Identity & Authorization Management (IAM)
Mobile integration
Technical specifications
Version 1.11

This document is provided to you, free of charge, by the

eHealth platform

Willebroekkaai 38

38, Quai de Willebroek

1000 BRUSSELS

Table of contents

To the attention of: “IT expert” willing to integrate this web service.

1 Document management

1.1 Document history

Version	Date	Author	Description of changes / remarks
1.00	08/06/2017	eHealth platform	First version
1.1	31/01/2018	eHealth platform	Add key rollover description, update introspection documentation, update global overview, adapt token specification
1.2	25/02/2019	eHealth platform	PKCE recommendation for public client using AuthorizationCode flow, consent/revocation information Security update for clients using Client Credentials Flow (JTI claim required)
1.3	20/10/2020	eHealth platform	Healthcare and M2M realm setup, introduction of new claim mappers for WebAccess. WS-I Profile Tracing
1.4	02/02/2021	eHealth platform	Implicit flow is no longer proposed. New section about accessToken consumptions Error handling
1.5	26/06/2021	eHealth platform	Update refresh token section
1.6	17/10/2022	eHealth platform	Update Logout & redirection
1.7	03/03/2023	eHealth platform	Add new section for pseudonymization
1.8	09/08/2023	eHealth platform	Adapt with new mapper V1 & Confidential client with signed JWT and JWKS URL
1.9	29/05/2024	eHealth platform	Add new section : token exchange
1.10	17/01/2025	eHealth platform	Add explanation profile choices Update Authorization code flow Update Refresh token Update Logout section Update Test and release procedure
1.11	28/04/2025	eHealth platform	New section recommendation for refresh token Adaptation on Access token lifespan

2 Introduction

2.1 Goal of the service

eHealth IAM Connect is an identity and access management solution for web applications (WA) and RESTful web services (WS).

It allows clients to request and receive information about authenticated sessions and end-users. Clients of all types are supported: web application clients, JavaScript clients, native app/mobile clients.

IAM Connect also allows clients to verify the identity of the end-user based on the authentication performed by an "Authorization Server".

2.2 Goal of the document

This document is not a development or programming guide for internal applications. Instead, it provides functional and technical information and allows an organization to integrate and use eHealth IAM Connect.

However, in order to interact in a smooth, homogeneous and risk controlled way with a maximum of partners, the partners of the eHealth platform must commit to comply with the requirements of specifications, data format and release processes described in this document.

Technical and business requirements must be met in order to allow the integration and validation of the service of the eHealth platform in the client application.

2.3 eHealth platform document references

All the document references can be found on the portal of the eHealth platform[1]. These versions or any following versions can be used for the services of the eHealth platform.

ID	Title	Version	Date	Author
1	IAM Connect - Claim mappers	1.0	28/05/2021	eHealth platform
2	SOA – Error guide	1.0	10/06/2021	eHealth platform
3	Request test case template	3.0	22/02/2018	eHealth platform
4	WSDL	N.A.	N.A.	eHealth platform
5	IAM eXchange – Ann A - Security commitment Trusted Platform	1.5	28/05/2024	eHealth platform
6	IAM Connect Token eXchange – Security commitment	1.0	29/05/2024	eHealth platform

2.4 External document references

All documents can be found on the internet. They are available to the public, but not supported by the eHealth platform.

[1] ***Https://www.ehealth.fgov.be/ehealthplatform***

ID	Title	Source	Date	Author
1	The OAuth 2.0 Authorization Framework	**https://tools.ietf.org/html/rfc6749**	October 2012	D. Hardt, Ed. (Microsoft)
2	JSON Web Token (JWT)	**https://tools.ietf.org/html/rfc7519**	May 2015	M. Jones (Microsoft) J. Bradley (Ping Identity) N. Sakimura (NRI)
3	OAuth 2.0 Token Introspection	**https://tools.ietf.org/html/rfc7662**	October 2015	J. Richer, Ed.
4	Date and Time on the Internet: Timestamps	**https://tools.ietf.org/html/rfc3339**	July 2002	G. Klyne (Clearswift Corporation) C. Newman (Sun Microsystems)
5	An IANA Registry for Level of Assurance (LoA) Profiles	**https://tools.ietf.org/html/rfc6711**	August 2012	L. Johansson (NORDUNet)
6	OpenID Connect Core 1.0 incorporating errata set 1	**http://openid.net/specs/openid-connect-core-1_0.html**	8th November 2014	N. Sakimura (NRI) J. Bradley (Ping Identity) M. Jones (Microsoft) B. de Medeiros (Google) C. Mortimore (Salesforce)
7	JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants	**https://tools.ietf.org/html/rfc7523**	May 2015	M. Jones (Microsoft) B. Campbell (Ping Identity) C. Mortimore (Salesforece)
8	Proof Key for Code Exchange by OAuth Public Clients	**https://tools.ietf.org/html/rfc7636**	September 2015	N. Sakimura (NRI) J. Bradley (Ping Identity) N. Agarwal (Google)
9	JSON Web Key (JWK)	**https://tools.ietf.org/html/rfc7517**	May 2015	N. Jones (Microsoft)
10	OAuth 2.0 for Native Apps	**https://tools.ietf.org/html/rfc8252**	October 2017	W. Denniss (Google) J. Bradley (Ping Identity)
11	OpenID Connect RP-Initiated Logout 1.0	**https://openid.net/specs/openid-connect-rpinitiated-1_0.html#RedirectionAfterLogout**	September 12, 2022	M. Jones (Microsoft)

