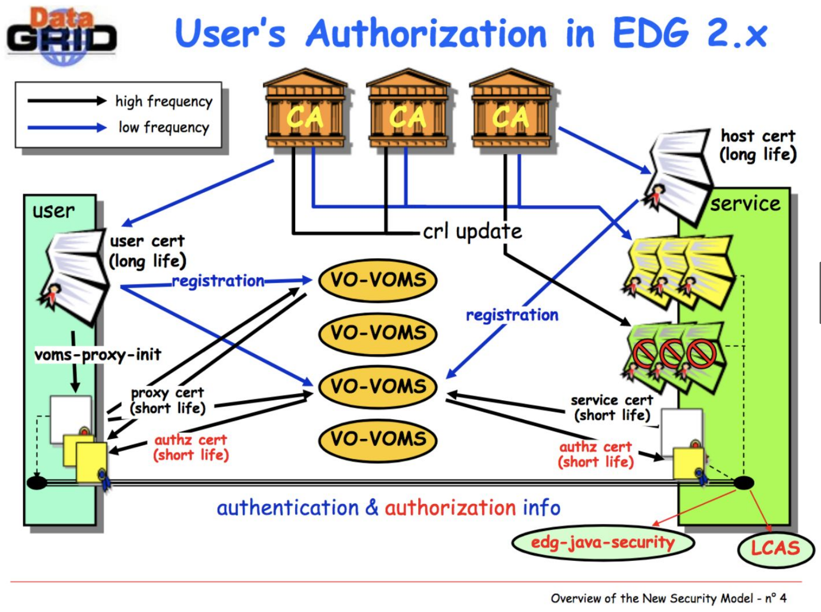
Move beyond X.509

Future, token-based AAI



Approach: leverage and build upon the WLCG experience

Evolution of the WLCG AAI beyond X.509



To access computing and storage resources in the WLCG community, users use a **VOMS proxy**

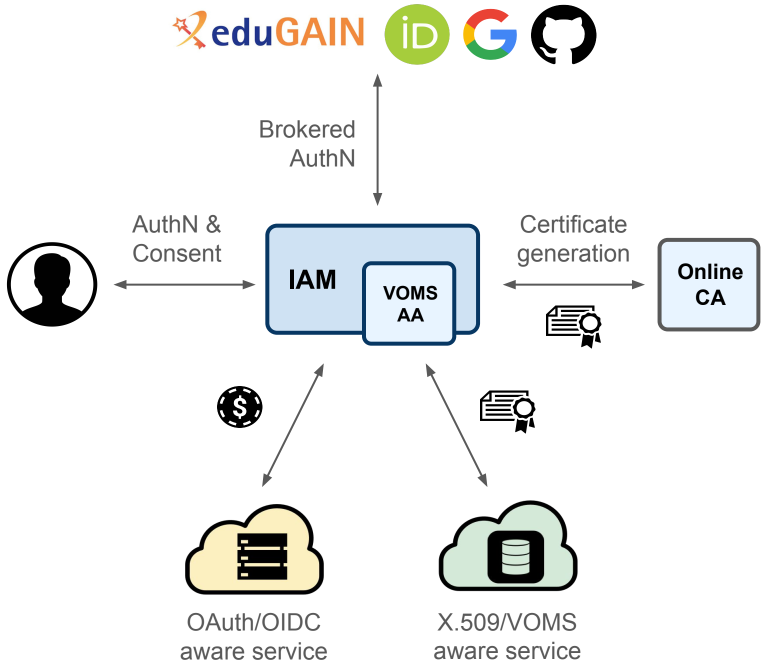
A VOMS proxy provides information about who you are, for which Virtual Organization (VO) you're acting and what you can do on the infrastructure (i.e. VOMS groups and roles)

Evolution of the WLCG AAI beyond X.509

In the near future we will use **tokens**, which will provide similar information

Tokens are obtained from a VO token issuer (e.g. INDIGO IAM) using **OAuth/OpenID Connect** protocol message exchanges (aka flows)

