Introduction to token-based AuthN/Z with OAuth/OpenID Connect and INDIGO IAM

Andrea Ceccanti
INFN CNAF

WLCG CE Hackathon
June, 3rd 2021

Getting an account on WLCG IAM

Getting an account on WLCG IAM

Please apply for an account in the WLCG IAM instance (if you haven't an account already)

https://wlcg.cloud.cnaf.infn.it

Click on the "Sign in with CERN SSO" button and fill in the registration form, putting "WLCG CE Hackathon" in the request notes.

Getting tokens out of WLCG IAM

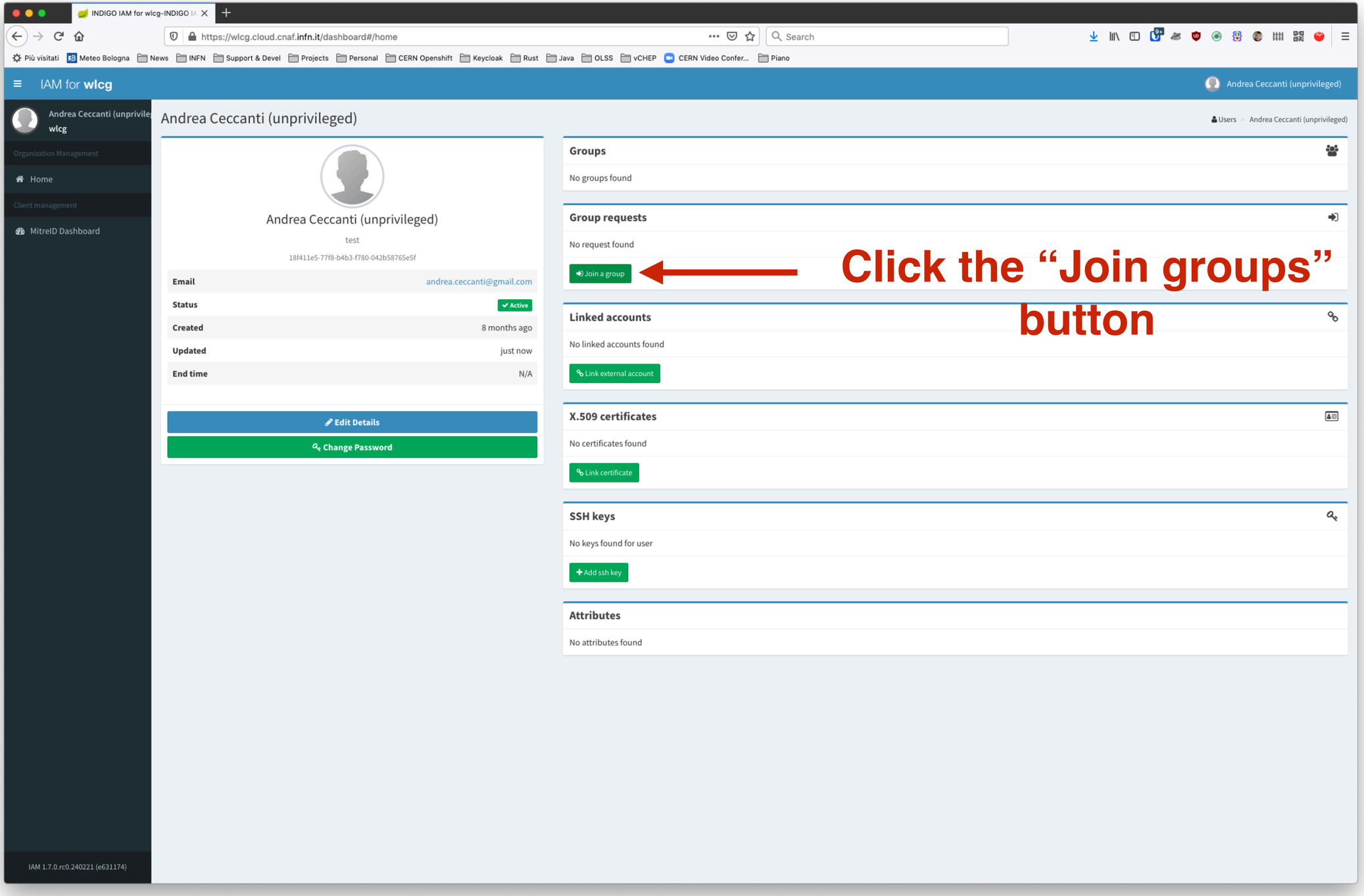
https://indigo-iam.github.io/docs/v/current/user-guide/getting-a-token.html

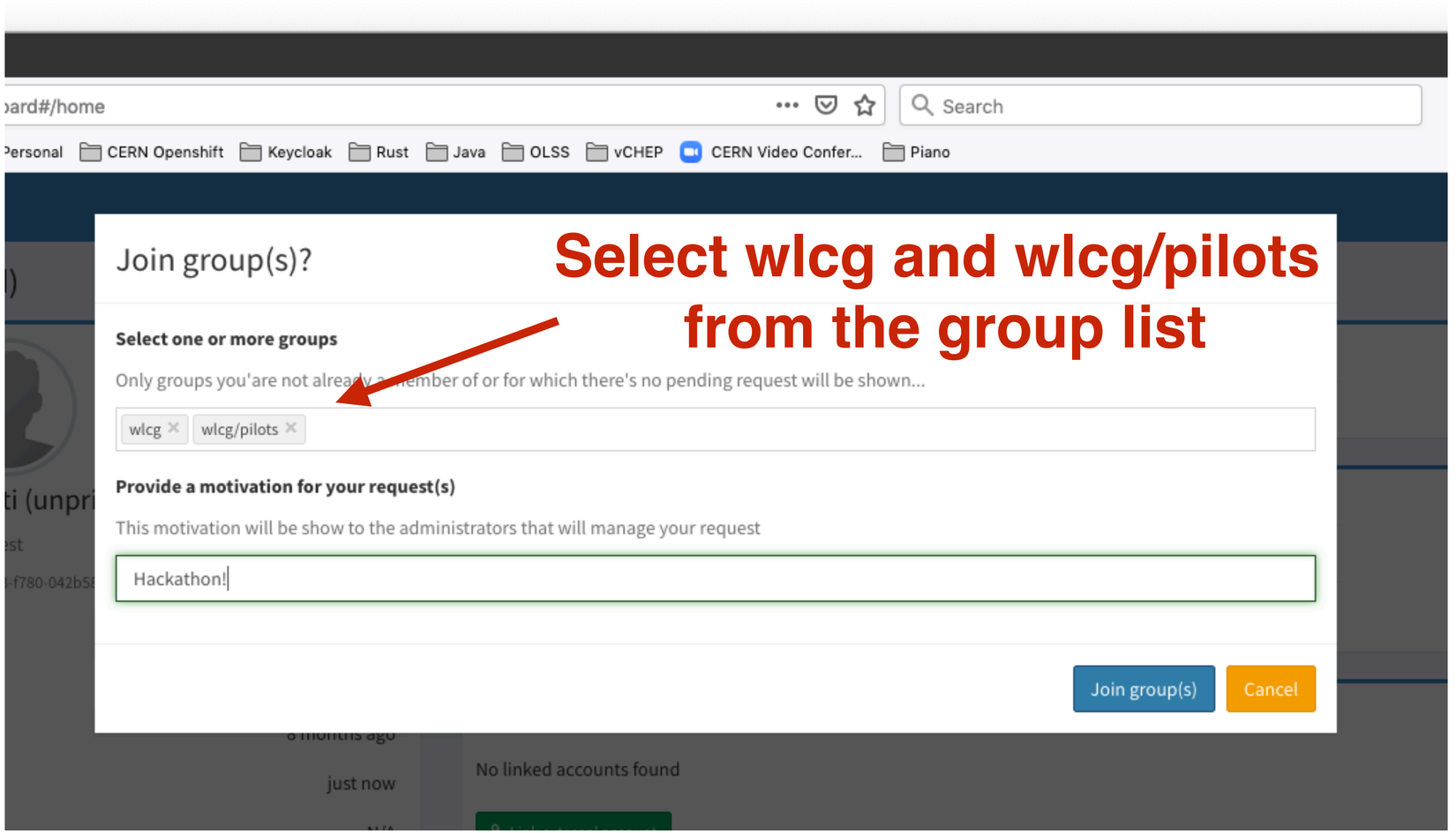
In order to submit jobs to the HTCondor CE, your token will require the following scopes:

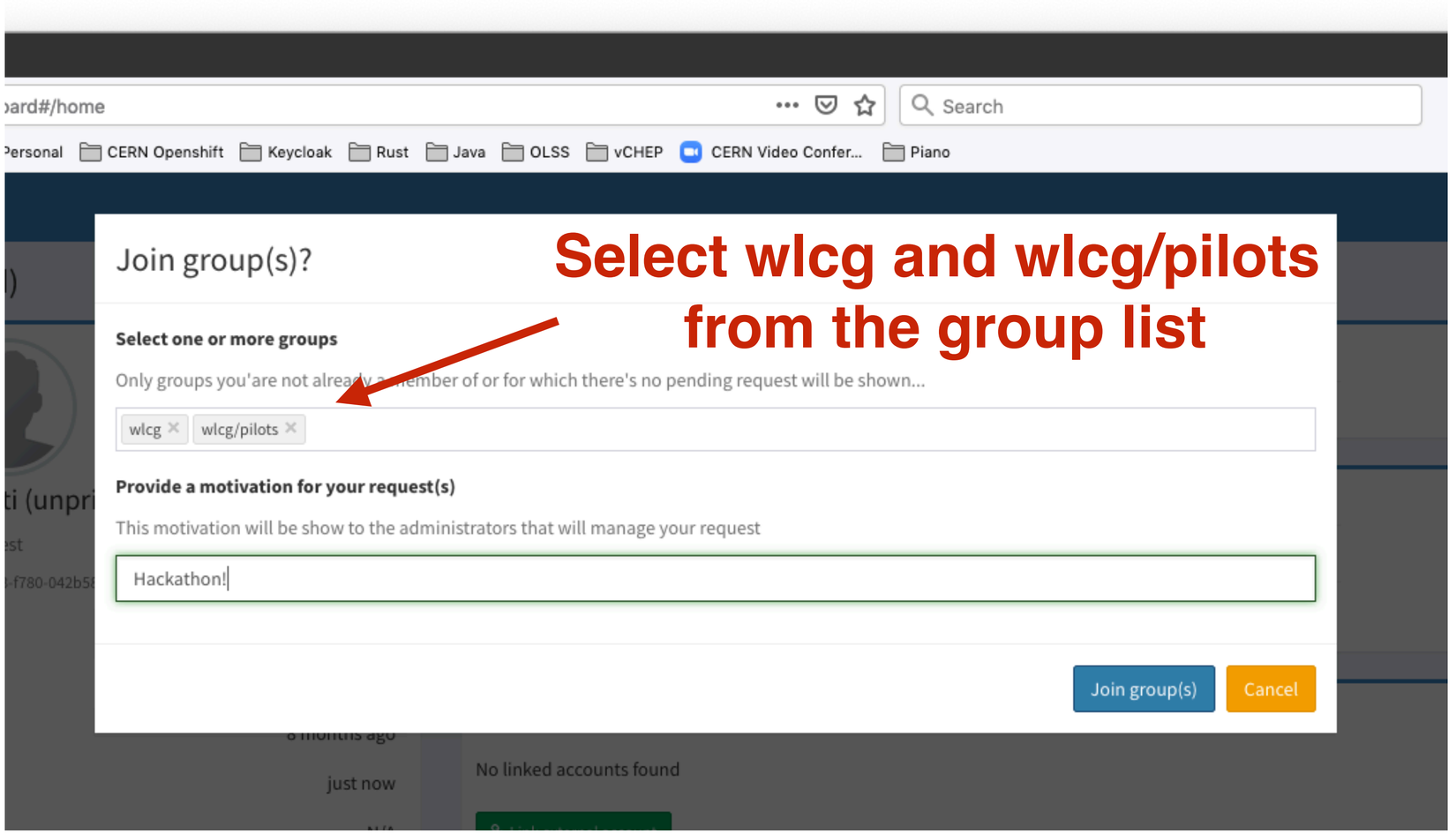
- **compute.create, compute.modify, compute.cancel, compute.read**

Access to these scopes is limited to members of the wlcg/pilots group

Submit a group membership request from the IAM dashboard if you're not already member of the group


INDIGO IAM for wlcg-INDIGO
https://wlcg.cloud.cnaf.infn.it/dashboard#/home
Search
IAM for wlcg
Andrea Ceccanti (unprivileged)
Andrea Ceccanti (unprivile
wlcg
Organization Management
Home
Client management
MitreID Dashboard
IAM 1.7.0.rc0.240221 (e631174)
Andrea Ceccanti (unprivileged)
Users > Andrea Ceccanti (unprivileged)
Andrea Ceccanti (unprivileged)
test
18f411e5-77f8-b4b3-f780-042b58765e5f
Email andrea.ceccanti@gmail.com
Status Active
Created 8 months ago
Updated just now
End time N/A
Edit Details
Change Password
Groups
No groups found
Group requests
No request found
Join a group
Click the “Join groups” button
Linked accounts
No linked accounts found
Link external account
X.509 certificates
No certificates found
Link certificate
SSH keys
No keys found for user
Add ssh key
Attributes
No attributes found


Search
Personal
CERN Openshift
Keycloak
Rust
Java
OLSS
vCHEP
CERN Video Confer...
Piano
Join group(s)?
Select wlcg and wlcg/pilots from the group list
Select one or more groups
Only groups you're not already a member of or for which there's no pending request will be shown...
wlcg
wlcg/pilots
Provide a motivation for your request(s)
This motivation will be show to the administrators that will manage your request
Hackathon!
Join group(s)
Cancel
No linked accounts found
just now



Select wlcg and wlcg/pilots from the group list

You won't be allowed to request membership if you are already a member of those groups

Registering a client with oidc-agent

1. Install oidc-agent (see https://indigo-dc.gitbook.io/oidc-agent/installation/install)
2. Register a new client for the hackathon:

   ```
   $ eval $(oidc-keychain)
   $ oidc-gen -w device hackathon
   ```

   (select the wlcg issuer and type in ‘max’ when prompted about which scopes should be requested)
3. Get a token

   ```
   $ oidc-token -s openid -s compute.modify -s compute.create hackathon
   ```

Registering a client with oidc-agent

1. Install oidc-agent (see https://indigo-dc.gitbook.io/oidc-agent/installation/install)
2. Register a new client for the
   ```
   $ eval $(oidc-keychain)
   $ oidc-gen -w device hackathor
   ```
   (select the wlcg issuer and type in requested)
3. Get a token
   ```
   $ oidc-token -s openid -s compute.modify -s compute.create hackathon
   ```

If you dont' provide scope arguments you will get a very privileged token with all the scopes your client is allowed to request.
Don't do this, limit the scope of the tokens as much as possible

Example: getting a token with oidc-agent

```
⌥⌘1                                                    root@amnesiac:/var/log/storm/webdav
02/6, 9:49 PM        ~/git/iam        -bash                                          6%

iam on  develop [$] via ☕ v1.8.0
❯ oidc-token -s openid -s compute.modify -s compute.create -s wlcg.groups hackathon | jwt

To verify on jwt.io:

https://jwt.io/#id_token=eyJraWQiOiJyc2ExIiwiYWxnIjoiUlMyNTYifQ.eyJ3bGNnLnZlciI6IjEuMCIsInN1YiI6IjE4ZjQxMWU1LTc3ZjgtYjRiMy1mNzgwLTA0MmI1ODc2NWU1ZiIsImF1ZCI6Imh
djFcL2FueSIsIm5iZiI6MTYyMjY2MzMzMSwic2NvcGUiOiJvcGVuaWQgY29tcHV0ZS5jcmVhdGUgY29tcHV0ZS5tb2RpZnkgd2xjZy5ncm91cHMiLCJpc3MiOiJodHRwczpcL1wvd2xjZy5jbG91ZC5jbmFmLmluZn
aWF0IjoxNjIyNjYzMzMxLCJqdGkiOiJlMDNmNzViYi1lZTRmLTQxNDktOWUyMS01YWY1MjUzNTVhNmYiLCJjbGllbnRfaWQiOiJjYTdiZGQwNi1kNTljLTQ0MjItYmJjYS02YzAzZDlhMWUxYjYiLCJ3bGNnLmd
bG90cyJdfQ.E-baMZAIZpwmNpLG3W3NxTwgwNVCurrka-BVE30ztaHuHeC57W3OVrqL9O56Fd_BBhiEt6GYaIOj65KA3WmFWzEqgqUhLm8KJVD9t_LAPOPmsvcW8OCxcKU4lOoBF7l7CaV3ybpULn2Y9DTjvjEz

* Header
{
  "kid": "rsa1",
  "alg": "RS256"
}

* Payload
{
  "wlcg.ver": "1.0",
  "sub": "18f411e5-77f8-b4b3-f780-042b58765e5f",
  "aud": "https://wlcg.cern.ch/jwt/v1/any",
  "nbf": 1622663331,
  "scope": "openid compute.create compute.modify wlcg.groups",
  "iss": "https://wlcg.cloud.cnaf.infn.it/",
  "exp": 1622666931,
  "iat": 1622663331,
  "jti": "e03f75bb-ee4f-4149-9e21-5af525355a6f",
  "client_id": "ca7bdd06-d59c-4422-bbca-6c03d9a1e1b6",
  "wlcg.groups": [
    "/wlcg",
    "/wlcg/pilots"
  ]
}
   Issued At: 1622663331 6/2/2021, 9:48:51 PM
   Not Before: 1622663331 6/2/2021, 9:48:51 PM
   Expiration Time: 1622666931 6/2/2021, 10:48:51 PM
```

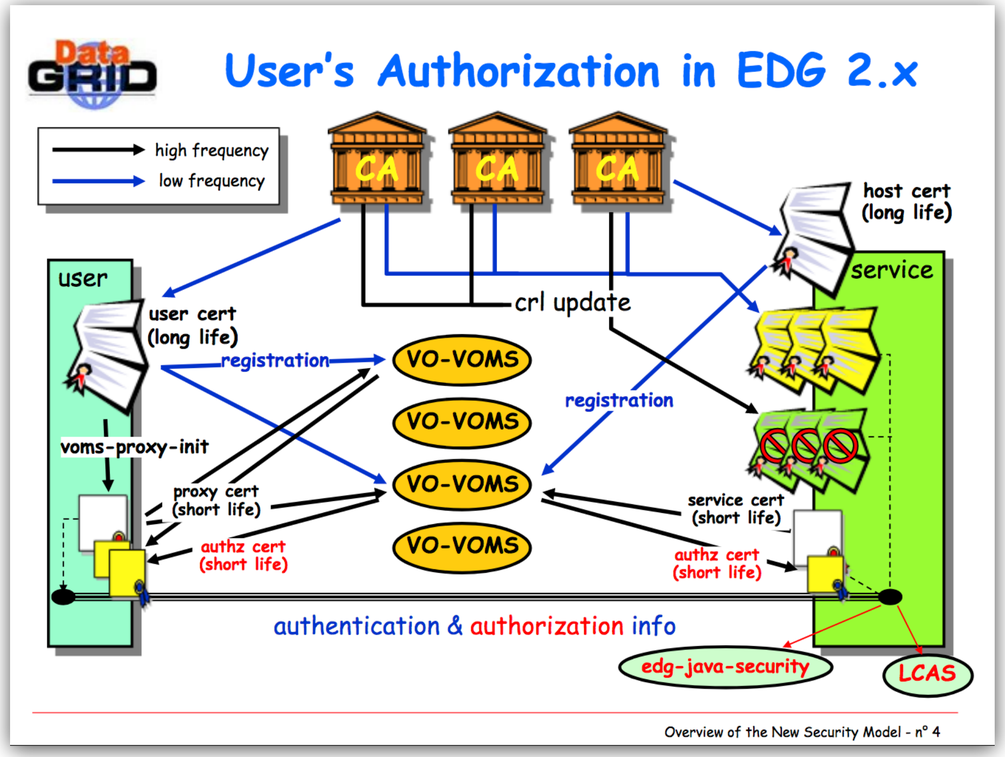
What happens behind the scenes

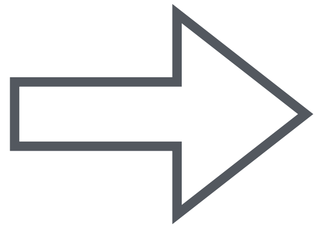
oidc-agent registers a new client and after authenticating the user and getting his/her consent to access the information linked to the requested scopes, **it stores a refresh token locally together with the client configuration and encrypts everything using a user provided password.**