12	OAuth 2.0 Authorization Server Issuer Identification	**https://www.rfc-editor.org/rfc/rfc9207.html**	March 2022	K. Meyer zu Selhausen *Hackmanit* D. Fett *yes.com*

3 Support

3.1 Helpdesk eHealth platform

3.1.1 Certificates

In order to access the secured eHealth platform environment you have to obtain an eHealth platform certificate, used to identify the initiator of the request. In case you do not have one, please consult the chapter about the eHealth Certificates on the portal of the eHealth platform

- ***https://www.ehealth.fgov.be/ehealthplatform/nl/ehealth-certificaten***
- ***https://www.ehealth.fgov.be/ehealthplatform/fr/certificats-ehealth***

For technical issues regarding eHealth platform certificates

- Acceptance: ***acceptance-certificates@ehealth.fgov.be***
- Production: ***support@ehealth.fgov.be***

3.1.2 For issues in production

eHealth platform contact centre:

- Phone: 02 788 51 55 (on working days from 7 am till 8 pm)
- Mail: ***support@ehealth.fgov.be***
- *Contact Form :*
  - ***https://www.ehealth.fgov.be/ehealthplatform/nl/contact*** (Dutch)
  - ***https://www.ehealth.fgov.be/ehealthplatform/fr/contact*** (French)

3.1.3 For issues in acceptance

Integration-support@ehealth.fgov.be

3.1.4 For business issues

- regarding an existing project: the project manager in charge of the application or service
- regarding a new project or other business issues: ***info@ehealth.fgov.be***

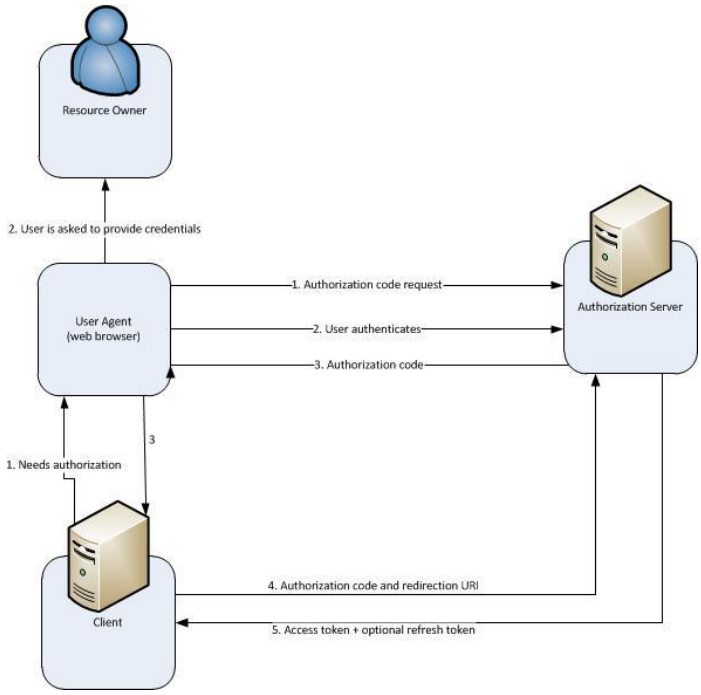
3.2 Status

The website ***https://status.ehealth.fgov.be*** is the monitoring and information tool for the ICT functioning of the eHealth services that are partners of the Belgian eHealth system.

4 Global overview

IAM Connect provides authorization flows for various needs. You can find a graphic description of the supported flows below. For the technical information see section 5.3.

4.1 Authorization code



The “Authorization Code” flow redirects the user agent to IAMConnect (1). Once the user has successfully authenticated with IAMConnect (2) an authorization code is created and the user agent is redirected back to the client application (3).

The application then uses the authorization code (4) to obtain an “Access Token”, “Refresh Token” and “ID Token” from IAMConnect (5).

The flow is suitable for web applications and native applications that have a backend component to exchange the authorization code for the token(s).