Tokens are sent to services/resources following **OAuth** recommendations

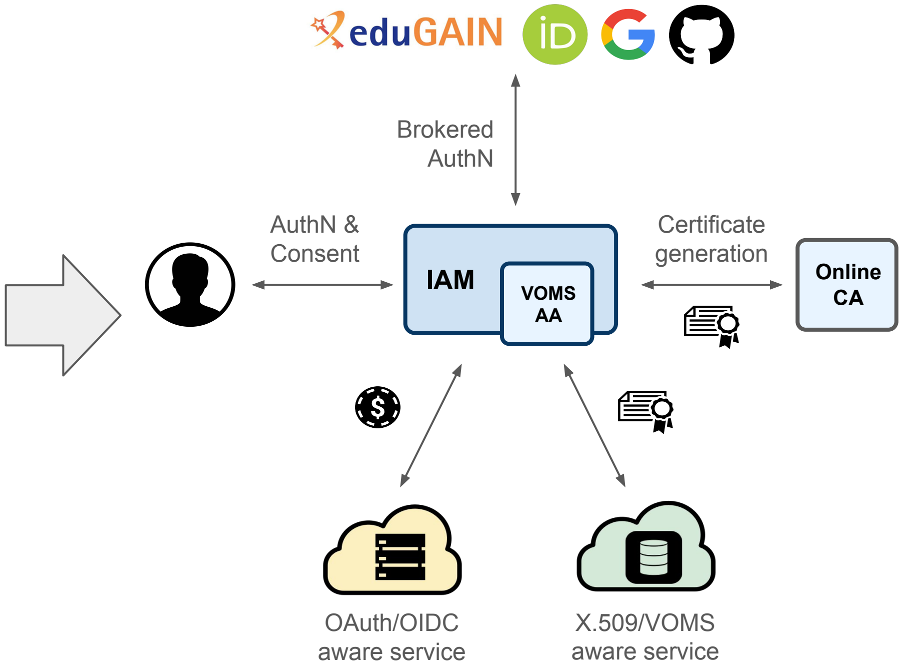


Evolution of the WLCG AAI beyond X.509

Authorization is then **performed at the services** leveraging info extracted from the token:

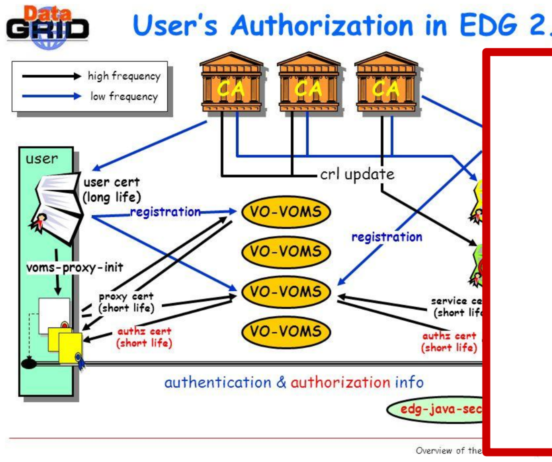
- **Identity attributes**: e.g. groups
- **OAuth scopes**: capabilities linked to access tokens at token creation time

Services can then grant or deny access to functionality based on this information

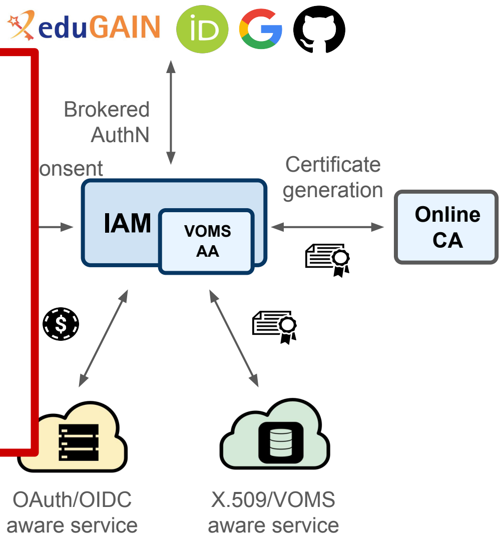


Evolution of the WLCG AAI beyond X.509

Current, X.509-based AAI



Future, token-based AAI



The transition will be gradual!