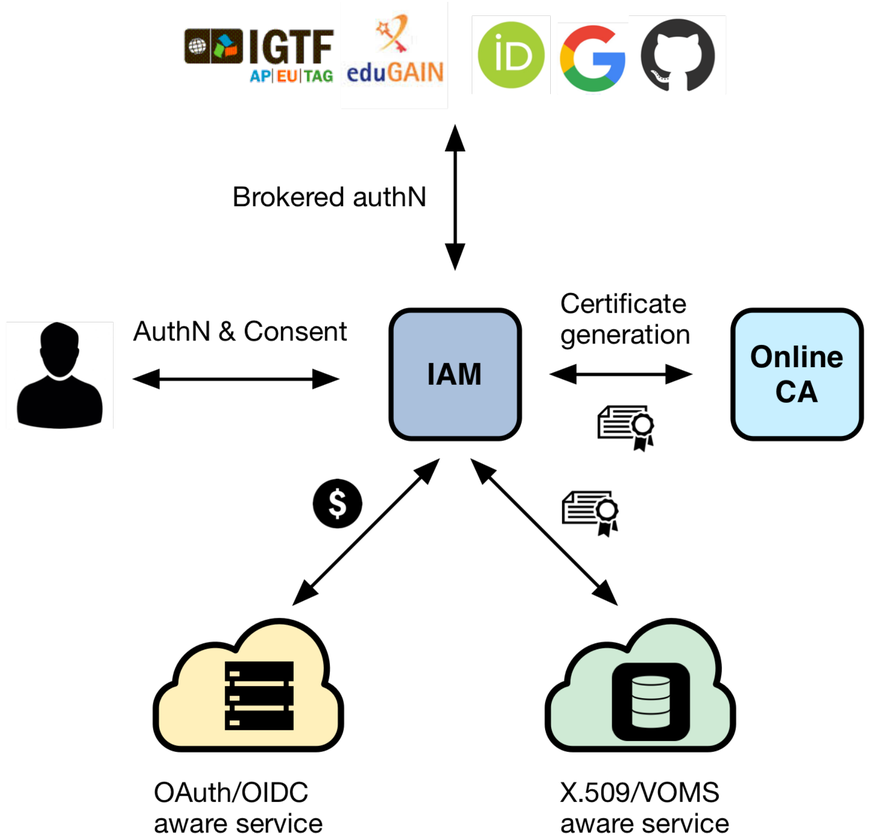
This refresh token is then used to request new tokens from IAM as needed

A brief introduction to OAuth, OpenID Connect and JWTs

Objective: evolution of the WLCG AAI beyond X.509







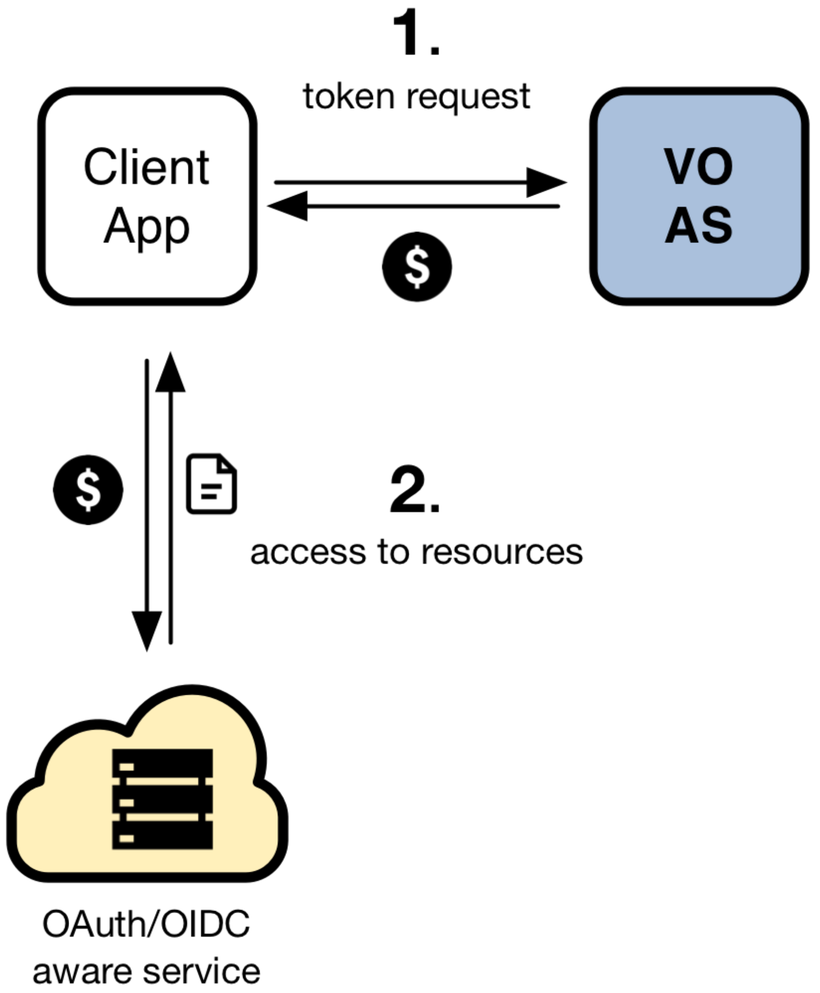
Token-based AuthN/Z for WLCG

In order to access resources/services, a **client application** needs an **access token**

The token is obtained from **a VO** (which acts as an OAuth Authorization Server) using standard **OAuth/OpenID Connect** flows

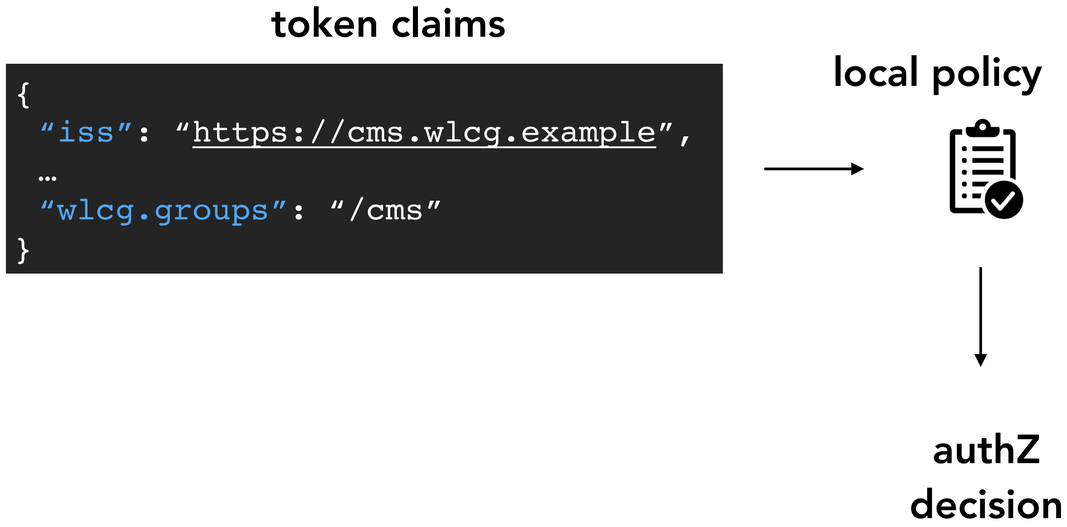
Authorization is then **performed at the services** leveraging info extracted from the token:

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

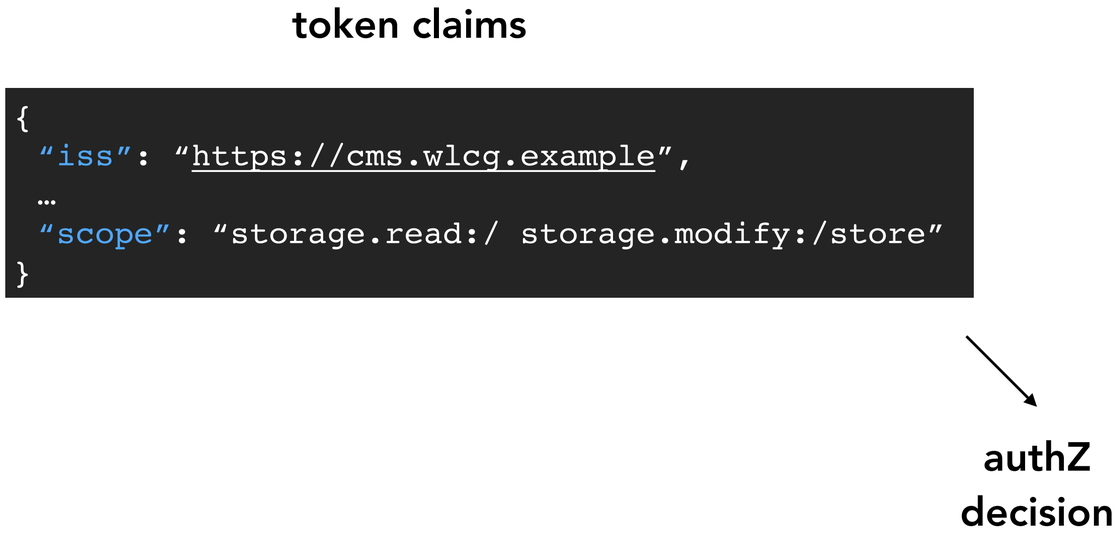


Identity-based vs Scope-based Authorization

Identity-based authorization: the token brings information about attribute ownership (e.g., groups/role membership), the service maps these attributes to a local authorization policy



Scope-based authorization: the token brings information about which actions should be authorized at a service, the service needs to understand these capabilities and honor them. The authorization policy is managed at the VO level

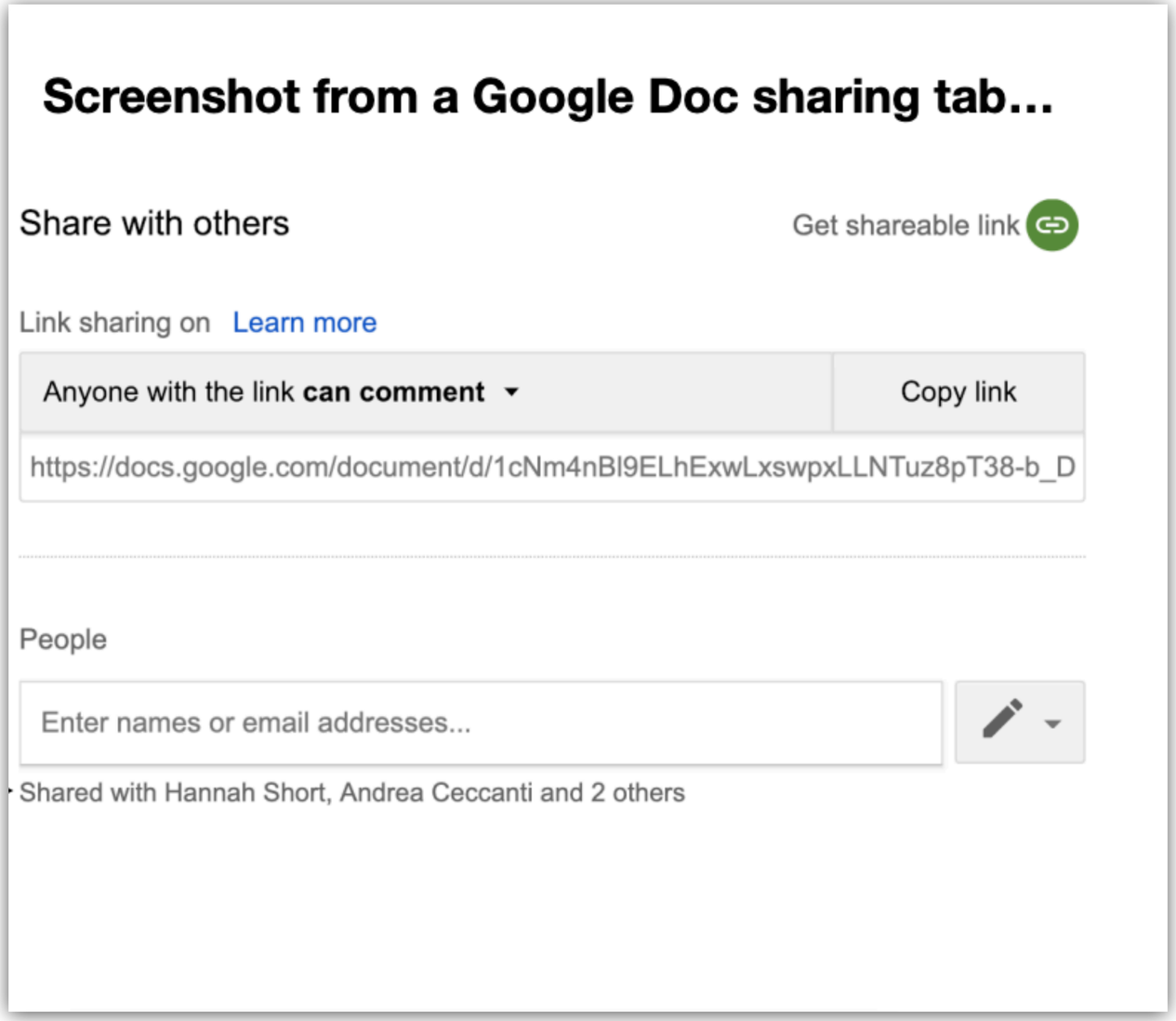


Identity-based vs Scope-based Authorization

The two models can coexist, even in the context of the same application!

scope-based authZ

identity-based authZ



* Slide courtesy of B. Bockelman

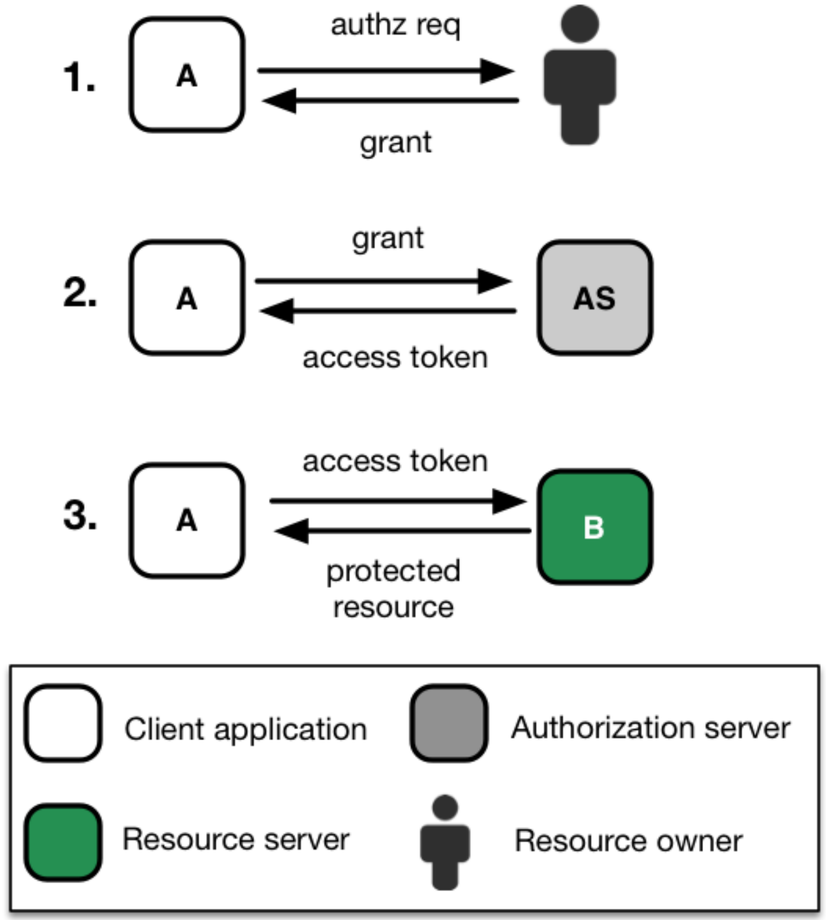
One slide summary

- To access computing and storage resources in the WLCG today you use a VOMS proxy, which provides information about who you are, for which VO you're acting and what you can do on the infrastructure (i.e., VOMS groups and roles)
- In the near future we will use **tokens,** which will provide more or less the same information
- Tokens are obtained from a VO token issuer (e.g., IAM) using OpenID Connect
- Tokens are sent to services/resources following OAuth recommendations (e.g., embedded in the header or an HTTP request)
- Tokens are self-contained, i.e. their integrity and validity can be verified locally with no callback to the token issuer

Funded by the European Union's
Horizon 2020 - Grant N° 824064

OAuth roles

- **Resource owner**
  - A user that owns resources hosted at a service
- **Client**
  - An application that wants to have access to user resources
- **Authorization server**
  - A service that authenticates users and client applications and issues access tokens according to some policy
- **Resource server**
  - A service that holds protected resources and grants access based on access tokens issued by the authorization server



Funded by the European Union's Horizon 2020 - Grant N° 824064

OAuth/OpenID Connect actors and roles

Actor	Role	Example
Authorization Server (AS)	Asserting party	WLCG IAM instance
Resource Server (RS)	Relying party	HTCondor job submission API
Client	Relying party	Experiment framework (e.g., PANDA)
Resource Owner	Subject	A registered WLCG IAM user

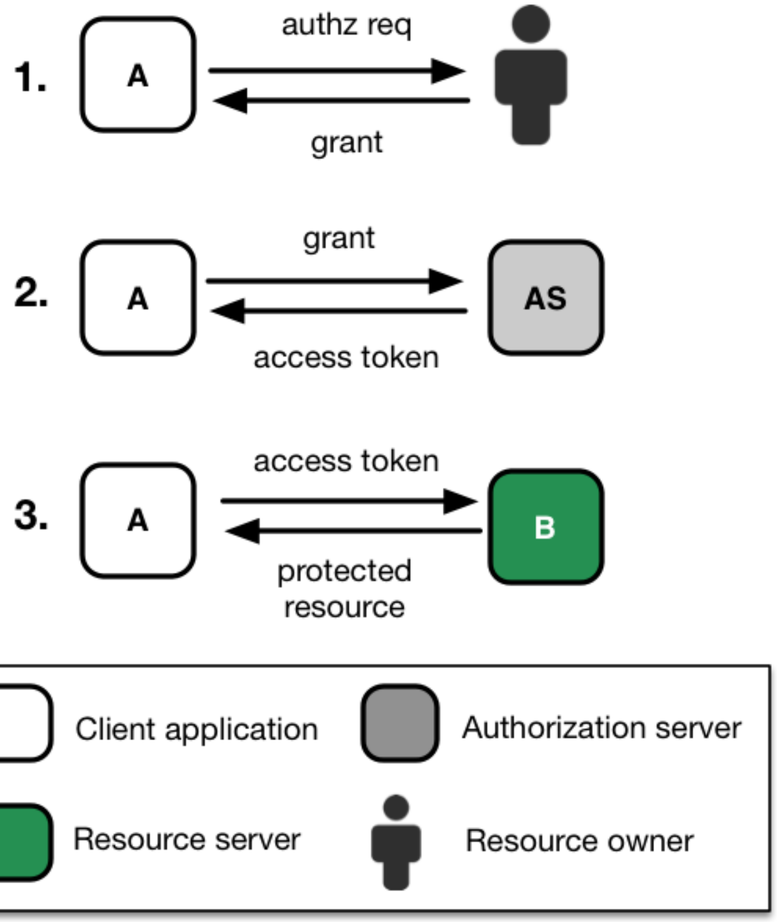
Funded by the European Union's Horizon 2020 - Grant N° 824064

OAuth client registration

- In OAuth clients that interact with an Authorization Server (AS) need to be **registered**

- When a client is registered, it typically receives the client **credentials**
  - **client_id:** the client "username"
  - **client_secret:** the client "password"

- Credentials are required in some OAuth/OpenID Connect flows or to access specific endpoints, where different privileges may be assigned to different clients



Funded by the European Union's Horizon 2020 - Grant N° 824064

OAuth client types

https://tools.ietf.org/html/rfc6749#section-2.1

- **confidential:** Clients capable of maintaining the confidentiality of their credentials (e.g., client implemented on a secure server with restricted access to the client credentials), or capable of secure client authentication using other means

- **public:** Clients incapable of maintaining the confidentiality of their credentials (e.g., clients executing on the device used by the resource owner, such as an installed native application or a web browser-based application), and incapable of secure client authentication via any other means.