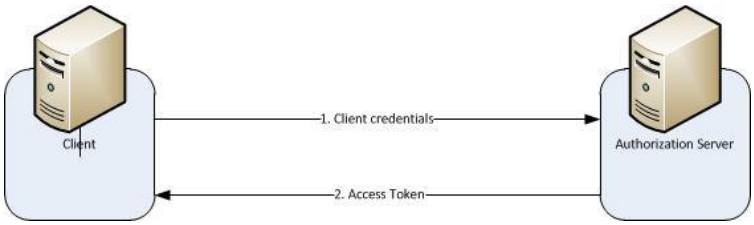
This flow may be used for client-side applications with mobile apps or web applications (running on user’s device). In many cases the client is the web browser.

The IAM Connect service uses a single user profile. When the profile is updated, it reflects the latest changes, meaning multiple profiles with different configurations for the same user are not possible. Each time the profile is modified, the user’s rights and access are recalculated when requesting a token, ensuring that access is based on the most current profile and minimizing the risk of unauthorized access.

When the user needs to authenticate on a client application with a different profile type, the authentication will be based on the user's most recent profile. If the user profile needs to be adjusted, this could be done using the 'prompt' parameter as described in the section 5.3.1.1.1.

4.2 Client credentials

Flow designed for a client requesting access to protected resources under the control of the client (the client is also the resource owner).



The client credentials flow is only available for organization/institution clients and does not support end user authentication.

4.3 Client definition

For each client, you can define

- the supported flows
  - authorization Code flow
  - client credentials flow
- his access type
  - confidential: clients who require a secret to initiate login protocol
  - public: clients who do not require a secret to initiate login protocol
  - bearer-only: web services that never initiate a login

In the table below, you will find the valid combinations between flows and access types.

	Authorization Code flow	**Client Credentials flow**
Public	**Authorized**	Not applicable
Confidential	**Authorized**	**Authorized**
Bearer-only	Not applicable	Not applicable

5 Step-by-step

5.1 Technical requirements

5.1.1 Application-level protocol

The endpoints and methods described hereafter all use HTTP as application-level protocol.

5.1.2 Transport Layer Security

Client applications MUST use TLS (i.e. HTTPS) communication with the authorization and resource endpoints to ensure protection of credentials and access tokens.

5.2 Process overview

This process describes how to use eHealth IAM Connect to access REST services for the domain of the eHealth platform which divides clients and services into security realms. Each client or service needs to be registered in a realm. Clients and services that need to connect with one another need to be defined in the same realm.

Two realms may be used :

- M2M realm dedicated to applications for machine-to-machine communication.
- Healthcare realm dedicated to applications for human actors in healthcare who use clients to access their data.

Other realms may be defined if there exist valid reasons to not use healthcare realm or M2M realm.

5.2.1 Registering clients with the authorization server

The registration procedure can be divided in two steps:

- Identify the realm
- Register the clients within the realm

When the realm has been identified, the partner must use the corresponding client registration form.

5.2.2 Environments

The following table describes the base URLs of the endpoints in each environment. The URLs described in later sections are relative to these base URLs.

Environment	Base URL
Integration	***https://api-int.ehealth.fgov.be/auth***
Acceptance	***https://api-acpt.ehealth.fgov.be/auth***
Production	***https://api.ehealth.fgov.be/auth***

5.2.3 Well-known

The following table describes the well-known endpoints for each realm in each environment. These URLs provide a lot of useful information about the realm openid configuration.

Realm	Realm id
M2M INT	***https://api-int.ehealth.fgov.be/auth/realms/M2M/.well-known/openid-configuration***

M2M ACC	***https://api-acpt.ehealth.fgov.be/auth/realms/M2M/.well-known/openid-configuration***
M2M PROD	***https://api.ehealth.fgov.be/auth/realms/M2M/.well-known/openid-configuration***
Healthcare INT	***https://api-int.ehealth.fgov.be/auth/realms/healthcare/.well-known/openid-configuration***
Healthcare ACC	***https://api-acpt.ehealth.fgov.be/auth/realms/healthcare/.well-known/openid-configuration***
Healthcare PROD	***https://api.ehealth.fgov.be/auth/realms/healthcare/.well-known/openid-configuration***

5.3 OpenID Connect flows

5.3.1 Authorization code flow

The authorization code flow offers more security than the implicit flow as tokens are not directly sent to the client application.

The OpenID Connect protocol, in abstract, follows the steps below.

1. The client sends (GET) a request to the authorization server.
2. The authorization server authenticates the end-user (if he does not have an active session yet).
3. The authorization server obtains end-user consent/authorization (if he has not given his consent yet).
4. The authorization server sends the end-user back to the client with an authorization code.
5. The client requests (POST) a response using the authorization code at the token endpoint.
6. The client receives a response (POST) which contains an IDToken and AccessToken in the response body.
7. The client validates the IDToken.