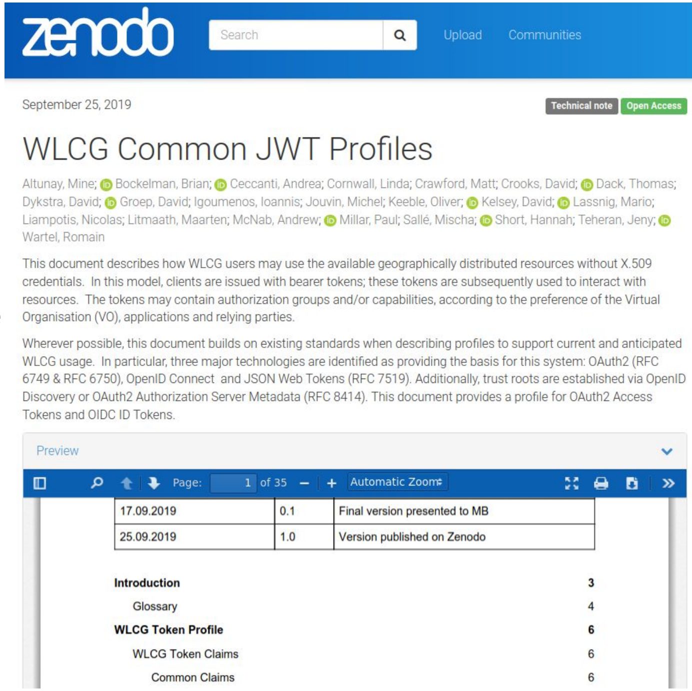
WLCG JWT profile

https://doi.org/10.5281/zenodo.3460258

“This document describes how WLCG users may use the available geographically distributed resources **without X.509 credentials**.”

“In this model, **clients are issued with bearer tokens**; these tokens are subsequently used to interact with resources. **The tokens may contain authorization groups and/or capabilities, according to the preference of the Virtual Organisation (VO), applications and relying parties.”**

“Three major technologies are identified as providing the basis for this system: **OAuth2**, **OpenID Connect** and **JSON Web Tokens**.”



September 25, 2019

WLCG Common JWT Profiles

Altunay, Mine; Bockelman, Brian; Ceccanti, Andrea; Cornwall, Linda; Crawford, Matt; Crooks, David; Dack, Thomas; Dykstra, David; Groep, David; Igoumenos, Ioannis; Jouvin, Michel; Keeble, Oliver; Kelsey, David; Lassnig, Mario; Liampotis, Nicolas; Litmaath, Maarten; McNab, Andrew; Millar, Paul; Sallé, Mischa; Short, Hannah; Teheran, Jeny; Wartel, Romain

This document describes how WLCG users may use the available geographically distributed resources without X.509 credentials. In this model, clients are issued with bearer tokens; these tokens are subsequently used to interact with resources. The tokens may contain authorization groups and/or capabilities, according to the preference of the Virtual Organisation (VO), applications and relying parties.

Wherever possible, this document builds on existing standards when describing profiles to support current and anticipated WLCG usage. In particular, three major technologies are identified as providing the basis for this system: OAuth2 (RFC 6749 & RFC 6750), OpenID Connect and JSON Web Tokens (RFC 7519). Additionally, trust roots are established via OpenID Discovery or OAuth2 Authorization Server Metadata (RFC 8414). This document provides a profile for OAuth2 Access Tokens and OIDC ID Tokens.

17.09.2019	0.1	Final version presented to MB
25.09.2019	1.0	Version published on Zenodo

WLCG specific token claims

- **wlcg.ver**: version of the WLCG token profile the Relying Parties must understand to validate the token
  - it corresponds to the version of the WLCG JWT profile document
  - example: `"wlcg.ver": "1.0"`
- **wlcg.groups**: group information about an authenticated End-User, following a UNIX-like path syntax
  - example: `"wlcg.groups": ["/atlas", "/atlas/pilots", "/atlas/xfers"]`
- **aud**: represents the recipient the JWT is intended for
  - defined more punctually in RFC 8707, BUT
  - the WLCG JWT profile specifies that the `"https://wlcg.cern.ch/jwt/v1/any"` audience must be accepted by all WLCG Relying Parties