Funded by the European Union's
Horizon 2020 - Grant N° 824064

Handling client credentials

- Client credentials must be maintained confidential
  - **not** stored in Docker images or source code
    - use ENV variables or other secret management mechanisms to pass secrets to your application
- Follow recommendations in the client app security section of the OAuth security recommendations
  - https://tools.ietf.org/html/rfc6819#section-5.3

Funded by the European Union's
Horizon 2020 - Grant N° 824064

Client registration in practice

- To register a new client in IAM, follow the instructions in the documentation:
  - https://indigo-iam.github.io/docs/v/current/user-guide/client-registration.html
- Client registration is necessary to integrate any application that needs to “drive” an authorization flow
  - i.e., if your app needs to show a “Login with WLCG IAM” button, i.e. needs to authenticate users, you need to register a client
- For protected resources (APIs) integration, registration is **NOT** needed

Funded by the European Union’s
Horizon 2020 - Grant N° 824064

OAuth/OpenID Connect grant types

Authorization grant types

=

Authorization Flows

=

Ways for an application to get tokens

Funded by the European Union's
Horizon 2020 - Grant N° 824064

OAuth/OpenID Connect grant types

Grant Type	Context	Client type
Authorization code	Server-side apps	Confidential
Implicit	Client-side, Javascript apps	Public
Device code	Limited-input devices, CLIs	Confidential
Resource owner password credentials	Trusted apps, CLIs	Confidential
Client credentials	Server-side apps	Confidential
Refresh token	Server-side apps	Confidential
Token exchange	Server-side apps	Confidential

Funded by the European Union's
Horizon 2020 - Grant N° 824064

OAuth/OpenID Connect grant types

Grant Type	Context	Client type
Authorization code	Server-side apps	Confidential
Implicit	Client-side, Javascript apps	Public
Device code	Limited-input devices, CLIs	Confidential
Resource owner password credentials	Trusted apps, CLIs	Confidential
Client credentials	Server-side apps	Confidential
Refresh token	Server-side apps	Confidential
Token exchange	Server-side apps	Confidential

These are the main grant types that will be used in WLCG

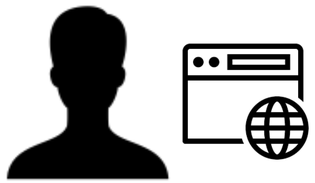
Funded by the European Union's Horizon 2020 - Grant N° 824064

Authorization code flow

- The recommended flow for server-side applications that can maintain the confidentiality of client credentials
- Allows an application to obtain tokens to act on behalf of a user for a potentially unbounded amount of time
- See more on this flow in the RFC:
  - https://openid.net/specs/openid-connect-core-1_0.html#CodeFlowAuth
  - https://datatracker.ietf.org/doc/html/rfc6749#page-24

Funded by the European Union's
Horizon 2020 - Grant N° 824064

Web application: authorization code flow



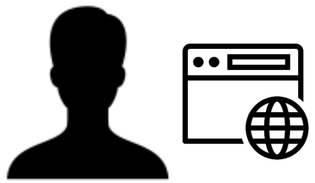
Web App

A Web App integrates with IAM to **delegate user authentication management** and **obtain authorization** information

Home IdP

IAM

Web application: authorization code flow



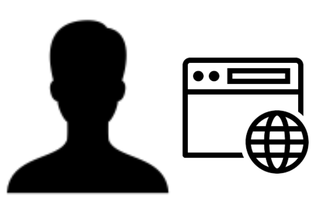
Web App

OAuth and OpenID connect provide the **authorization code flow** in support of this integration use case

Home IdP

IAM

Web application: authorization code flow



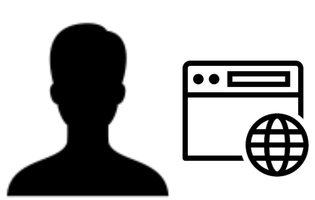
Web App

User points its browser to web app, which redirects back to IAM for authentication

Home IdP

IAM

Web application: authorization code flow



Web App

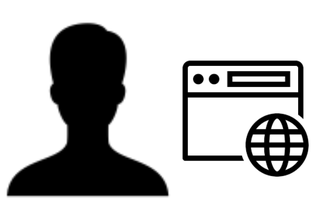
authorization request

IAM

Home IdP

User points its browser to web app, which redirects back to IAM for authentication

Web application: authorization code flow



Web App

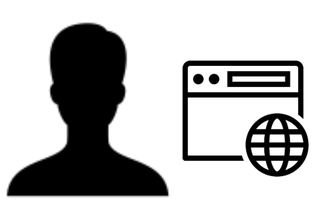
authorization request

IAM

Home IdP

This authorization request starts the authorization flow, and includes parameters (e.g., OAuth scopes) that will influence which information is returned by IAM

Web application: authorization code flow



Web App

authorization request

IAM

Home IdP

User does not have a valid session at IAM, so IAM shows the login page

Web application: authorization code flow

authorizatio
request

INDIGO - DataCloud

Welcome to **dodas**

Sign in with your dodas credentials

Username

Password

Sign in

Forgot your password?

Or sign in with

Google

eduGAIN

egi

Not a member?

Register a new account

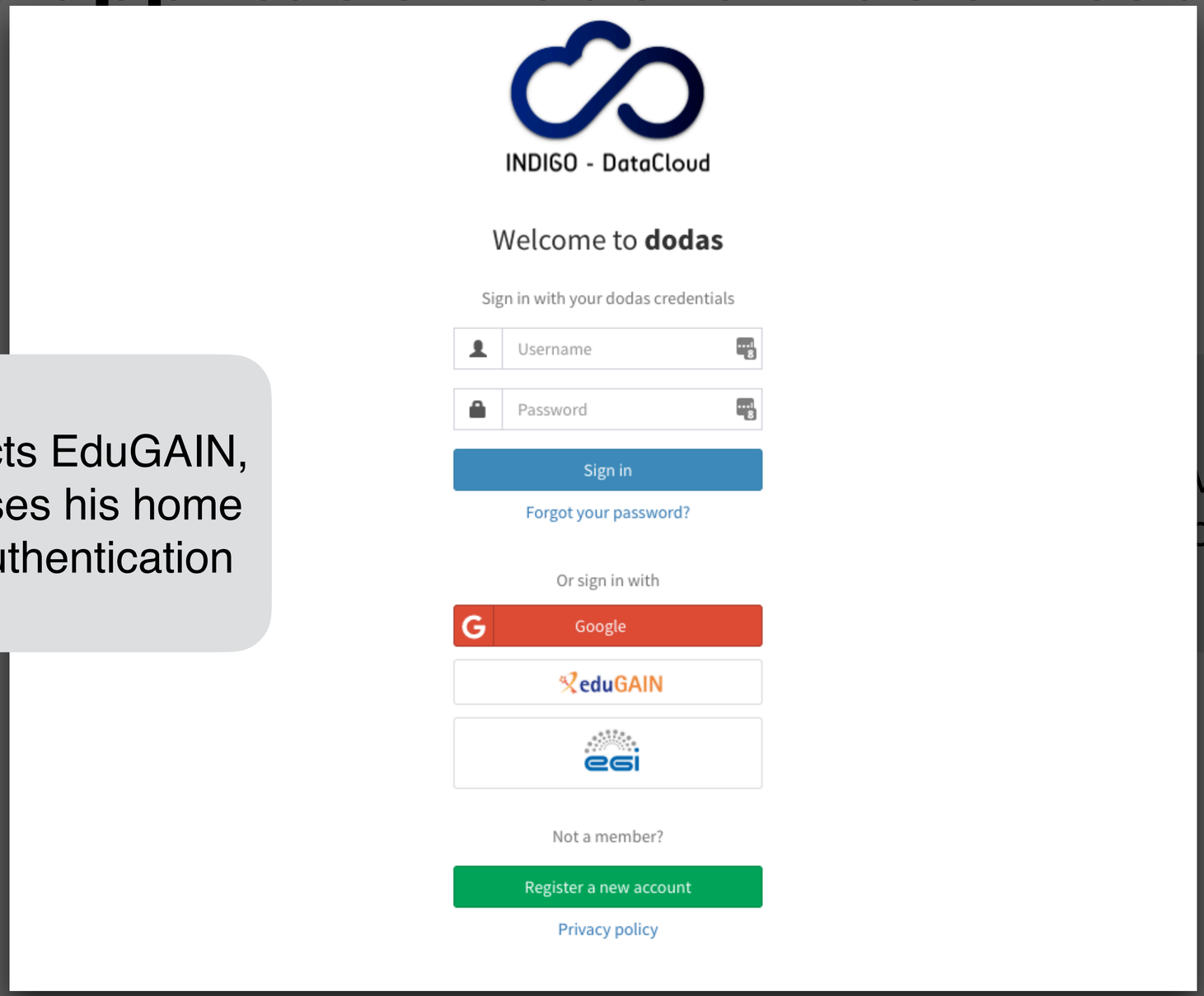
Privacy policy

ve a valid session at
ows the login page

Home IdP

Web application: authorization code flow

User selects EduGAIN, and chooses his home IDP for authentication

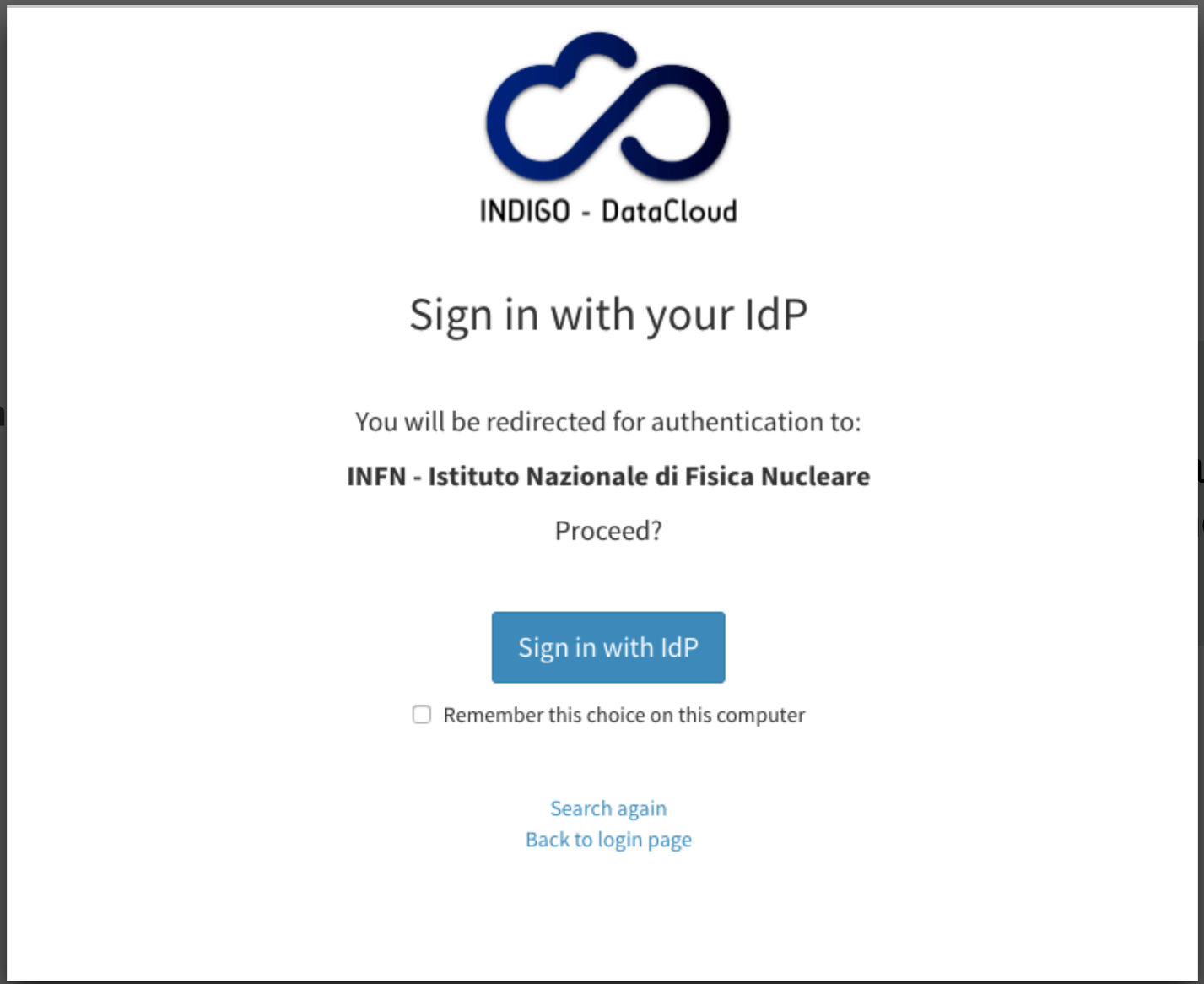


ve a valid session at
ows the login page

Home IdP

Web application: authorization code flow

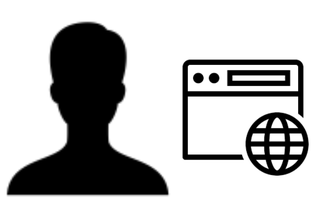
authorization request



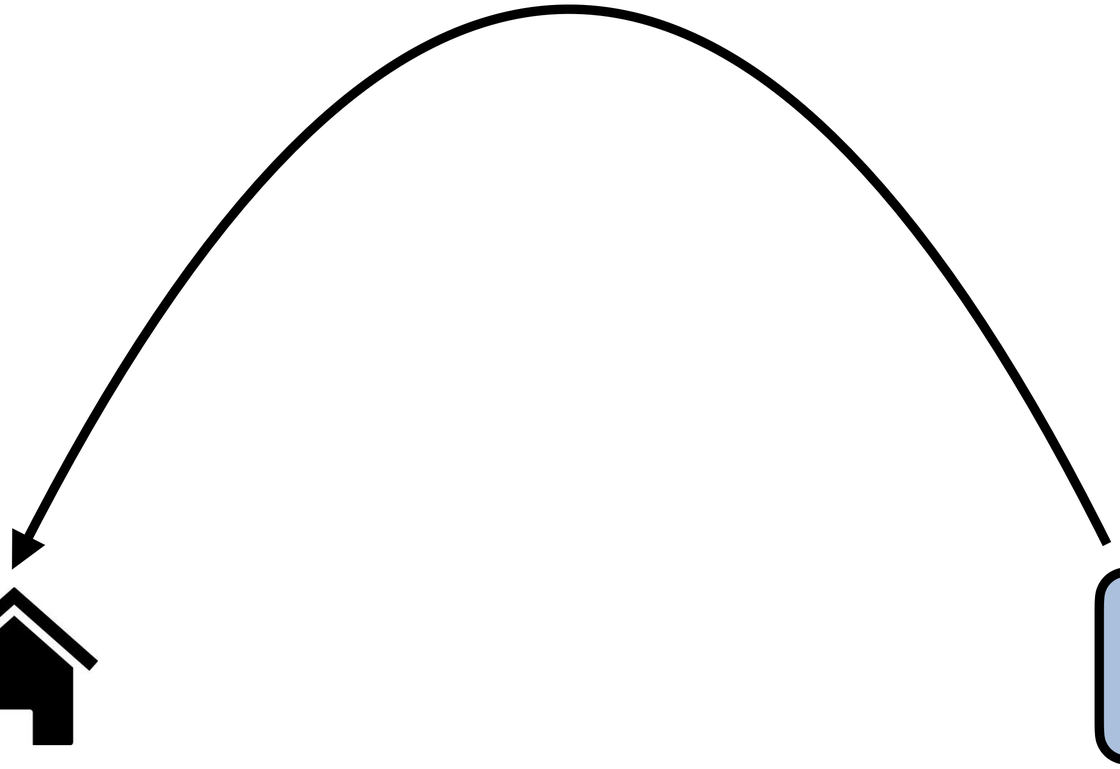
ve a valid session at
ows the login page