Authorization models in WLCG

Capability-based authorization: *scope*

- When a capability is asserted, it has to be honoured by the Resource Servers (RS). It is **the VO** (*i.e.* the Authorization Server), NOT the RS, who **manages authorization within its area**
- The WLCG authorization model follows the recommendation of Section 3.3 of RFC 6749:
  - each desired capability should be requested in the `scope` parameter during the authorization request
  - if an entity is not entitled to a capability, the requested scope may be ignored by the authorization server and the corresponding access token may not have the corresponding claims
  - if the issued access token scope is different from the one requested by the client, the authorization server MUST include the "scope" response parameter to inform the client of the actual scope granted
- The scopes **limit what are the operations that can be authorized by clients presenting an access token** to a RS
- The interpretation of such authorizations would result in a list of operations the bearer is allowed to perform
- Building on the SciTokens experience, define scopes that would match our computing use-cases

Authorization models in WLCG

Identity-based authorization: *wlcg.groups*

- When groups are asserted, the bearer has the access privileges corresponding to the VO's listed groups. It is up to the **RS to determine the mapping of the group names to the access privileges**
- Require the `wlcg.groups` scope to implement a group selection mechanism for groups equivalent to the one provided by VOMS, following the approach outlined in the OpenID Connect standard
    - "*For OpenID Connect, scopes can be used to request that specific sets of information be made available as Claim Values*"
    - in WLCG, scopes are defined and mapped to claims, which are returned in access tokens, ID tokens, and in responses to userinfo and introspection requests
- It results in a `wlcg.groups` claim whose value is an ordered JSON array reflecting the VO groups of which the token subject is a member

Capability-based authorization for storage access

- **storage.read**: Read data. Only applies to *online* resources such as disk (as opposed to *nearline* such as tape where the `storage.stage` authorization should be used in addition)
- **storage.create**: Upload data. This includes renaming files if the destination file does not already exist. This authorization DOES NOT permit overwriting or deletion of stored data
- **storage.modify**: Change data. This includes renaming files and writing data. This permission includes overwriting or replacing stored data in addition to deleting or truncating data
- **storage.stage**: Cause data to be staged from a nearline resource to an online resource. This is a superset of `storage.read`

Capability-based authorization for storage access

Storage scopes additionally provide a resource path, which further limits the authorization

- The resource path follows the format **`$AUTHZ:$PATH`**
  - Example: `storage.read:/foo` provides a read authorization for the resource at `/foo` but not `/bar`
- The resource path may be `/` to authorize the entire resource associated with the issuer
  - Example: a token issued by the Atlas IAM and containing the `storage.modify:/` scope allows to write data in the entire Atlas namespace
- Following the Scitokens model, permissions granted on a path apply transitively to subpaths
  - Example: `storage.read:/cms` grants read access to the `/cms` directory and to all its content, but does not grant read access to the `/atlas` directory

Capability-based authorization for storage access

- This approach is **not equivalent** with POSIX semantics, but matches well with our experiments data access authorization models
    - For example, if a token contains the `storage.read:/home` scope, an implementation must override normal POSIX access control and leave the bearer to access all user's home directories
- Implementing this authorization is up to Client applications (*i.e.* StoRM WebDAV, dCache, *etc.*)

The token just provides a (signed) string!

Capability-based authorization for job submission

- **compute.read**: “Read” or query information about a job status and attributes
- **compute.modify**: Modify or change the attributes of an existing job
- **compute.create**: Create or submit a new job at the computing resource
- **compute.cancel**: Delete a job from the computing resource, potentially terminating a running job

Currently, they refer to all jobs owned by the issuer (*i.e.* a finer-grained path authorization is not foreseen).

For instance, a token with `compute.read` scope issued by https://cms-auth.cern.ch would be able to query the status of any CMS job at the resource

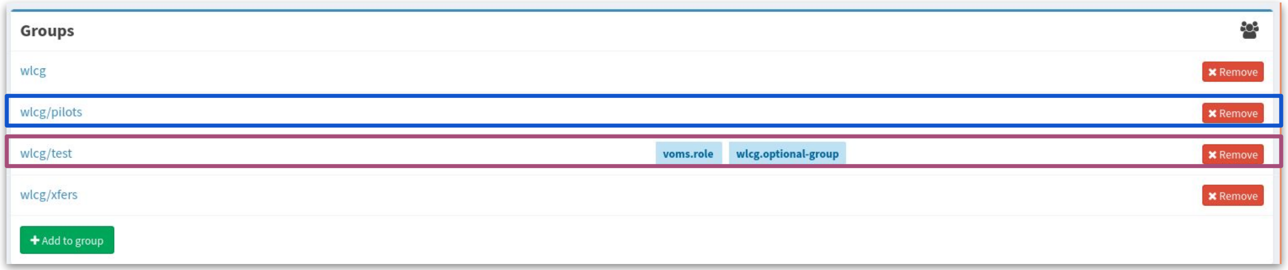
Identity-based authorization using groups

The `wlcg.groups` scope is used to implement an attribute selection mechanism

In the WLCG JWT profile two types of groups have been defined

- **Default groups**, whose membership is always asserted (similar to *VOMS groups*)
- **Optional groups**, whose membership is asserted only when explicitly requested by the Client application (similar to *VOMS roles*)

Those groups appear in the access token when a user (*i.e.* the *sub* of an AT) delegates access to a Client application based on its attributes membership



Groups

wlcg		Remove
wlcg/pilots		Remove
wlcg/test	voms.role wlcg.optional-group	Remove
wlcg/xfers		Remove

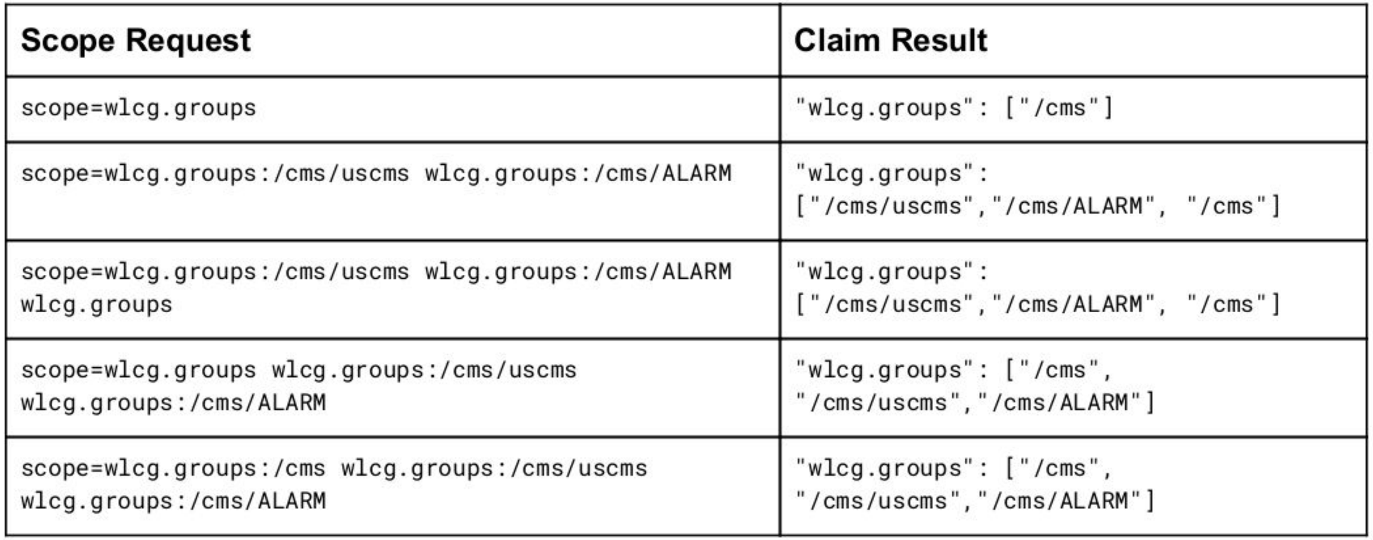
Identity-based authorization using groups

- A parametric `wlcg.groups` scope is introduced with the following form:
  `wlcg.groups[:<group-name>]`
- and the the following rules:
  - if the scope does not have the parametric part, *i.e.* its value is `wlcg.groups`, the authorization server will return the list of default groups for the user being authenticated as a value in the `wlcg.groups` claim
  - if the scope is parametric, (*i.e.* it has the form `wlcg.groups:<group-name>`), in addition to the default groups the authorization server will also return the requested group if the user is member of such group
  - the order of the groups in the returned `wlcg.groups` claim complies with the order in which the groups were requested
  - to request multiple groups, multiple `wlcg.groups:<group-name>` scopes are included in the authorization request
- This seems complex, but it’s the attribute selection mechanism we use everyday with VOMS

Implementing this authorization is (mostly) up to the WLCG AuthZ server (*i.e.*, IAM)!