Home IdP

Web application: authorization code flow



Web App

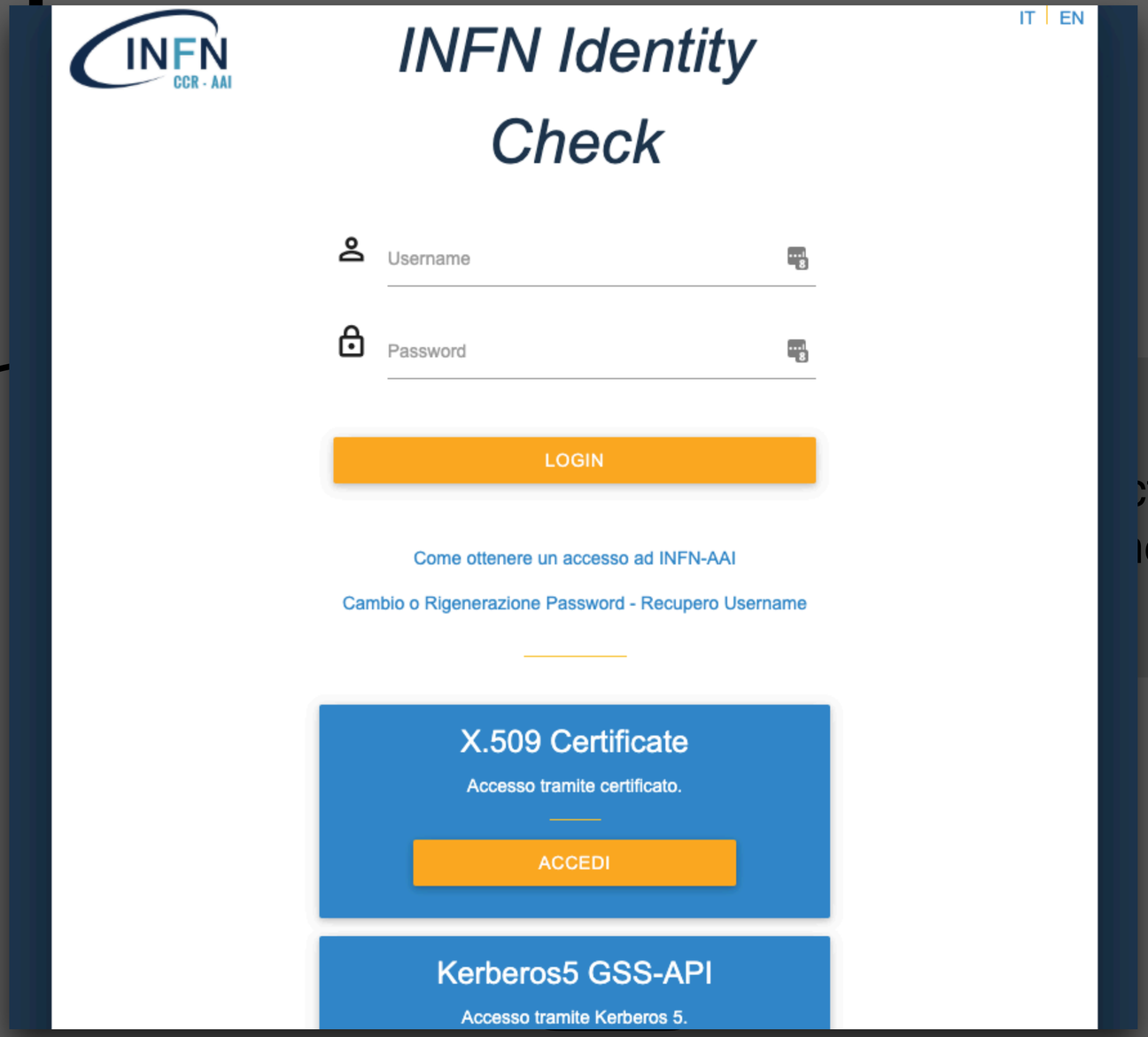


IAM

Home IdP

User is redirected to home IDP for authentication

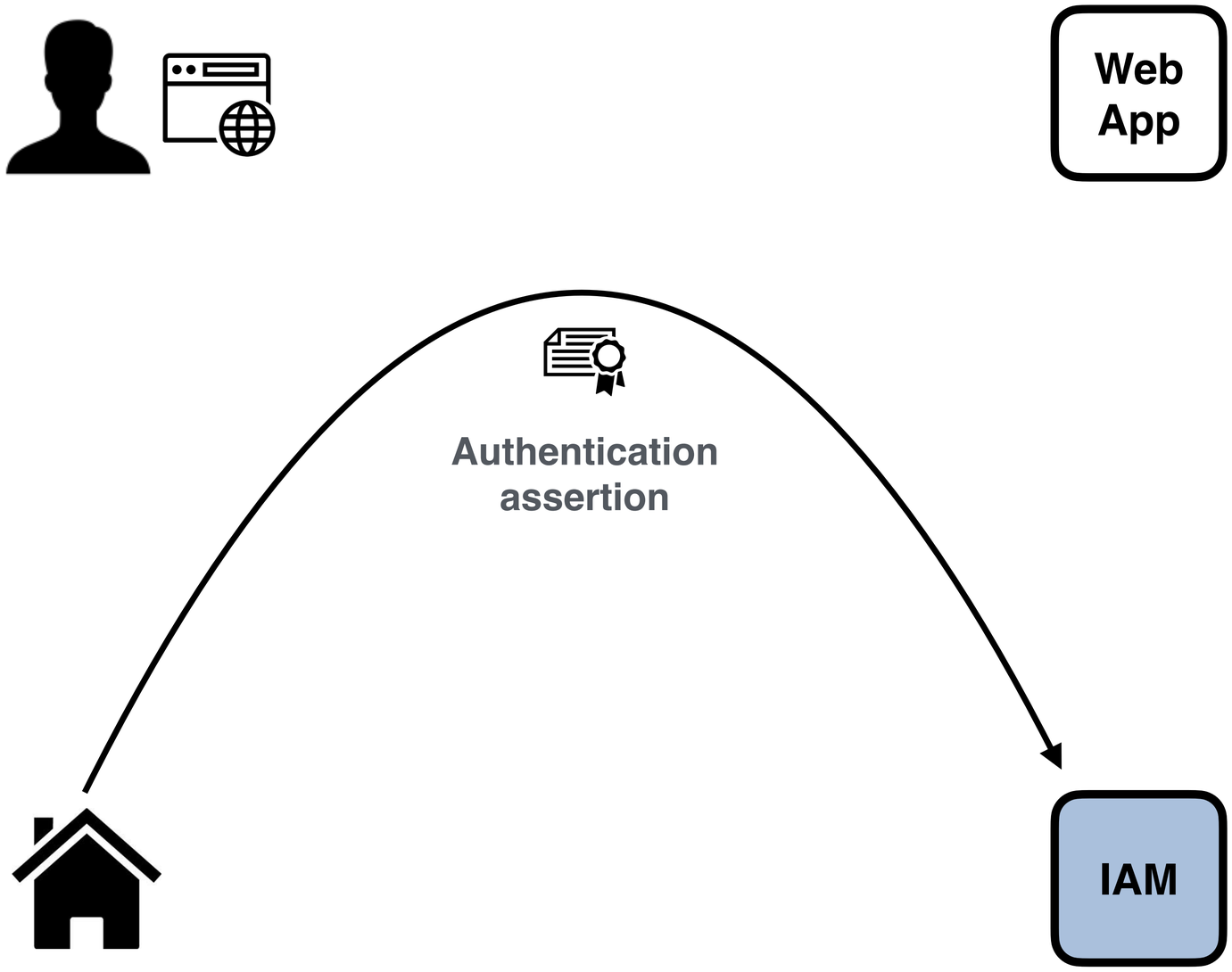
Web application: authorization code flow



cted to home IDP
entication

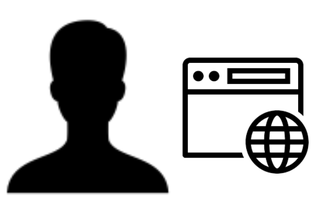
Home IdP

Web application: authorization code flow



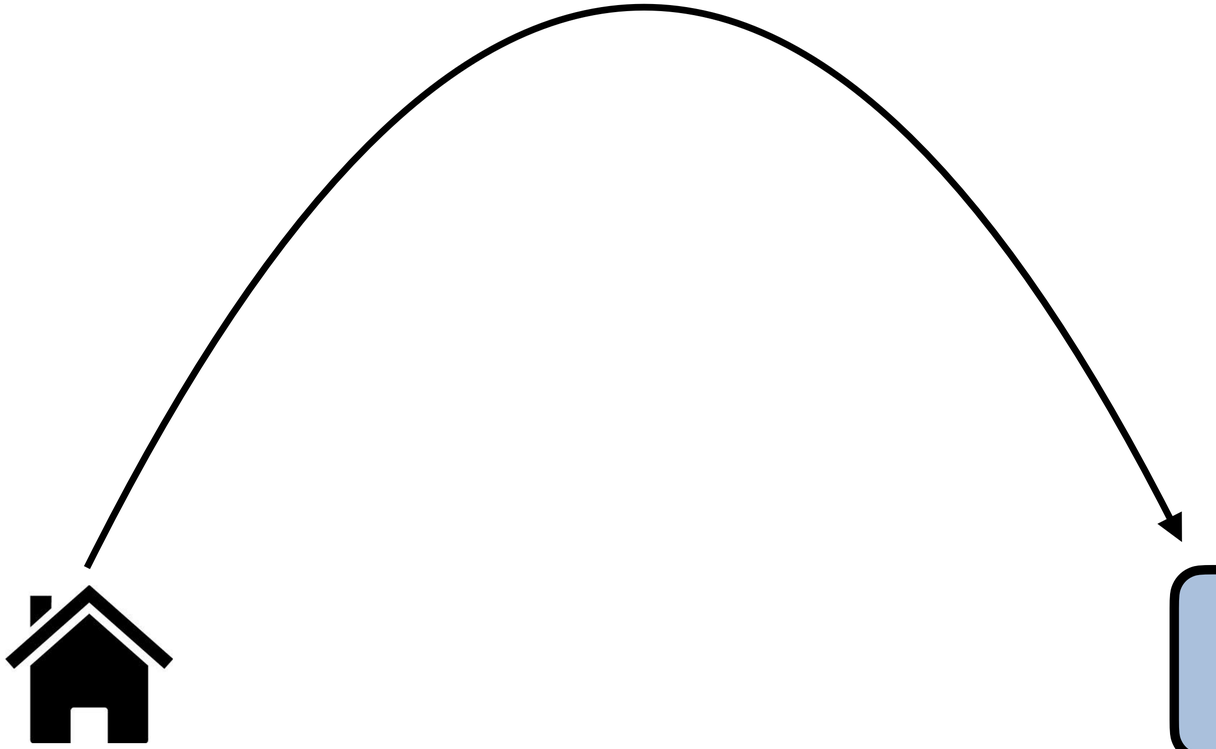
Home IDP authenticates user and sends back an authentication assertion, via redirection and possibly other interactions between IAM and the IDP

Web application: authorization code flow



Web App

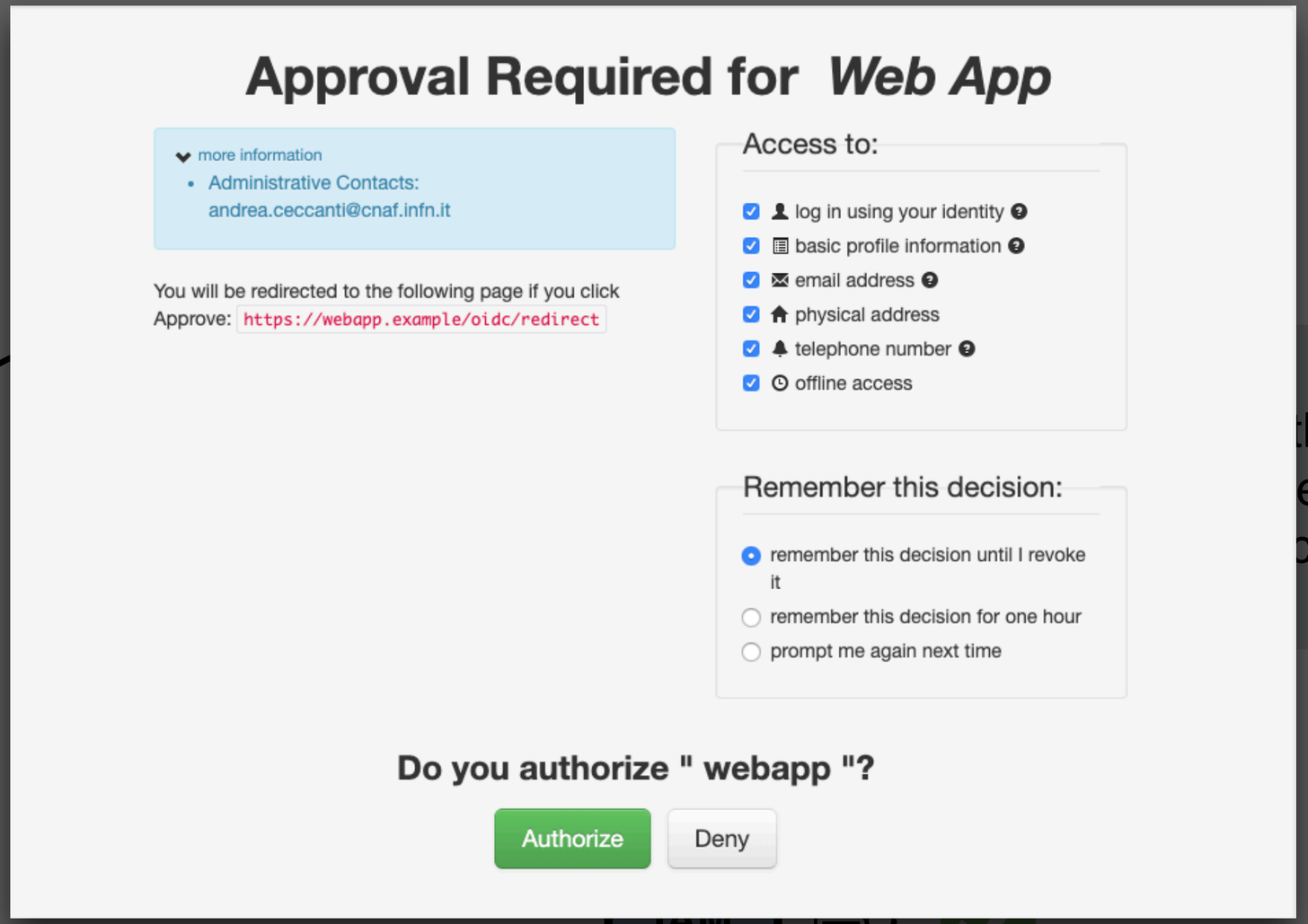
IAM validates the assertion,
the user is a registered one, so IAM shows a “Give consent” page



IAM

Home IdP

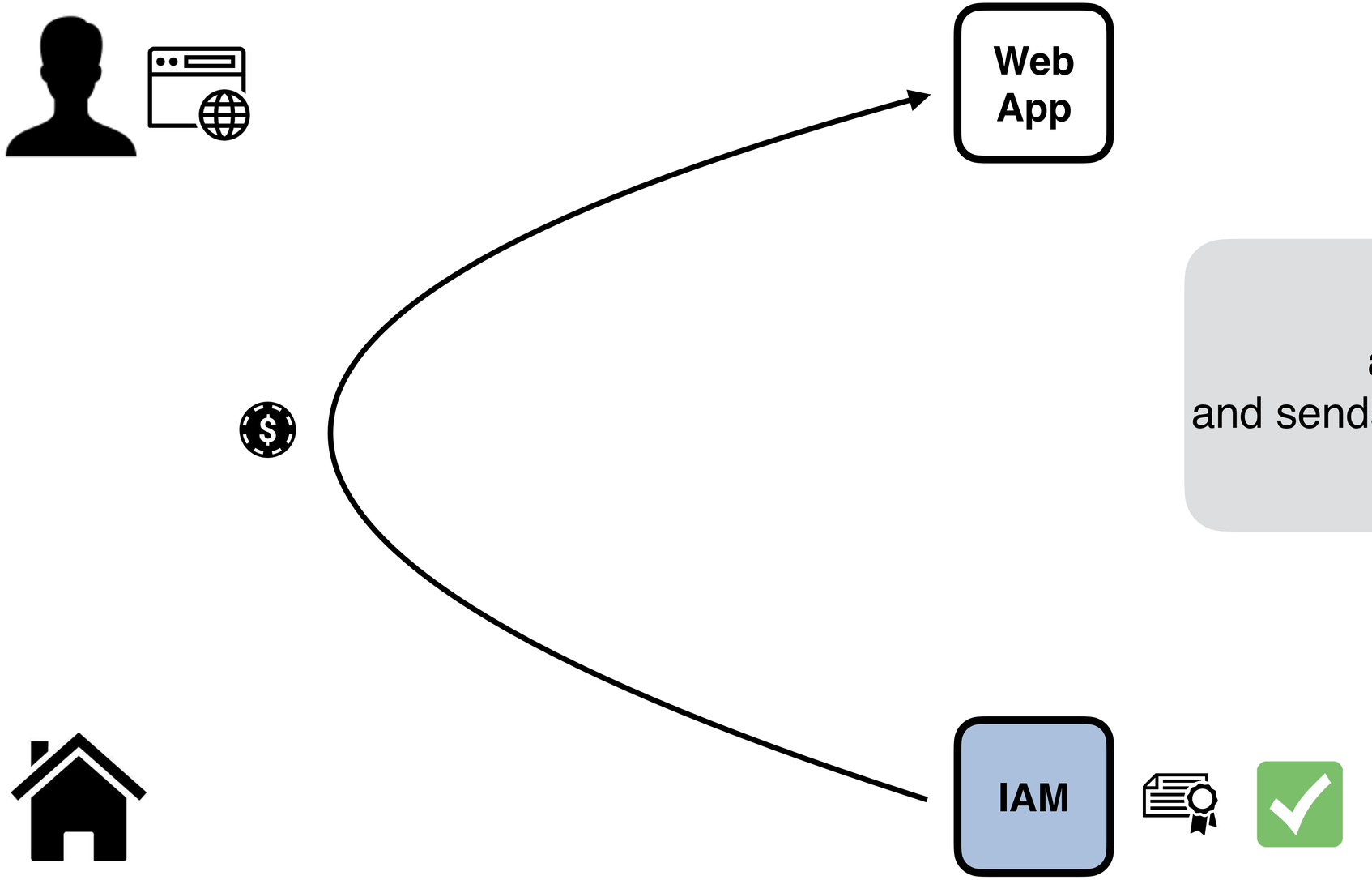
Web application: authorization code flow



the assertion,
ered one, so IAM
consent" page

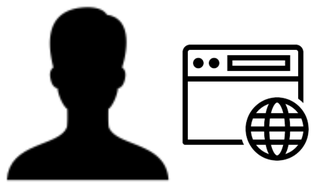
Home IdP

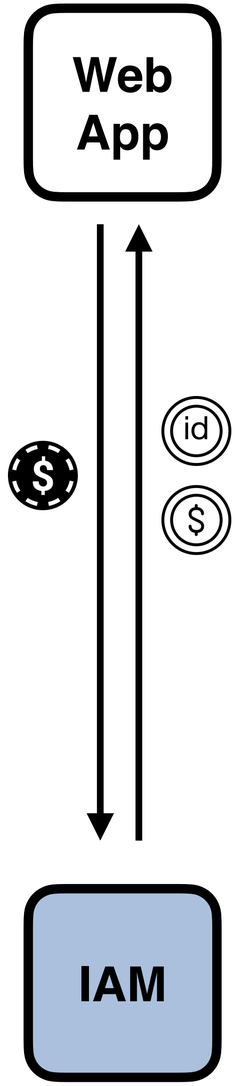
Web application: authorization code flow



IAM generates an
authorization code
and sends it back to web app using an HTTP redirect

Web application: authorization code flow

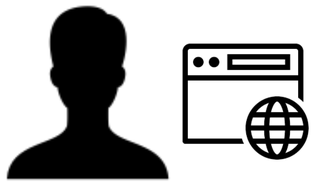


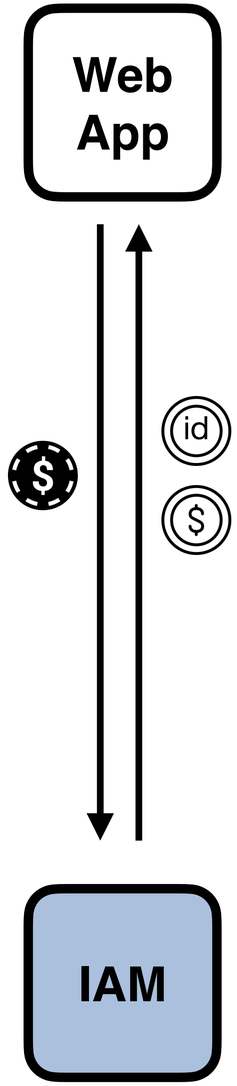


The Web App exchanges the **authorization code** with a couple of tokens: an **access token** and an **id token**

Home IdP

Web application: authorization code flow

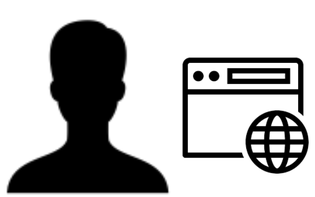


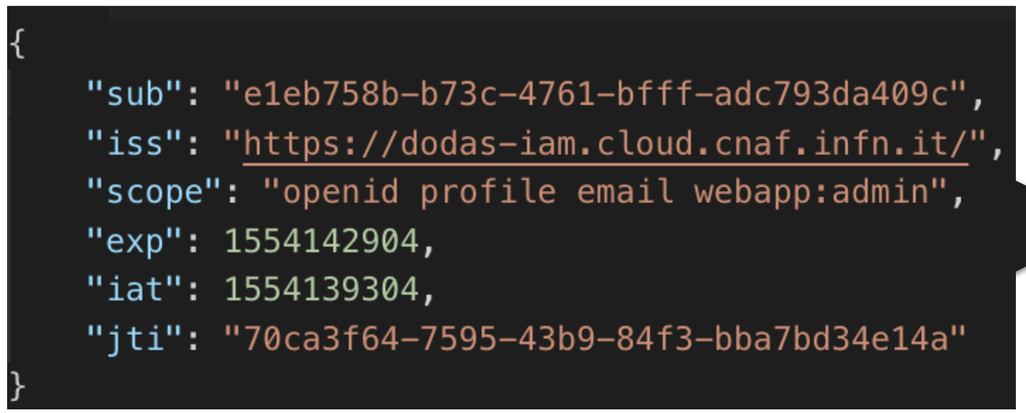


In IAM,
both tokens are
JWT tokens.