Identity-based authorization using groups: example

In the following examples `/cms` is the only default group



Scope Request	Claim Result
scope=wlcg.groups	"wlcg.groups": ["/cms"]
scope=wlcg.groups:/cms/uscms wlcg.groups:/cms/ALARM	"wlcg.groups": ["/cms/uscms","/cms/ALARM", "/cms"]
scope=wlcg.groups:/cms/uscms wlcg.groups:/cms/ALARM wlcg.groups	"wlcg.groups": ["/cms/uscms","/cms/ALARM", "/cms"]
scope=wlcg.groups wlcg.groups:/cms/uscms wlcg.groups:/cms/ALARM	"wlcg.groups": ["/cms", "/cms/uscms","/cms/ALARM"]
scope=wlcg.groups:/cms wlcg.groups:/cms/uscms wlcg.groups:/cms/ALARM	"wlcg.groups": ["/cms", "/cms/uscms","/cms/ALARM"]

Trust & security

The profile document also provides recommendations on token lifetimes, trust establishment and other important aspects

Token Type	**Recommended Lifetime**	**Minimum Lifetime**	**Maximum lifetime**	**Justification**
Access Token & ID Token	20 minutes	5 minutes	6 hours	Access token lifetime should be short as we do not foresee the deployment of a revocation mechanism. The granted lifetime has implications for the maximum allowable downtime of the Access Token server
Refresh Token	10 days	1 day	30 days	Refresh token lifetimes should be kept bounded, but can be longer-lived as they are revocable. Meant to be long-lived enough to be on a “human timescale.” Refresh tokens are not necessarily signed and not tied to issuer public key lifetime
Issuer Public Key Cache	6 hours	1 hour	1 day	The public key cache lifetime defines the minimum revocation time of the public key. The actual lifetime is the maximum allowable downtime of the public key server.
Issuer Public key	6 months	2 days	12 months	JWT has built-in mechanisms for key rotation; these do not need to live as long as CAs. This may evolve following operational experience, provision should be made for flexible lifetimes

WLCG JWT profile v1.1

- There is a draft for the next version of the WLCG JWT profile
- In particular:
  - definition of `wlcg.capability` scope/claim
  - specify the hierarchical authorization based on sub-groups
  - clarify the authorization model when the capability and identity is asserted in the AT
  - improve authorization based on `storage.*` scopes

JWT profiles in INDIGO IAM

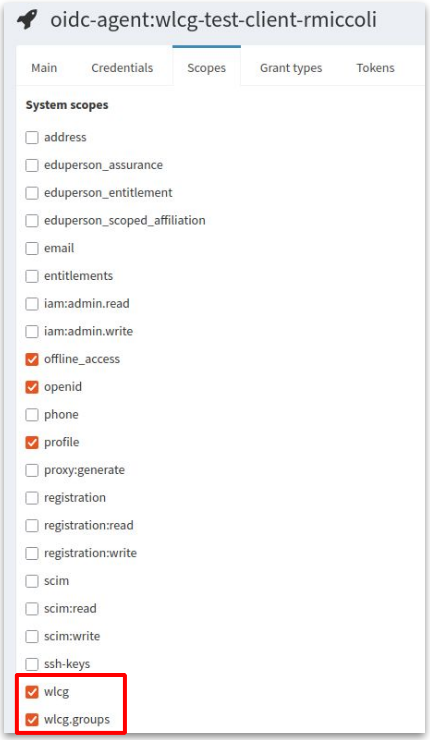
- A JWT profile* is a named set of rules that defines which information is included in access tokens, id tokens, userinfo and introspection responses issued by IAM in OAuth/OIDC flows
- The JWT default profile is `IAM` and is set using the `IAM_JWT_DEFAULT_PROFILE` environment variable
  - the default profile will be used for all clients that do not explicitly and correctly request the use of a profile using *scopes*
- To select the JWT profile used by a client, include one of the following scopes in the list of scopes authorized for a client
  - **`iam`**, **`wlcg`**, **`aarc`**, **`keycloak`**
  - this will override the default JWT profile for that specific client
  - when multiple profiles are linked to a client, IAM reverts to the configured default profile

*explained in more detail in INDIGO IAM slides

WLCG JWT profile in INDIGO IAM

- Enabled with the **`wlcg`** scope
- Groups are encoded in the **`wlcg.groups`** claim
  - all the non-optional groups the user is member of are included
  - not included by default in access and ID tokens, but can be requested using the `wlcg.groups` scope (to be added among the *Scopes* by IAM administrators)
- To configure an IAM group as optional group, add the **`wlcg.optional-group`** label to the group

WLCG JWT profile in INDIGO IAM


oidc-agent:wlcg-test-client-rmiccoli

Main | Credentials | Scopes | Grant types | Tokens

System scopes

- [ ] address
- [ ] eduperson_assurance
- [ ] eduperson_entitlement
- [ ] eduperson_scoped_affiliation
- [ ] email
- [ ] entitlements
- [ ] iam:admin.read
- [ ] iam:admin.write
- [✓] offline_access
- [✓] openid
- [ ] phone
- [✓] profile
- [ ] proxy:generate
- [ ] registration
- [ ] registration:read
- [ ] registration:write
- [ ] scim
- [ ] scim:read
- [ ] scim:write
- [ ] ssh-keys
- [✓] wlcg
- [✓] wlcg.groups

AT request <u>without</u> the `wlcg.groups` scope

```
$ oidc-token wlcg-test-client

{
  "sub":
"73f16d93-2441-4a50-88ff-85360d78c6b5",
  "iss": "http://localhost:8080",
  "preferred_username": "admin",
  "client_id":
"b47eb46a-f2dc-4a7e-b0c8-2f1e81dd5d4a",
  "wlcg.ver": "1.0",
  "aud": "https://wlcg.cern.ch/jwt/v1/any",
  "nbf": 1746020859,
  "scope": "openid offline_access profile",
  "name": "Admin User",
  "exp": 1746024459,
  "iat": 1746020859,
  "jti":
"253bf3ad-da86-43e3-86c7-70ffd2204d25"
}
```

If no audience is requested, the audience claim will be populated by `https://wlcg.cern.ch/jwt/v1/any`

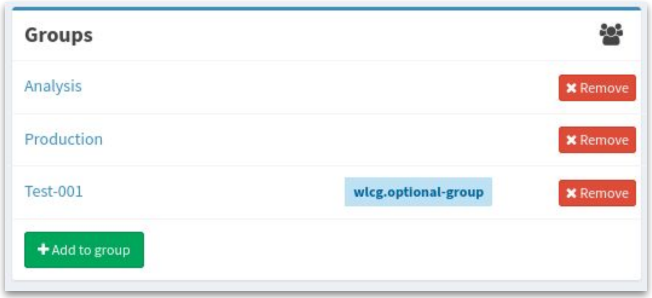
AT request <u>with</u> the `wlcg.groups` scope

```
$ oidc-token -s wlcg.groups wlcg-test-client

{
  "wlcg.ver": "1.0",
  "sub":
"73f16d93-2441-4a50-88ff-85360d78c6b5",
  "aud": "https://wlcg.cern.ch/jwt/v1/any",
  "nbf": 1746021100,
  "scope": "wlcg.groups",
  "iss": "http://localhost:8080",
  "exp": 1746024700,
  "iat": 1746021100,
  "jti":
"457467c2-95cd-4caa-b22b-3d010a9add13",
  "client_id":
"b47eb46a-f2dc-4a7e-b0c8-2f1e81dd5d4a",
  "wlcg.groups": [
        "/Analysis",
        "/Production"
  ]
}
```

The user must first authorize the client to access the `wlcg.groups` scope via the consent page!

WLCG JWT profile in INDIGO IAM



AT request with the optional group as parametric scope

```
$ oidc-token -s wlcg.groups:/Test-001 wlcg-test-client

{
  "wlcg.ver": "1.0",
  "sub": "73f16d93-2441-4a50-88ff-85360d78c6b5",
  "aud": "https://wlcg.cern.ch/jwt/v1/any",
  "nbf": 1746022716,
  "scope": "wlcg.groups:/Test-001",
  "iss": "http://localhost:8080",
  "exp": 1746026316,
  "iat": 1746022716,
  "jti": "2500cda3-9ea7-46c5-a393-20e4031319f3",
  "client_id":
"b47eb46a-f2dc-4a7e-b0c8-2f1e81dd5d4a",
  "wlcg.groups": [
        "/Test-001",
        "/Analysis",
        "/Production"
  ]
}
```