Home IdP

Web application: authorization code flow





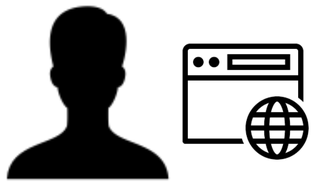
The **access token** provides (mainly) authorization information

Home IdP

Web application: authorization code flow

The **id token** provides (mainly) authentication information

Web application: authorization code flow

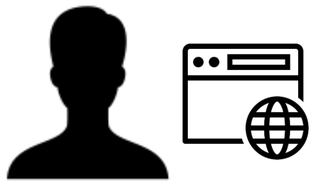


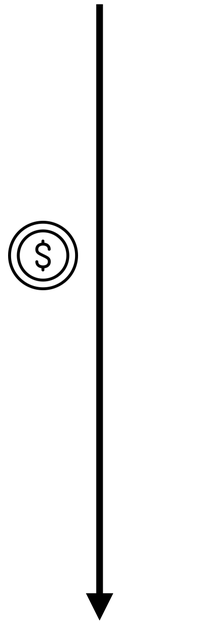
Both tokens are **validated** following to the JWT and OpenID Connect guidelines, checking **temporal validity**,
token signature, **audience**, etc…

Home IdP

IAM

Web application: authorization code flow





Additional information about the user can be requested by querying the **/userinfo** endpoint and providing the just obtained **access token** for authentication/authorization purposes

Home IdP

Authorization code flow in practice

- In practice, decent OAuth/OpenID Connect client libraries implement all the above **behind the scenes.**

- As an example, Apache mod_auth_openidc requires the following information to enable a working OpenID Connect integration
  - The OpenID Connect provider discovery/metadata URL
  - Client credentials

- The library then takes care of exchanging messages with the OpenID provider, implementing verification checks, and provides the obtained authentication/ authorization information to the protected web application
  - typically via env variables or HTTP headers

Funded by the European Union's
Horizon 2020 - Grant N° 824064

Refresh token flow

- Used by a client to refresh an access token that is about to expire using a refresh token obtained in a former authorization flow
- Authenticated call to the IAM/AS token endpoint
  - Produces a new access token and possibly an updated refresh toke
- The scope request parameter can be used to attenuate the token privileges, by requesting a subset of the scopes linked to user authorization grant

Funded by the European Union's
Horizon 2020 - Grant N° 824064

Refresh token flow request: example

```
curl -s -L \
   --user ${IAM_CLIENT_ID}:${IAM_CLIENT_SECRET} \
   -d grant_type=refresh_token \
   -d refresh_token=${REFRESH_TOKEN} \
   ${IAM_TOKEN_ENDPOINT}
```


Funded by the European Union's
Horizon 2020 - Grant N° 824064

OAuth/OpenID Connect provider metadata

- OAuth & OpenID Connect provide a standard way to expose the authorization server/OpenID provider configuration to clients

- Information is published at **a well-known endpoint** for the server, e.g.:
  - https://dodas-iam.cloud.cnaf.infn.it/**.well-known/openid-configuration**

- Clients can use this information to know about
  - location of key material used to sign/encrypt tokens
  - supported grant types/authorization flows
  - endpoint locations
  - supported claims
  - …

- and implement **automatic client configuration**

Funded by the European Union's
Horizon 2020 - Grant N° 824064

OAuth/OpenID Connect provider metadata

Example metadata document:

https://wlcg.cloud.cnaf.infn.it/.well-known/openid-configuration

Funded by the European Union's
Horizon 2020 - Grant N° 824064

OAuth/OIDC scopes

- OAuth provides **scopes** as a standard mechanism to express authorization permissions granted to client applications

- In practice, scopes are a set of strings included in an access token that limit what are the operations that can be authorized by clients presenting such access token
  - User consent is based on scopes requested

- OAuth scopes are commonly used in industry to define the authorization on service APIs. Examples:
  - https://api.slack.com/docs/oauth-scopes
  - https://developer.github.com/apps/building-oauth-apps/understanding-scopes-for-oauth-apps/#available-scopes
  - https://developers.google.com/identity/protocols/googlescopes

Funded by the European Union's Horizon 2020 - Grant N° 824064

Standard commonly used OAuth/OIDC scopes

- `openid`: signals that the client wants to receive authentication information about the user
- `profile`: used to request profile information (name, address and other information)
- `email`: used to request access to the user's email (name, address)
- `offline_access`: used to request refresh tokens

Funded by the European Union's
Horizon 2020 - Grant N° 824064

WLCG profile OAuth/OIDC scopes

- `wlcg.groups`: used to request the inclusion of group information in tokens

- `storage.read`, `storage.modify`, `storage.create`: these scopes are used to manage access to WLCG storage

- `compute.read`, `compute.modify`, `compute.create`, `compute.cancel`: these scopes are used to manage access to WLCG computing resources

Funded by the European Union's
Horizon 2020 - Grant N° 824064

OAuth bearer token usage

- There's a standard that defines how to send tokens to resource servers
- Typically, tokens are sent in the **`Authorization`** HTTP header, following the rules defined in RFC 6750, as in the following example HTTP request

```
GET / HTTP/1.1
Host: apache.test.example
Authorization: Bearer eyJraWQiOiJy…rYI
User-Agent: curl/7.65.3
Accept: */*
```

Funded by the European Union's
Horizon 2020 - Grant N° 824064

JSON Web Tokens: definition

Citing RFC 7519:

- **JSON Web Token** (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties.

- The **claims** in a JWT are **encoded as a JSON object** that is used as the payload of a JSON Web Signature (JWS) structure OR as the plaintext of a JSON Web Encryption (JWE) structure, **enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted**.

Funded by the European Union's
Horizon 2020 - Grant N° 824064

JWT

Citing RFC 7519:

- A JWT is represented as a sequence of **URL-safe parts** separated by period ('.') characters. Each part contains a **base64url-encoded value**.

- The number of parts in the JWT is dependent upon the representation of the resulting JSON Web Signature (JWS) using the JWS Compact Serialization or JSON Web Encryption (JWE) using the JWE Compact Serialization.

Funded by the European Union's
Horizon 2020 - Grant N° 824064

JWT: Header.Body.Signature


```
eyJraWQiOiJyc2ExIiwiYWxnIjoiUlMyNTYifQ.eyJ3bGNnLnZlciI6IjEuMCIsInN
1YiI6IjI0MTY4N2U4LTUzNzQtNDU0OS1iOWY2LWEzODY2ZjBiZjZkYSIsImF1ZCI6I
mh0dHBzOlwvXC93bGNnLmNlcm4uY2hcL2p3dFwvdjFcL2FueSIsIm5iZiI6MTYxMDk
4MzAzOCwic2NvcGUiOiJvcGVuaWQgcHJvZmlsZSB3bGNnLmdyb3VwcyIsImlzcyI6I
mh0dHBzOlwvXC9pYW0tZXNjYXBlLmNsb3VkLmNuYWYuaW5mbi5pdFwvIiwiZXhwIjo
xNjEwOTg2NjM4LCJpYXQiOjE2MTA5ODMwMzgsImp0aSI6IjA5NjIwZTQ3LWE5NTQtN
GZjNS1hMzMxLTE1NDBiMmU0MjYzYyIsImNsaWVudF9pZCI6IjEyMDIwYjM1LTQ0ZTI
tNDljYS1hODU2LWQwNzE2OTUyNzkwZCIsIndsY2cuZ3JvdXBzIjpbIlwvZXNjYXBlI
iwiXC9lc2NhcGVcL2NtcyIsIlwvZXNjYXBlXC9waWxvdHMiLCJcL2VzY2FwZVwveGZ
lcnMiXX0.b64QOAjMoQfcJtin6hTLxtUepqjbbZ9pmb4xp5MoXeM3d4TyY1OIyQtcg
eZl4_mAzc22thTLbtu675xM7LswfrqFdc9eNPqi2VQzpdYae4S-
bK_3r9Dev-8o7PKiHNLtytNTK6Djre8WQF2TUX-oHsDqP2EJDskuqu-GAdhjLVI
```


Funded by the European Union's
Horizon 2020 - Grant N° 824064

JWS compact serialization form

- From https://tools.ietf.org/html/rfc7515#section-3.1

In the JWS Compact Serialization, a JWS is represented as the concatenation:

BASE64URL(UTF8(JWS Protected Header)) || '.' ||

BASE64URL(JWS Payload) || '.' ||

BASE64URL(JWS Signature)

Funded by the European Union's
Horizon 2020 - Grant N° 824064

JWT: Header.Body.Signature

Header

```
{
  "kid": "rsa1",
  "alg": "RS256"
}
```

Body

```
{
  "wlcg.ver": "1.0",
  "sub": "241687e8-5374-4549-b9f6-a3866f0bf6da",
  "aud": "https://wlcg.cern.ch/jwt/v1/any",
  "nbf": 1610983038,
  "scope": "openid profile wlcg.groups",
  "iss": "https://iam-escape.cloud.cnaf.infn.it/",
  "exp": 1610986638,
  "iat": 1610983038,
  "jti": "09620e47-a954-4fc5-a331-1540b2e4263c",
  "client_id": "12020b35-44e2-49ca-a856-d0716952790d",
  "wlcg.groups": [
    "/escape",
    "/escape/cms",
    "/escape/pilots",
    "/escape/xfers"
  ]
}
```

Signature

```
b64QOAjMoQfcJtin6hTLxtUep
qjbbZ9pmb4xp5MoXeM3d4TyY1
OIyQtcgeZl4_mAzc22thTLbtu
675xM7LswfrqFdc9eNPqi2VQz
pdYae4S-
bK_3r9Dev-8o7PKiHNLtytNTK
6Djre8WQF2TUX-
oHsDqP2EJDskuqu-GAdhjLVI
```


Funded by the European Union's Horizon 2020 - Grant N° 824064

JWT claim names

- Registered claim names (i.e. a set of basic claims defined by the JWT standard
  - “iss” (Issuer): the principal that issued the JWT (e.g., IAM ESCAPE)
  - “sub” (Subject): the principal that is the subject of the JWT (e.g., a unique id linked to an IAM account)
  - “aud” (Audience): identifies the recipients that the JWT is intended for (e.g., RUCIO)
  - “exp” (Expiration time): identifies the expiration time on or after which the JWT MUST NOT be accepted for processing
  - “nbf” (Not before): identifies the time before which the JWT MUST NOT be accepted for processing
  - “iat” (Issued at): identifies the time at which the JWT was issued
  - “jti” (JWT ID): provides a unique identifier for the JWT
- Public claim names
  - Either a registered public claim name or one that has a collision-resistant name
- Private claim names
  - Claim names that are not registered or public (i.e. are not collision-resistant)

Funded by the European Union’s
Horizon 2020 - Grant N° 824064

The WLCG JWT profile

The WLCG JWT profile

How is **authentication** and **authorization** information encoded in **identity** and **access tokens**?

How is **trust** established between parties exchanging tokens?

What's the recommended **token lifetime?**

Approach:

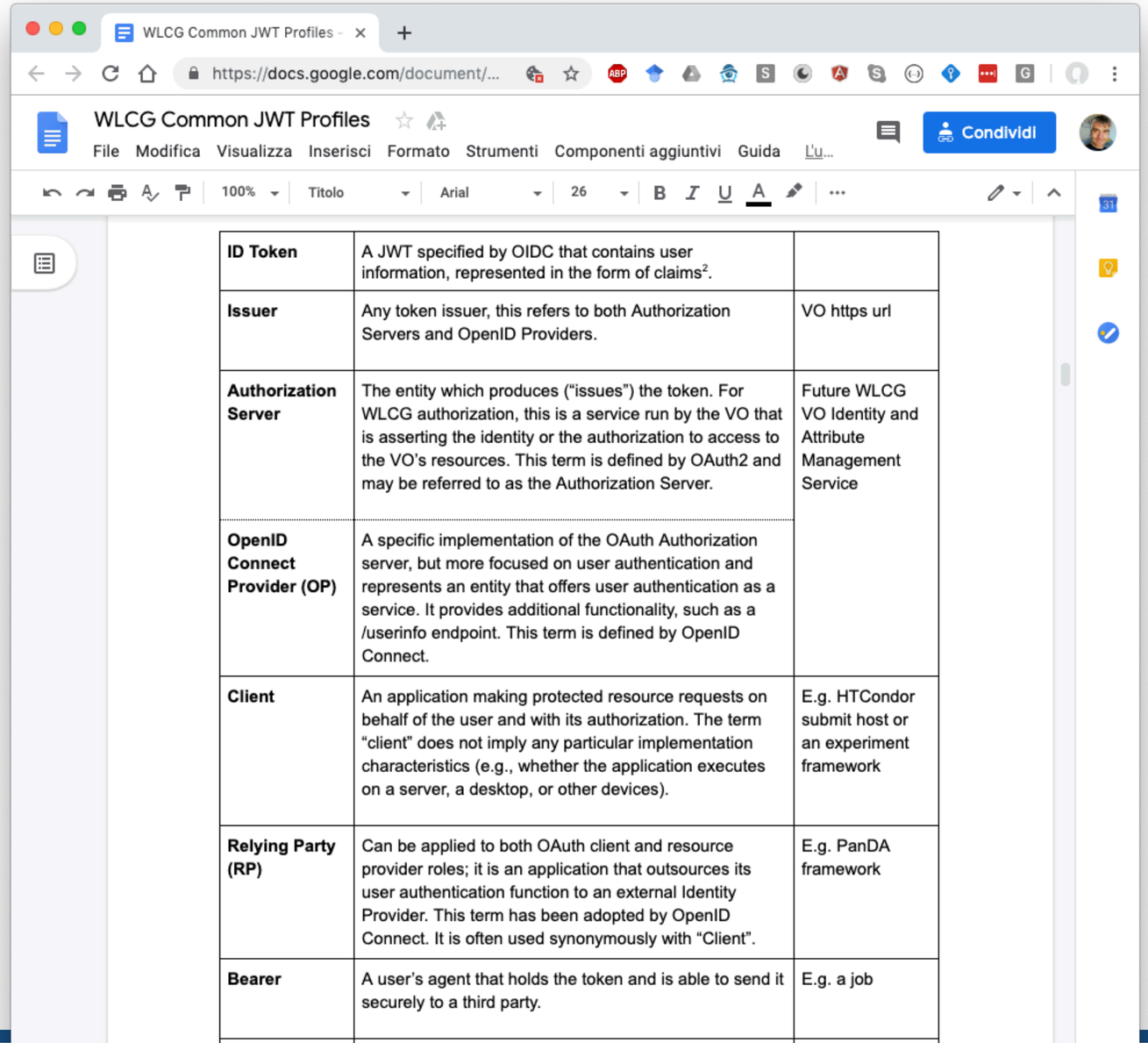
rely on existing standards as much as possible, extend only when needed

WLCG JWT profile: glossary

Define common terms and meaning

Leverage standard definitions wherever possible

Map general concepts to our use cases



ID Token	A JWT specified by OIDC that contains user information, represented in the form of claims[2].	
Issuer	Any token issuer, this refers to both Authorization Servers and OpenID Providers.	VO https url
Authorization Server	The entity which produces (“issues”) the token. For WLCG authorization, this is a service run by the VO that is asserting the identity or the authorization to access to the VO’s resources. This term is defined by OAuth2 and may be referred to as the Authorization Server.	Future WLCG VO Identity and Attribute Management Service
OpenID Connect Provider (OP)	A specific implementation of the OAuth Authorization server, but more focused on user authentication and represents an entity that offers user authentication as a service. It provides additional functionality, such as a /userinfo endpoint. This term is defined by OpenID Connect.	
Client	An application making protected resource requests on behalf of the user and with its authorization. The term “client” does not imply any particular implementation characteristics (e.g., whether the application executes on a server, a desktop, or other devices).	E.g. HTCondor submit host or an experiment framework
Relying Party (RP)	Can be applied to both OAuth client and resource provider roles; it is an application that outsources its user authentication function to an external Identity Provider. This term has been adopted by OpenID Connect. It is often used synonymously with “Client”.	E.g. PanDA framework
Bearer	A user’s agent that holds the token and is able to send it securely to a third party.	E.g. a job

WLCG JWT profile: token claims

What are the **required** claims to be included in access tokens and ID tokens, and what is the meaning.

Common claims: claims commons to access and ID tokens

ID token claims: claims specific to ID tokens (mainly focusing on user authentication and identity)

Access token claims: claims specific to access tokens (mainly focusing on authorization capabilities or attributes)

The profile mostly **reuses existing, standard** claims, with some WLCG specific additions. Additional, application-specific claims are allowed

WLCG specific token claims

wlcg.ver: the version of the WLCG token profile the relying party must understand to validate the token. Example:

```
wlcg.ver = “WLCG:1.0”
```

wlcg.groups: group information about an authenticated end-user, following a UNIX-like path syntax. Example:

```
wlcg.groups = {“/cms”, “/cms/itcms”}
```

Other claims used in the profile come from JWT and OpenID connect core standard

Scope-based authorization

OAuth provides **scopes** as a standard mechanism to express authorization permissions granted to client applications.

In practice, scopes are a set of strings included in an access token that **limit what are the operations that can be authorized by clients presenting such access token.**

OAuth scopes are commonly used in industry to define the authorization on service APIs. Examples:

https://api.slack.com/docs/oauth-scopes

https://developer.github.com/apps/building-oauth-apps/understanding-scopes-for-oauth-apps/#available-scopes

https://developers.google.com/identity/protocols/googlescopes

WLCG OAuth scopes

Building on the SciTokens experience, define scopes that would match our computing use-cases.

First use case: **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.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.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.stage: Cause data to be staged from a nearline resource to an online resource.

Storage scopes and resource paths

Storage scopes may additionally provide a resource path*, which further limits the authorization. The resource path is provided respecting the following format:

scope:path

Examples:

storage.read:/

storage.modify:/protected

* the path is required for selected scopes, more details in the profile

Path semantics

Following the Scitokens model, permissions granted on a path **apply transitively to subpaths,** e.g.:

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.

This approach is **not equivalent** with POSIX semantics, but matches well with our experiments data access authorization models.

Path semantics

Following the Scitokens model, permissions granted on a path **apply transitively to subpaths,** e.g.:

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.

This approach is **not equivalent** with POSIX semantics, but matches well with our experiments data access authorization models.

Note that implementing this semantic is up to client applications, i.e. dCache, DPM, EOS, StoRM, XRootD, etc…., the token just provides a (signed) string!

Scope-based group selection

Use scopes to implement a group selection mechanism for groups equivalent to the one provided by VOMS, following the approach outlined in the OpenID Connect standard.

Two types of groups:

- **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)

Scope-based group selection

A parametric `wlcg.groups` scope is introduced with the following form:

`wlcg.groups[:<group_name>]?`

With 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 for the target client.
- if the scope is parametric, i.e. it has the form `wlcg.groups:<group_name>`, in addition to the default groups as described in the previous point, the authorization server will also return the requested group as a value in the `wlcg.groups` claim if the user is member of such group.

Scope-based group selection

…with the following rules:

- To request multiple groups, multiple `wlcg.groups:<group_name>` scopes are included in the authorization request
- The order of the groups in the returned `wlcg.groups` claim complies with the order in which the groups were requested
- the returned groups claim will not contain duplicates

This seems complex, but it’s the attribute selection mechanism we use everyday with VOMS

Scope-based group selection

…with the following rules:

- To request multiple groups, multiple `wlcg.groups:<group_name>` scopes are included in the authorization request
- The order of the groups in the returned `wlcg.groups` claim complies with the order in which the groups were requested
- the returned groups claim will not contain duplicates

This seems complex, but it's the attribute selection mechanism we use everyday with VOMS

Note that implementing this semantic is (mostly) up to the WLCG AuthZ server (i.e., IAM).

Scope-based group selection: examples

An authorization request with the following scope:

```
scope=wlcg.groups:/cms/uscms wlcg.groups:/cms/ALARM wlcg.groups
```

will return the following `wlcg.groups` claim

```
"wlcg.groups": ["/cms/uscms","/cms/ALARM", “/cms"]
```

assuming `/cms` is the only default group defined at the authorization server

Trust & security

The profile document also provides recommendations on token lifetimes and 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 there is no 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”.
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.

* Slide courtesy of H. Short

Supporting the WLCG JWT profile

What does it mean supporting the WLCG profile?

Depends on the **role** of your service:

- OAuth resource server
  - The typical example is an **HTTP Restful API**
  - Does not need the ability to start an OAuth/OpenID Connect authentication flow
  - Does not need to be registered in IAM
  - Needs to extract token from incoming requests and validate token and map authn/authz info in the token to local authz enforcement

- OAuth/OpenID Connect client:
  - The typical example is a **Web application (a portal)** that wants to delegate authentication to IAM
  - Needs to be registered in IAM
  - Needs the ability to start OAuth/OpenID Connect auhn/z flow, store securely client credentials, validate tokens, refresh them when needed …

- Some services will naturally fit in **both roles** defined above
  - e.g., RUCIO, FTS, dCache

What does it mean supporting the WLCG profile?

As an **OAuth resource server** (RS):

- Ability to extract an access token from an incoming HTTP request
- Ability to parse and validate the incoming access token
  - identify if it has been issue by a trusted and recognized authorization server
  - verify temporal validity
  - verify signature, following OAuth/OIDC conventions
- Ability to honour access token audience restrictions
  - the RS needs the ability to identity itself with (one or multiple) audience labels and honour audience restrictions in access tokens
- Ability to map defined scopes to local authZ
  - e.g., storage.read:/folder on a storage area grants read access to the /folder part of the namespace (including sub-directories)
- Ability to map group-based to local authZ
  - e.g., /cms group membership grants read access to the /cms namespace

What does it mean supporting the WLCG profile?

As an **OAuth resource server** (RS):

- Ability to extract an access token from an incoming HTTP request
- Ability to parse and validate the incoming access token
  - identify if it has been issue by a trusted and recognized authorization server
  - verify temporal validity
  - verify signature, following OAuth/OIDC conventions
- Ability to honour access token audience restrictions
  - the RS needs the ability to identity itself with (one or multiple) audience labels and honour audience restrictions in access tokens

This is typically sorted out by OAuth/OIDC libraries

- Ability to map defined scopes to local authZ
  - e.g., storage.read:/folder on a storage area grants read access to the /folder part of the namespace (including sub-directories)
- Ability to map group-based to local authZ
  - e.g., /cms group membership grants read access to the /cms namespace

What does it mean supporting the WLCG profile?

As an **OpenID Connect client**:

- Ability to store client credentials securely
- Ability to start and manage an OAuth/OpenID Connect flow to obtain tokens from the Authorization Server (i.e., IAM)
  - Authorization code flow, for most use cases
  - Refresh token flow, to refresh access tokens about the expire
  - Client credentials flow, to obtain tokens linked not linked to user identities, but to the service itself
- Ability to parse and validate ID tokens resulting from OpenID Connect authentication flows in compliance with the OpenID connect spec
- Ability to honour audience restrictions
  - the ability to identity itself with (one or multiple) audience labels and honour audience restrictions in ID tokens
- (Optional) Ability to implement Level Of Assurance (LoA) policies

What does it mean supporting the WLCG profile?

This is typically sorted out by OAuth/OIDC libraries

As an **OpenID Connect client**:

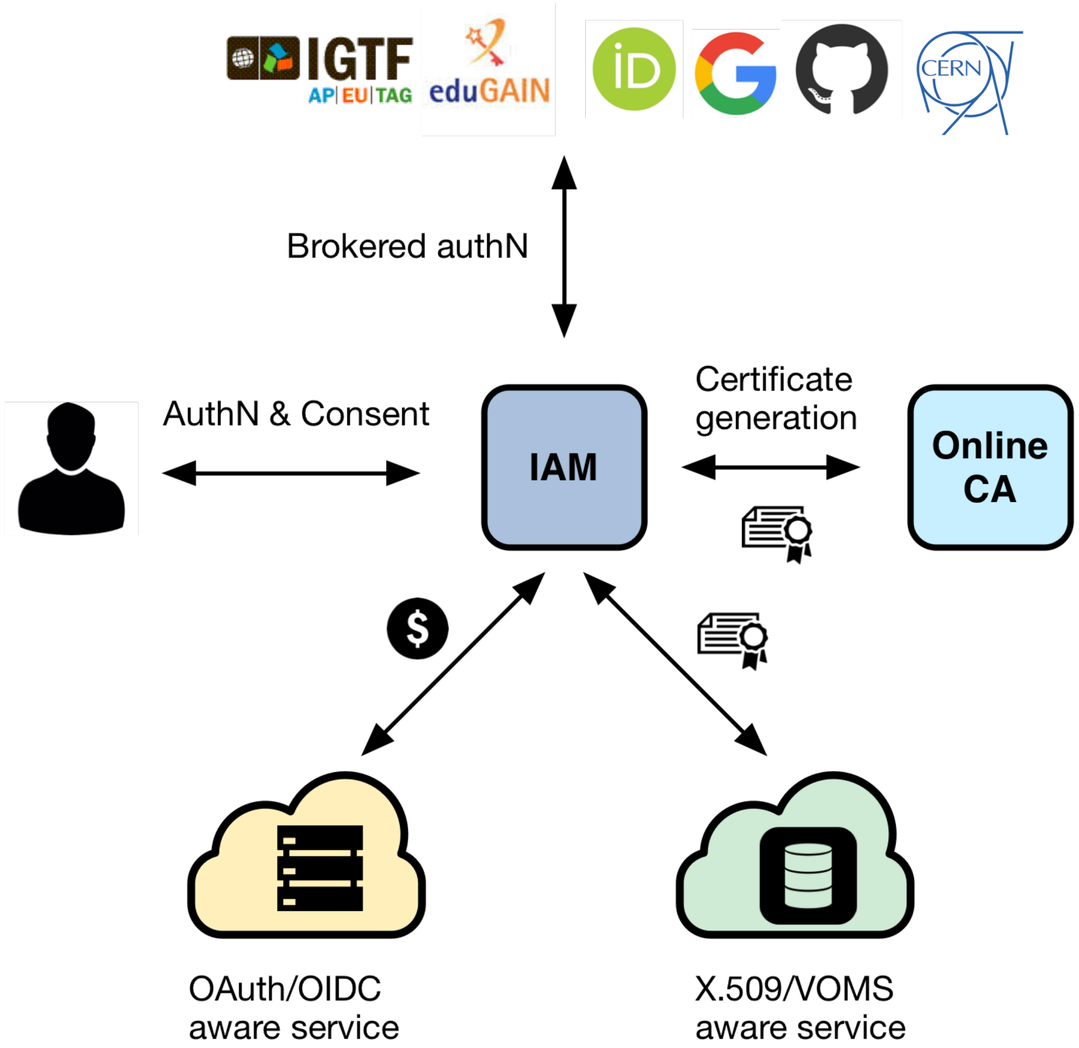
- Ability to store client credentials securely
- Ability to start and manage an OAuth/OpenID Connect flow to obtain tokens from the Authorization Server (i.e., IAM)
  - Authorization code flow, for most use cases
  - Refresh token flow, to refresh access tokens about the expire
  - Client credentials flow, to obtain tokens linked not linked to user identities, but to the service itself
- Ability to parse and validate ID tokens resulting from OpenID Connect authentication flows in compliance with the OpenID connect spec
- Ability to honour audience restrictions
  - the ability to identity itself with (one or multiple) audience labels and honour audience restrictions in ID tokens
- (Optional) Ability to implement Level Of Assurance (LoA) policies

INDIGO IAM
(in a bit more detail)

INDIGO Identity and Access Management Service

An authentication and authorization service that

- supports **multiple authentication mechanisms**
- provides users with a **persistent, organization scoped** identifier
- exposes **identity information**, **attributes** and **capabilities** to services via **JWT** tokens and standard **OAuth & OpenID Connect** protocols
- can integrate existing **VOMS**-aware services
- supports **Web** and **non-Web access**, **delegation** and **token renewal**

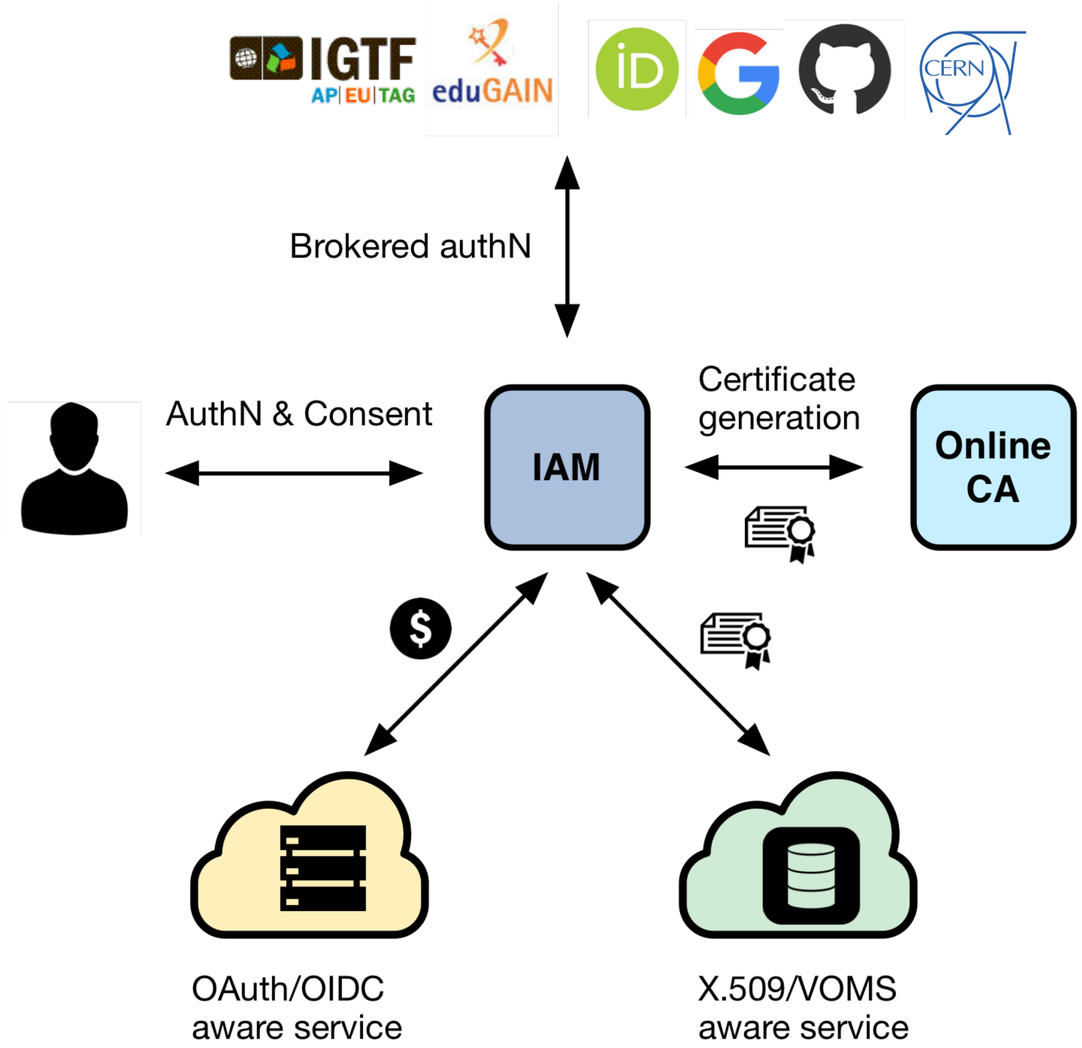


INDIGO Identity and Access Management Service

First developed in the context of the **H2020 INDIGO DataCloud** project

Selected by the WLCG management board to be the core of the future, token-based WLCG AAI

Sustained by INFN for the foreseeable future, with current support from:

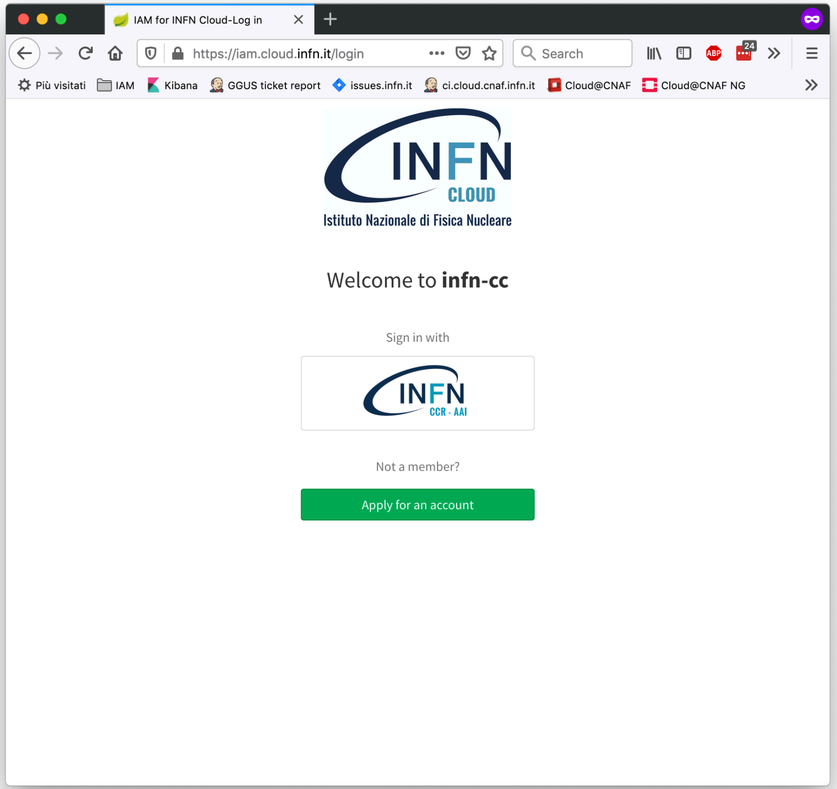


IAM deployment model

An IAM instance is deployed for a **community** of users sharing resources, the good old **Virtual Organization (VO)** concept.

Client applications and services are integrated with this instance via **standard OAuth/OpenID Connect** mechanisms.

The IAM Web appearance can be **customized** to include a **community logo**, **AUP** and **privacy policy** document.

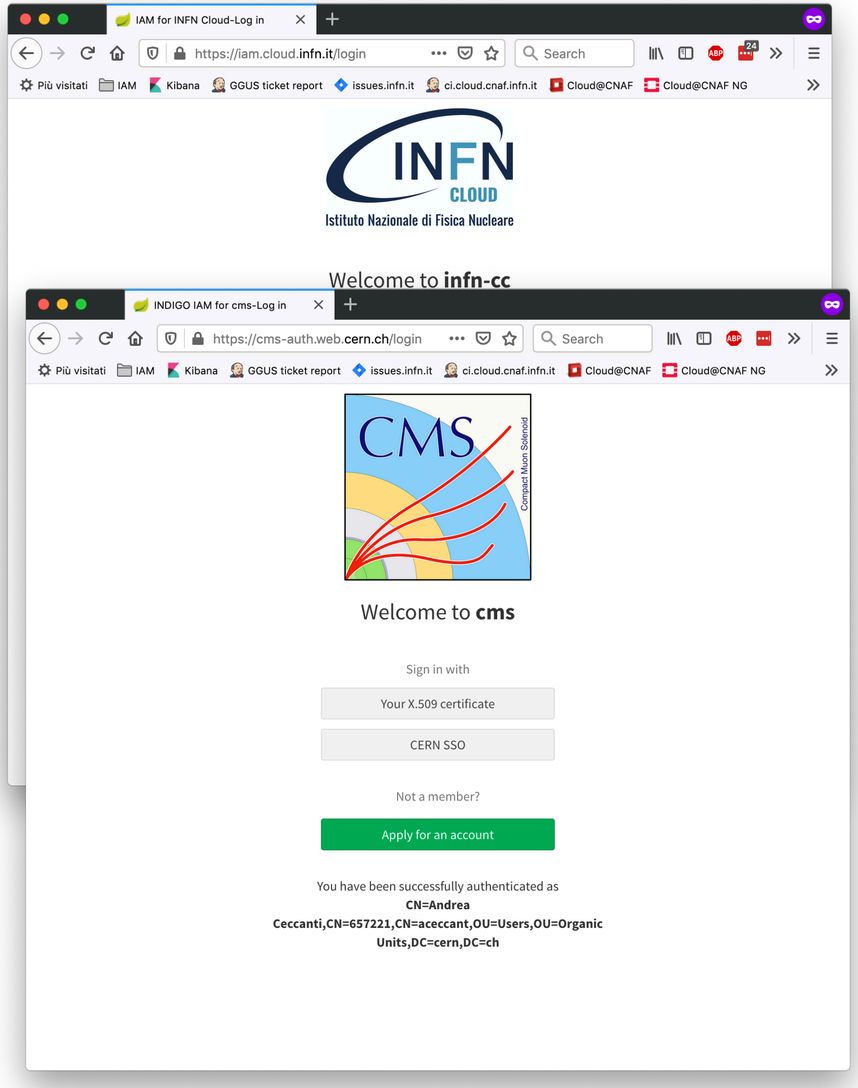


IAM deployment model

An IAM instance is deployed for a **community** of users sharing resources, the good old **Virtual Organization (VO)** concept.

Client applications and services are integrated with this instance via **standard OAuth/OpenID Connect** mechanisms.

The IAM Web appearance can be **customized** to include a **community logo**, **AUP** and **privacy policy** document.

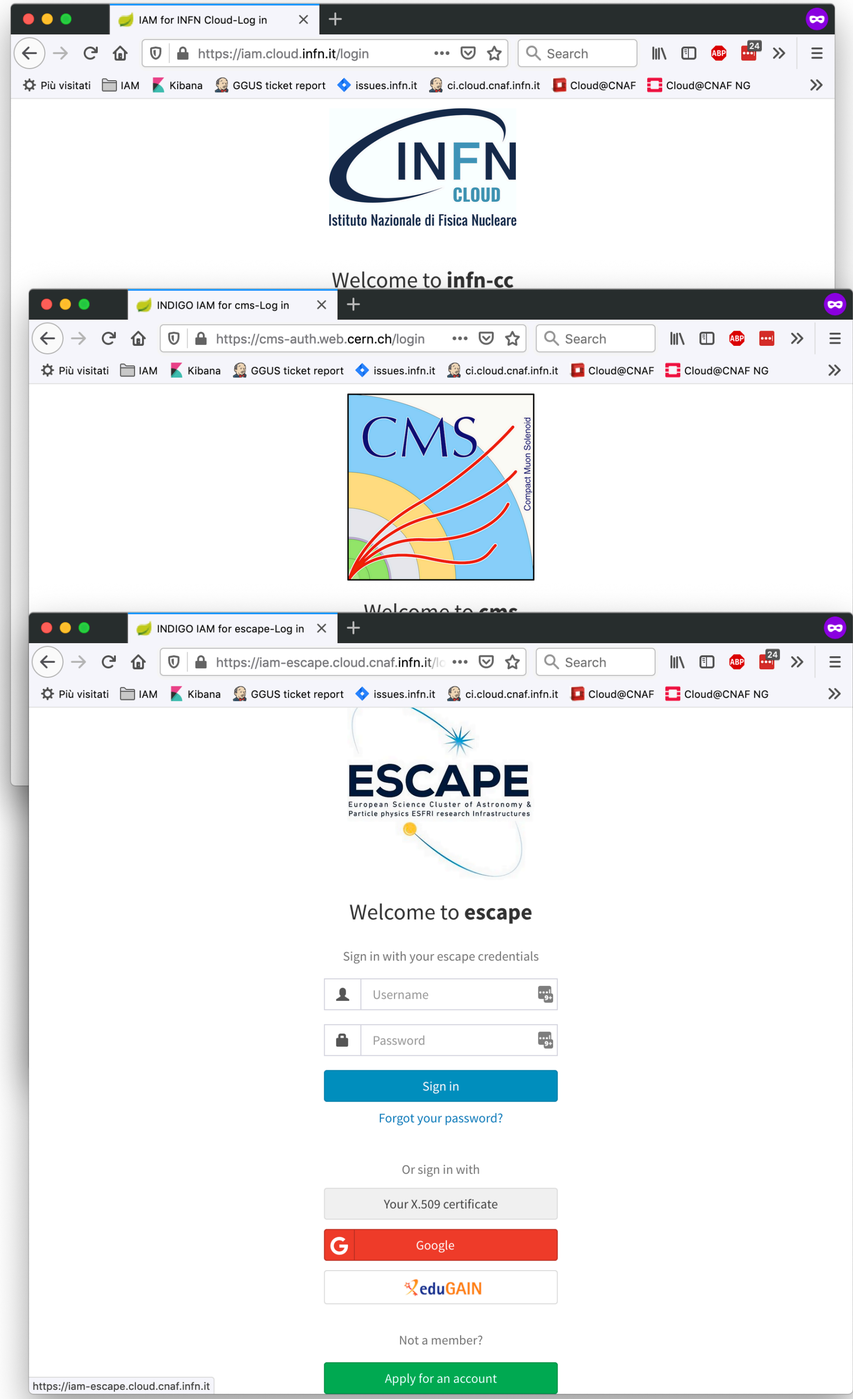


IAM deployment model

An IAM instance is deployed for a **community** of users sharing resources, the good old **Virtual Organization (VO)** concept.

Client applications and services are integrated with this instance via **standard OAuth/OpenID Connect** mechanisms.

The IAM Web appearance can be **customized** to include a **community logo**, **AUP** and **privacy policy** document.



User enrolment & registration service

IAM currently supports two **enrolment flows:**

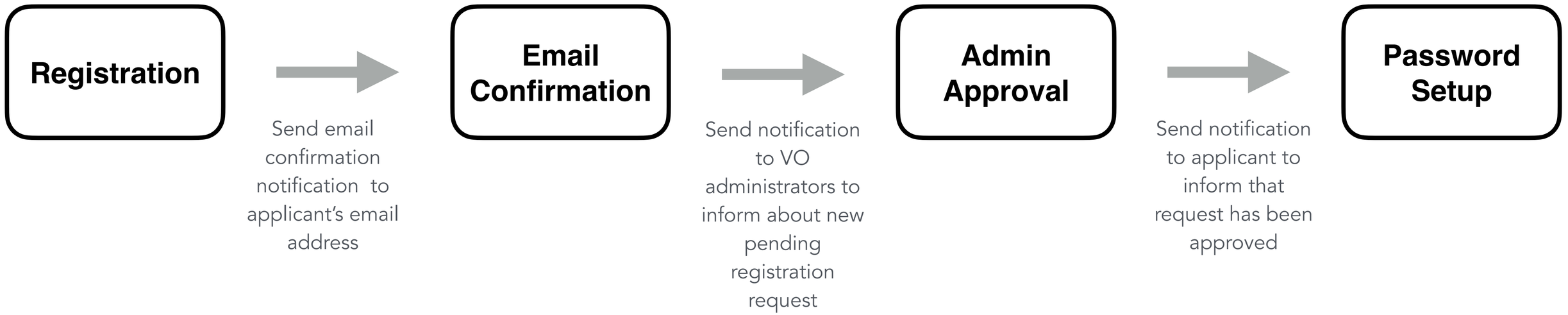
Admin-moderated flow

- The applicant fills basic registration information, accepts AUP, proves email ownership
- VO administrators are informed by email and can approve or reject incoming membership requests
- The applicant is informed via email of the administrator decision

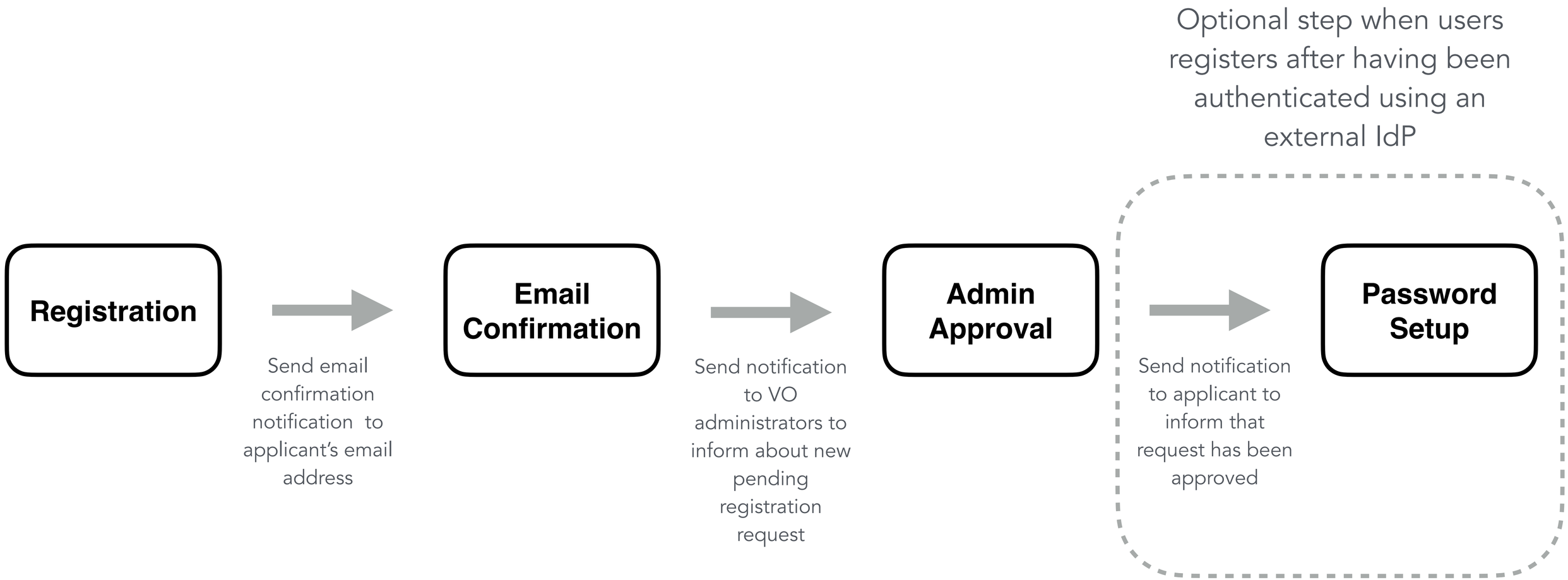
Automatic-enrolment flow

- Users authenticated at **trusted**, **configurable** IdPs are automatically on-boarded, without requiring administrator approval

IAM moderated enrolment flow



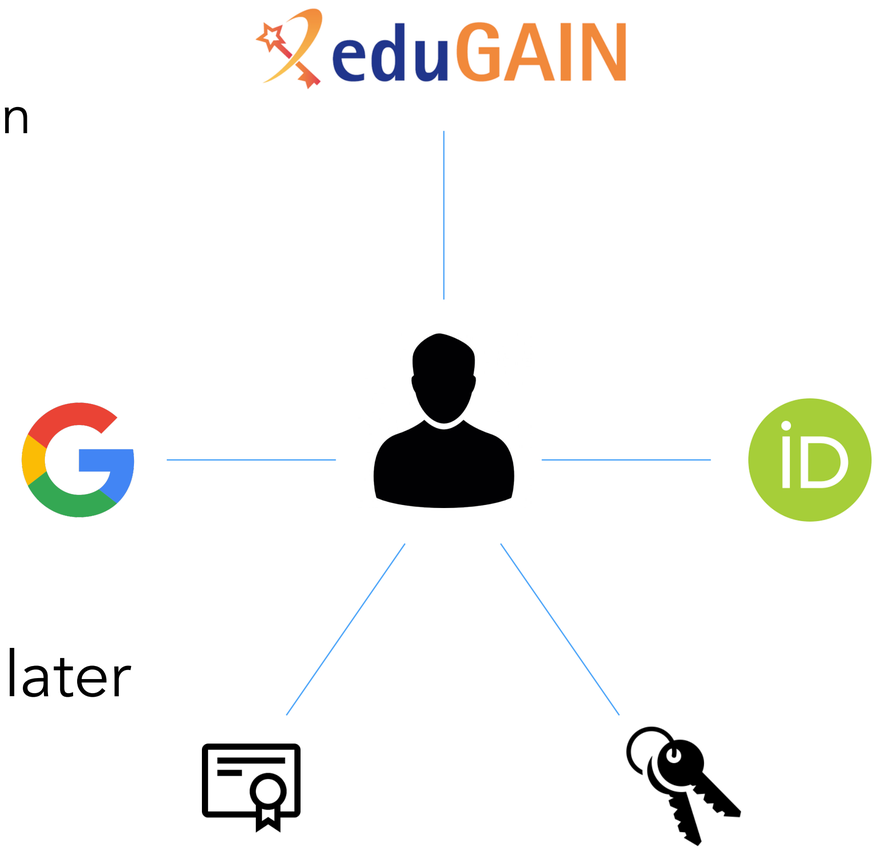
IAM moderated enrolment flow



Flexible authentication & account linking

Authentication supported via

- **local username/password** credentials (created at registration time)
- **SAML** Home institution IdP (e.g., EduGAIN)
- **OpenID Connect** (Google, Microsoft, Paypal, ORCID)
- **X.509** certificates



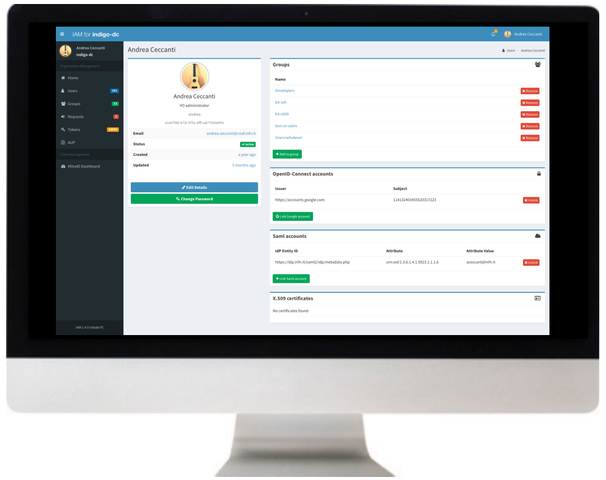
Users can link any of the supported authentication credentials to their IAM account at registration time or later

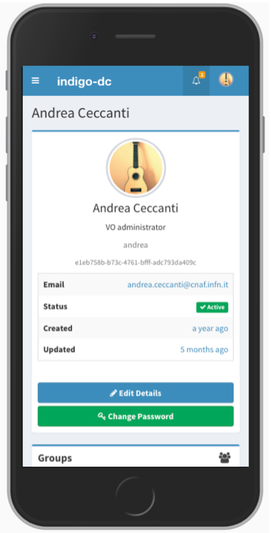
To link an external credential/account, the user has to **prove** that he/she owns such account

Management tools

IAM provides a **mobile-friendly** dashboard for:

- User management
- Group management
- Membership request management
- Account linking and personal details editing
- Token management



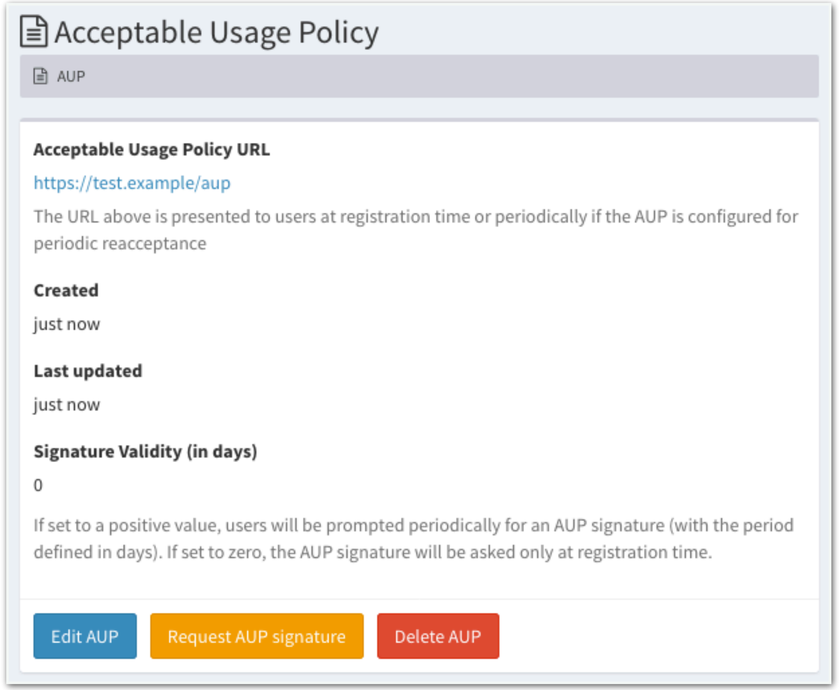


All management functionality is also exposed by REST APIs

AUP enforcement support

AUP acceptance, if enabled, can be configured to be:

- requested once at user registration time
- periodically, with configurable period



User cannot login to the system (and as such be authenticated at authorized at services) unless the **AUP** has been accepted

SCIM provisioning APIs

IAM provides a RESTful API, based on the System for Cross-domain Identity Management (SCIM) standard, that can be used to access information in the IAM database

- users, groups, group memberships, etc…

The API can be used as an **integration point towards external systems**

- Example:
  - The SCIM API is used in the integration with the HTCondor batch system to do UNIX account pre-provisioning based on IAM account information

On-demand X.5O9 certificate generation

IAM integrates with the RCAuth.eu online certificate authority so that **users without an X.509 certificate can easily request one and link it to their membership**, via the IAM dashboard

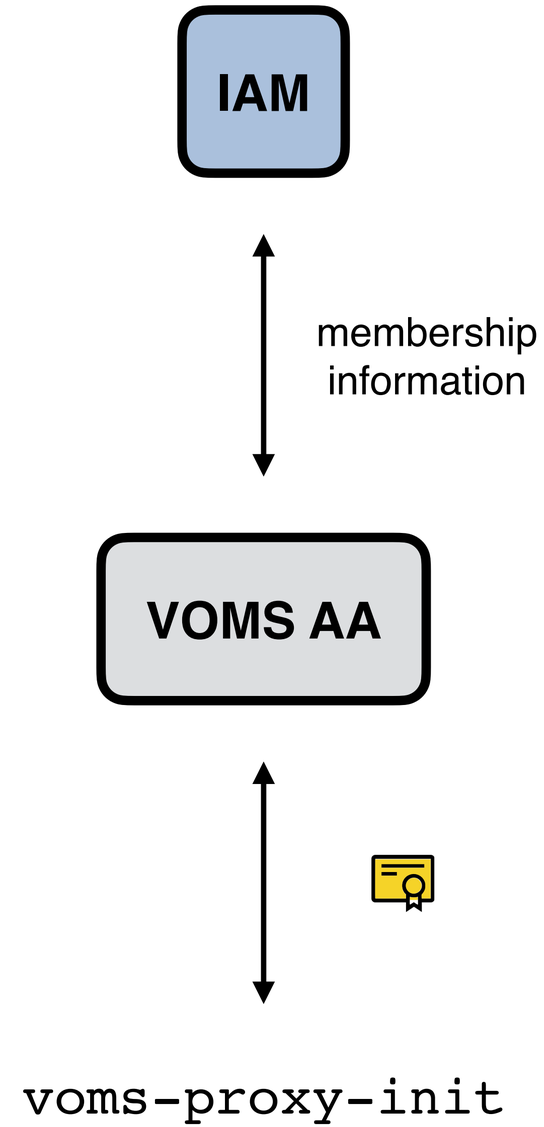
A long-lived X.509 proxy certificate is generated from the certificate obtained from RCAuth and stored in the IAM database

An **RESTful API** provides access to the certificate **to trusted clients**

VOMS provisioning

IAM includes a VOMS attribute authority micro-service that can encode IAM membership information in a **standard VOMS Attribute Certificate**

Proven compatibility with existing clients and Grid services



Easy integration with relying services

Standard OAuth/OpenID Connect enables **easy integration** with off-the-shelf services and libraries.

IAM has been successfully integrated with

- Openstack, Atlassian JIRA & Confluence, Moodle, Rocketchat, Grafana, Kubernetes, JupyterHub, dCache, StoRM, XRootD (HTTP), FTS, RUCIO, HTCondor






IAM documentation reference

https://indigo-iam.github.io/docs/

Provides information for:

- IAM service manager
- IAM VO administrators
- IAM users

IAM demo

Thanks for your attention.
Questions?

Backup slides

Token-based flows for WLCG data management

Scope-based AuthZ scenario

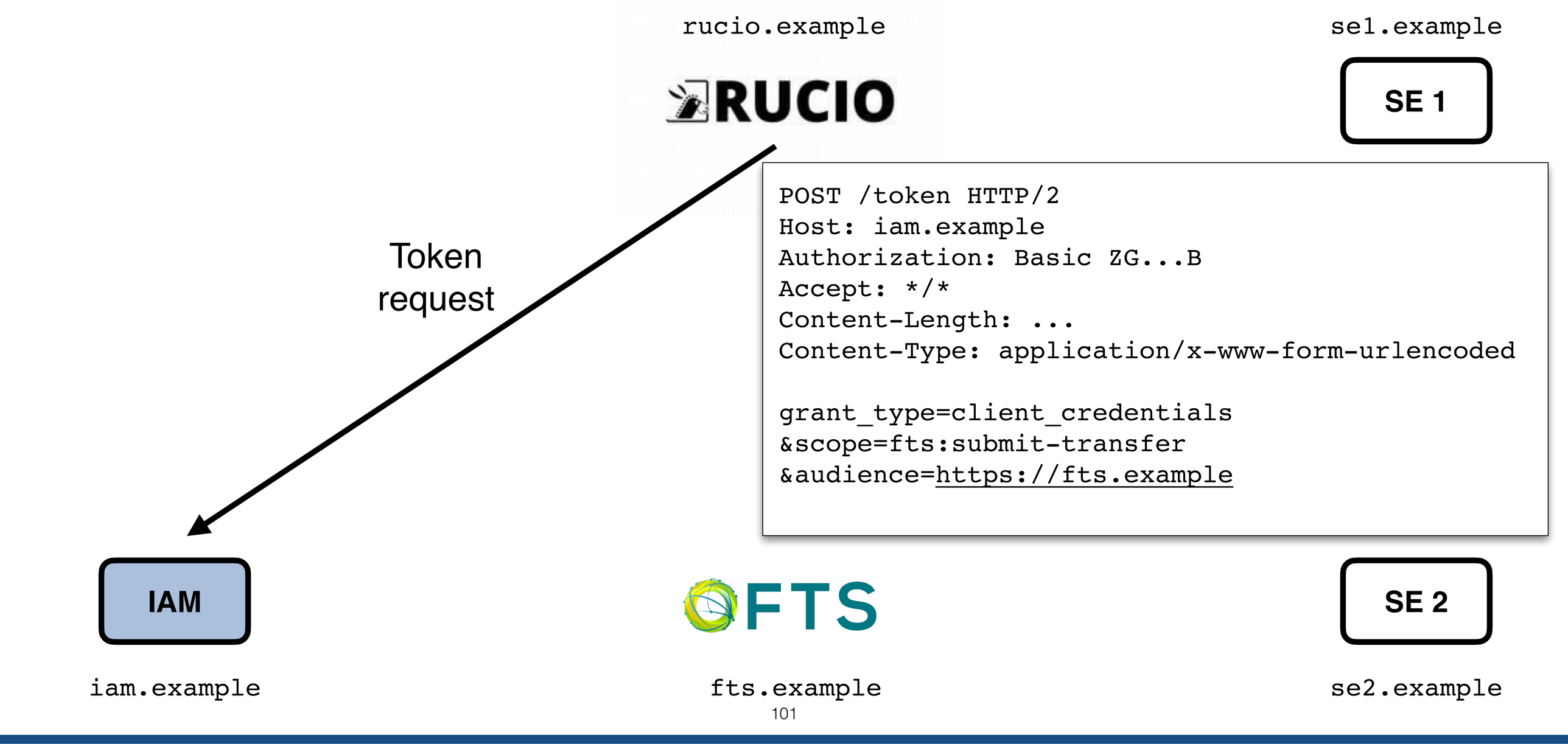
Token-based AuthN/Z for DOMA xfers: RUCIO delegated identity

In this scenario,
RUCIO delegates its identity to FTS to manage a third-party data transfer between SE 1 and SE 2

Token-based AuthN/Z for DOMA xfers: RUCIO delegated identity

RUCIO gets a token from IAM using the OAuth **client_credentials** grant type. The token needs to provide the minimum privileges need to interact with FTS

Token-based AuthN/Z for DOMA xfers: RUCIO delegated identity



Token-based AuthN/Z for DOMA xfers: RUCIO delegated identity

rucio.example

RUCIO

se1.example

SE 1

Token request

requested scopes & audience

```
POST /token HTTP/2
Host: iam.example
Authorization: Basic ZG...B
Accept: */*
Content-Length: ...
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials
&scope=fts:submit-transfer
&audience=https://fts.example
```

IAM

FTS

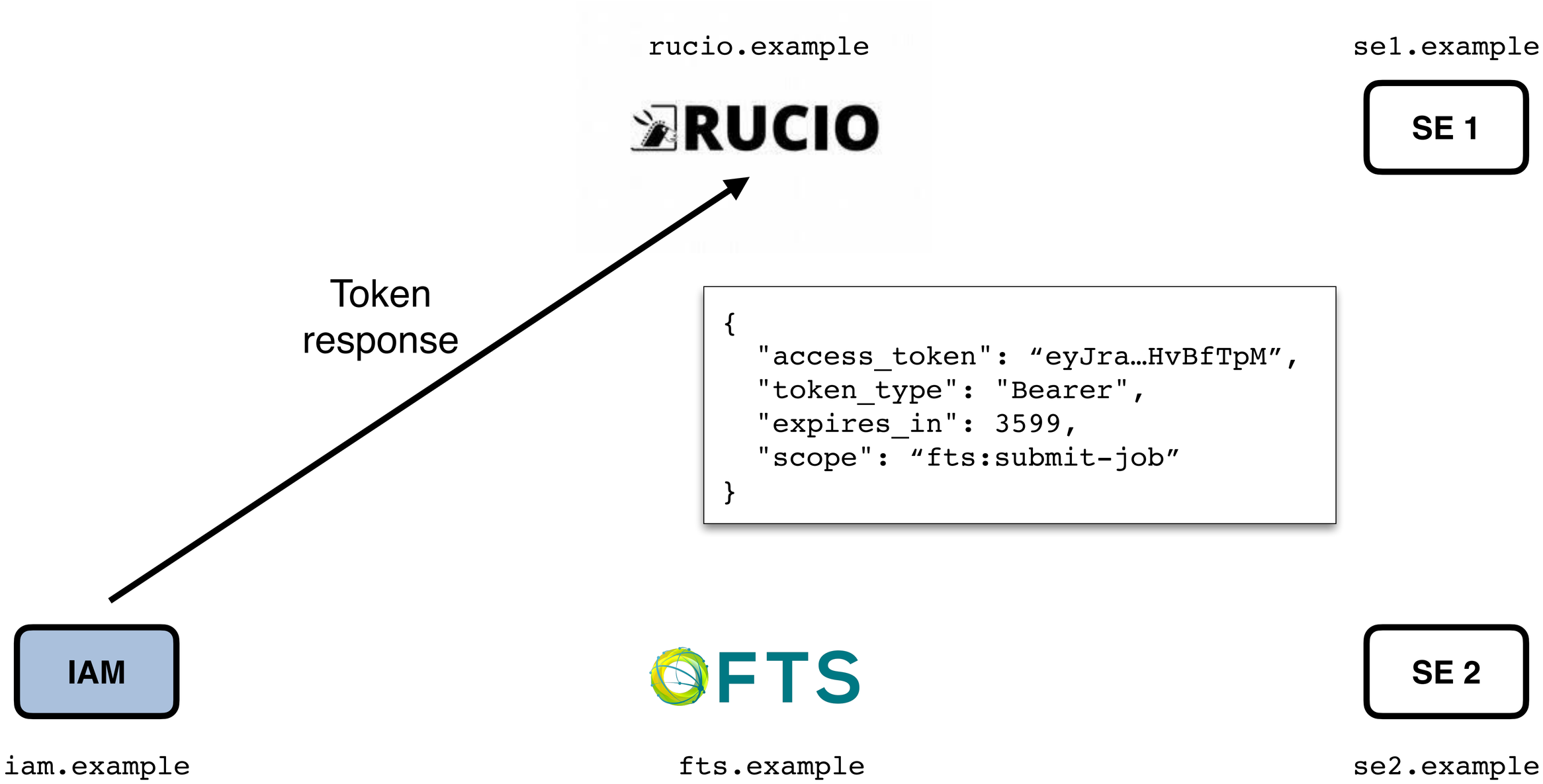
SE 2

iam.example

fts.example

se2.example

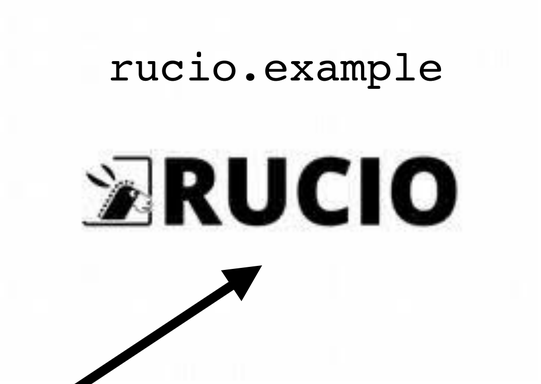
Token-based AuthN/Z for DOMA xfers: RUCIO delegated identity



Token-based AuthN/Z for DOMA xfers: RUCIO delegated identity

rucio.example

se1.example



SE 1

Token response

```
{
  "access_token": “eyJra…HvBfTpM”,
  "token_type": "Bearer",
  "expires_in": 3599,
  "scope": “fts:submit-job”
}
```

IAM

SE 2

iam.example

fts.example

se2.example

Token-based AuthN/Z for DOMA xfers: RUCIO delegated identity

Rucio extracts the access token from the response, and stores it in local memory.

rucio.example

se1.example

SE 1

```
{
  "access_token": “eyJra…HvBfTpM”,
  "token_type": "Bearer",
  "expires_in": 3599,
  "scope": “fts:submit-job”
}
```

parse & validate JWT

access token body:

```
{
  "sub": "rucio.example",
  "aud": “https://fts.example”,
  "nbf": 1572840340,
  "scope": "fts:submit-transfer",
  "iss": "https://iam.example/",
  "exp": 1572843940,
  "iat": 1572840340,
  "jti": "be48f2ab-8dd9-4df2-ae0b-bcb1fdfafaa6"
}
```

IAM

iam.example

fts.example

SE 2

se2.example

Token-based AuthN/Z for DOMA xfers: RUCIO delegated identity

The token audience is limited to FTS, and the requested scope has been granted.

rucio.example

RUCIO

se1.example

SE 1

access token body:

```
{
  "sub": "rucio.example",
  "aud": “https://fts.example”,
  "nbf": 1572840340,
  "scope": "fts:submit-transfer",
  "iss": "https://iam.example/",
  "exp": 1572843940,
  "iat": 1572840340,
  "jti": "be48f2ab-8dd9-4df2-ae0b-bcb1fdfafaa6"
}
```

IAM

iam.example

FTS

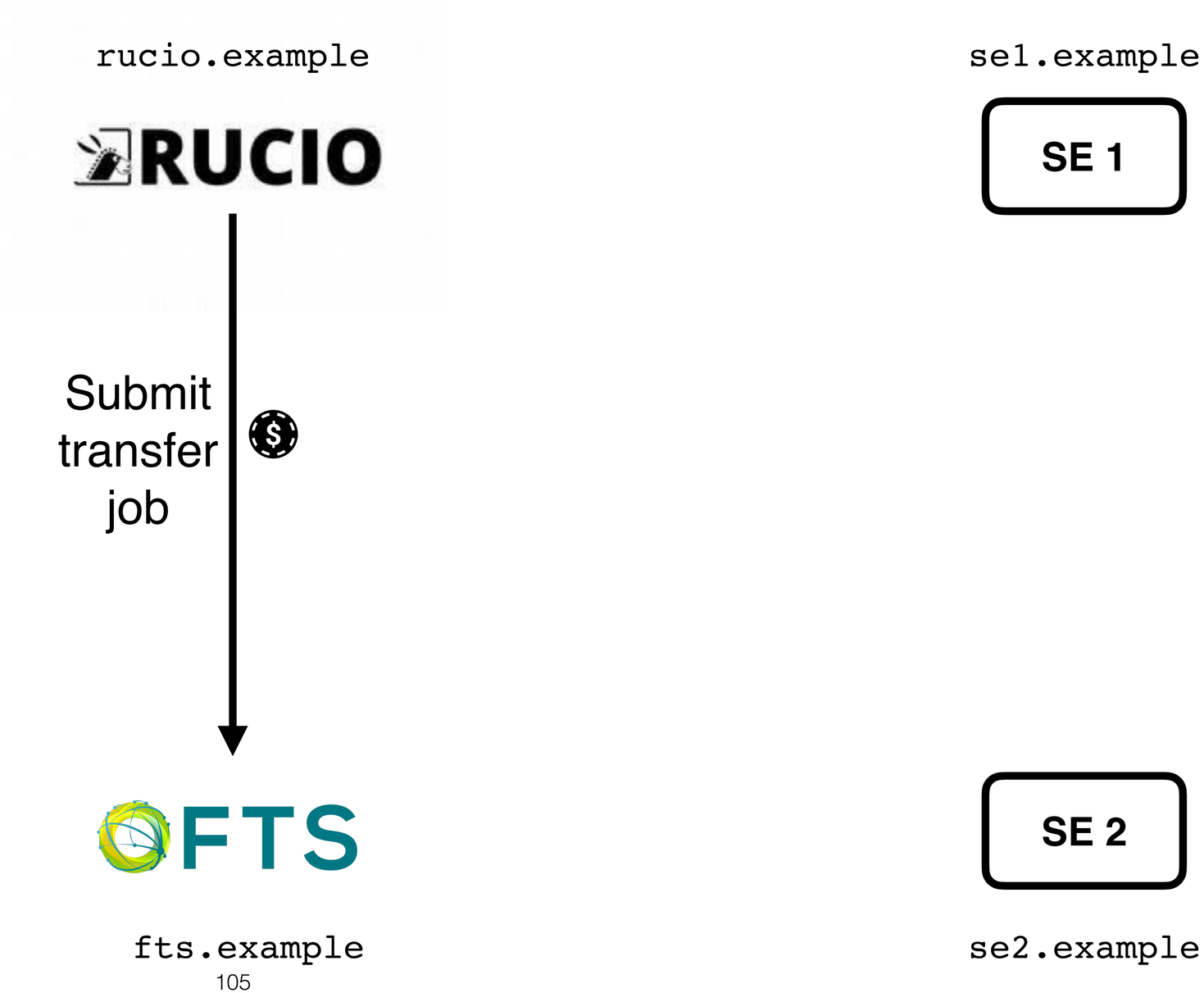
fts.example

SE 2

se2.example

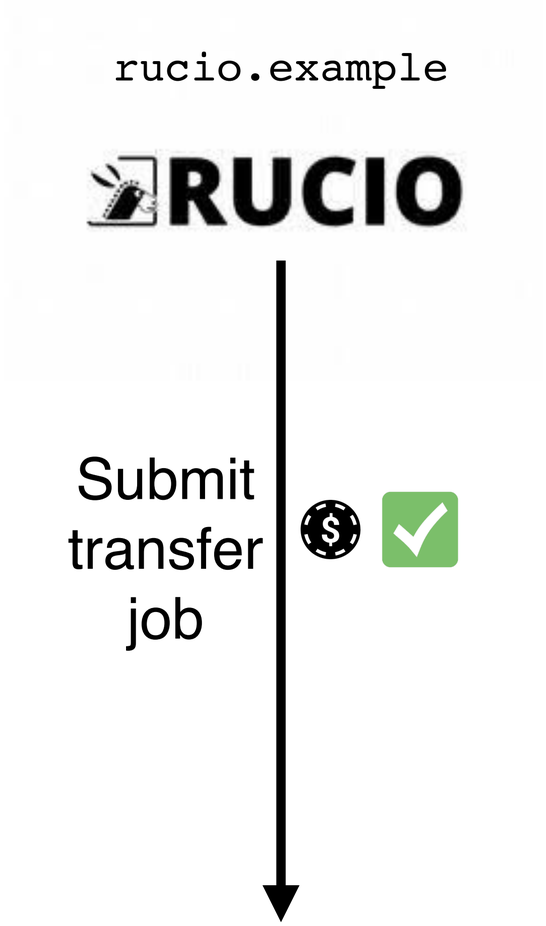
Token-based AuthN/Z for DOMA xfers: RUCIO delegated identity

RUCIO submits a transfer job to FTS, including the token obtained from IAM in the request



Token-based AuthN/Z for DOMA xfers: RUCIO delegated identity

FTS validates the token extracted from the request and accepts the transfer, assuming the token is valid and provides the necessary rights



se1.example

SE 1

IAM

iam.example

se2.example

SE 2

Token-based AuthN/Z for DOMA xfers: RUCIO delegated identity

rucio.example

se1.example

SE 1

FTS now needs a token that will be used for AuthN/Z at the storage elements. In this scenario, FTS impersonates RUCIO.

IAM

iam.example

fts.example

SE 2

se2.example

Token-based AuthN/Z for DOMA xfers: RUCIO delegated identity

rucio.example

se1.example

SE 1

The token it already has cannot be used for the transfer:
it's scoped to https://fts.example
and does not provide the necessary rights to read and store files at storage elements

IAM

iam.example

fts.example

SE 2

se2.example

Token-based AuthN/Z for DOMA xfers: RUCIO delegated identity

rucio.example

se1.example

SE 1

FTS then **exchanges the obtained token** with a couple of tokens, an access token and refresh token, that will be used to manage the transfer

IAM

iam.example

fts.example

SE 2

se2.example

Token-based AuthN/Z for DOMA xfers: RUCIO delegated identity

FTS requests the following scopes:
storage.read:/
storage.create:/
offline_access

rucio.example

RUCIO

se1.example

SE 1

```
POST /token HTTP/2
Host: iam.example
Authorization: Basic u89…
Accept: */*
Content-Length: ...
Content-Type: application/x-www-form-urlencoded

grant_type=urn:ietf:params:oauth:grant-type:token-exchange
&subject_token=eyJra…HvBfTpM
&audience=se1.example%20se2.example
&scope=storage.read%3A%2F%20storage.create%3A%2F%20offline_access
```

Token exchange request

SE 2

iam.example

fts.example

se2.example

Token-based AuthN/Z for DOMA xfers: RUCIO delegated identity

The audience of the token is limited to only apply to the storage elements involved in the transfer

rucio.example

RUCIO

se1.example

SE 1

```
POST /token HTTP/2
Host: iam.example
Authorization: Basic u89…
Accept: */*
Content-Length: ...
Content-Type: application/x-www-form-urlencoded

grant_type=urn:ietf:params:oauth:grant-type:token-exchange
&subject_token=eyJra…HvBfTpM
&audience=https%3A%2F%2Fse1.example%20https%3A%2F%2Fse2%2Fexample
&scope=storage.read%3A%2F%20storage.create%3A%2F%20offline_access
```

Token exchange request

IAM

FTS

SE 2

iam.example

fts.example

se2.example

Token-based AuthN/Z for DOMA xfers: RUCIO delegated identity

IAM validates the token exchange request, and assuming there's a policy that authorizes the exchange, issues the requested tokens

Token-based AuthN/Z for DOMA xfers: RUCIO delegated identity

rucio.example

RUCIO

se1.example

SE 1

```
{
  "access_token": “e7nd…HvBfTpM”,
  “refresh_token": “9njuk…”,
  "token_type": "Bearer",
  "expires_in": 3599,
  "scope": “storage.read:/ storage.create:/ offline_access”
}
```

Token exchange response

IAM

iam.example

FTS

fts.example

SE 2

se2.example

Token-based AuthN/Z for DOMA xfers: RUCIO delegated identity

FTS extracts the tokens from the response and saves them locally

rucio.example

se1.example

SE 1

```
{
  "access_token": “e7nd…HvBfTpM”,
  “refresh_token": “9njuk…”,
  "token_type": "Bearer",
  "expires_in": 3599,
  "scope": “storage.read:/
      storage.create:/
      offline_access”
}
```

IAM

iam.example

fts.example

SE 2

se2.example

Token-based AuthN/Z for DOMA xfers: RUCIO delegated identity

The new access token can be refreshed from IAM with the **refresh_token** flow.

Refresh tokens are typically much longer lived than access tokens and

rucio.example

RUCIO

se1.example

SE 1

access token body:

```
{
  "sub": "rucio.example",
  "aud": [“https://se1.example”,
          ”https://se2.example”],
  "nbf": 1572840345,
  "scope": “storage.read:/ storage.create:/
            offline_access”,
  "iss": "https://iam.example/",
  "exp": 1572843945,
  "iat": 1572840345,
  "jti": “be48…”
}
```

IAM

iam.example

FTS

fts.example

SE 2

se2.example

Token-based AuthN/Z for DOMA xfers: RUCIO delegated identity

FTS will enqueue the transfer job, and when the transfer is about to start can use the refresh token to get a fresh access token that will be used for the transfer.

Token-based AuthN/Z for DOMA xfers: RUCIO delegated identity

FTS then submits the third-party transfer against SE 2, including the token in the request

Token-based AuthN/Z for DOMA xfers: RUCIO delegated identity

The same token will be used for authn/z at se1 and se2.

It's also possible to have two separate tokens for each SE

Token-based AuthN/Z for DOMA xfers: RUCIO delegated identity

SE2 will then use the obtained token for authn/z against SE1

Group-based AuthZ scenario

Group-based authZ scenario

See flow description from the last hackathon

- https://github.com/WLCG-AuthZ-WG/hackathon/blob/master/authorization-flows.md