The URL depends on the realm the token is requested for. It can be retrieved using the well-known endpoints:

- Authentication endpoint : *authorization_endpoint*
  https://[BaseUrl]/auth/realms/[realm id][2]/protocol/openid-connect/auth
- Token endpoint : *token_endpoint*
  https://[BaseUrl]/auth/realms/[realm id]/protocol/openid-connect/token

5.3.1.1 *Authentication request*

5.3.1.1.1 Input arguments

Endpoint: Authentication endpoint

HTTP method: GET.

The client constructs the request URI by adding the following parameters to the query component of the authorization endpoint URI using the "application/x-www-form-urlencoded" format.

Field name	Description
client_id	MANDATORY. Client identifier valid at the authorization server.
response_type	MANDATORY. Value MUST contain “code”.

[2] realm id = M2M or healthcare

scope	MANDATORY. Space-delimited list of scopes of the access request. MUST contain “openid” scope.
redirect_uri	MANDATORY. Redirection URI to which the response will be sent. This URI MUST exactly match one of the redirection URI values for the client pre-registered at the OpenID provider (Simple String Comparison). When using this flow, the Redirection URI SHOULD use the `https` scheme; however. The Redirection URI MAY use an alternate scheme, such as one, intended to identify a call back into a native application.
nonce	MANDATORY. String value used to associate a client session with an ID token, and to mitigate replay attacks. The value is passed on through unmodified from the authentication request to the ID Token. Sufficient entropy MUST be present in the nonce values used to prevent attackers from guessing values (for more information, see OpenID Connect Core 1.0 specification[3]).
state	RECOMMENDED. Opaque value used to maintain state between the request and the call back. Typically, Cross-Site Request Forgery (CSRF) mitigation is done by cryptographically binding the value of this parameter with a browser cookie.
prompt	OPTIONAL. String value used to specify whether the AuthorizationServer prompts the end user for an action (re-authentication, consent, …) Possible values : • login : the user will be prompted to select a profile even if he/she already chose a profile in a previous login attempt. • none : the login page will never be shown, instead the user will be redirected to the application, with an error if the user is not yet authenticated • consent : applicable only for the clients with Consent Required. If it is used, the Consent page will always be displayed, even if the user previously granted consent to this client.

5.3.1.1.2 Output arguments (success)

HTTP status code: 302 Found.

If the resource owner grants the access request, the authorization server issues an authorization code and delivers it to the client by adding the following parameters to the query component of the redirection URI using the "application/x-www-form-urlencoded" format.

Field name	Description
Code	REQUIRED. The authorization code generated by the authorization server. The client MUST NOT use the authorization code more than once. The authorization code is bound to the client identifier and redirection URI.
State	REQUIRED if the state parameter is present in the Authorization Request. Clients MUST verify that the state value is equal to the value of state parameter in the Authorization Request.
Iss	REQUIRED. This parameter is used to explicitly include the issuer identifier of the authorization server in the authorization response of an OAuth authorization flow.

Authentication response validation

When using the Authorization Code Flow, the Client MUST validate the response according to RFC 6749, especially Sections 4.1.2 and 10.12 and RFC9207 for ISS parameter.

[3] ***http://openid.net/specs/openid-connect-core-1_0.html#NonceNotes***

5.3.1.1.3 Output arguments (error)

HTTP status code: 302 Found.

If the resource owner does not grant the access request, the authorization server issues an error and delivers it to the client by adding the following parameters to the fragment component of the redirection URI using the "application/x-www-form-urlencoded" format.

Field name	Description
Error	An error code.
error_description	Human-readable description of the error.
State	The exact value initially received from the client.

The list of errors can be found in section 8.

5.3.1.2 *Token endpoint*

The token endpoint is used to obtain tokens. Tokens can be obtained by exchanging an authorization code or by supplying credentials directly depending on what flow is used. The token endpoint may also be used to obtain new access tokens when they expire.

To obtain an access token, an ID token, and optionally a refresh token, the client sends a token request to the token endpoint to obtain a token response, as described in Section 3.2 of OAuth 2.0 (RFC6749).

The refresh token can be used only once.

A new refresh token will be delivered upon successful renewal of the accessToken.

5.3.1.2.1 Input arguments for access token request

Endpoint: Token endpoint

HTTP method: POST.

A client makes a token request by presenting its authorization grant (in the form of an authorization code) to the token endpoint using the *grant_type* value *authorization_code*, as described in Section 4.1.3 of OAuth 2.0 (RFC6749).

Field name	Description
grant_type	MANDATORY. Value MUST be set to “authorization_code”
Code	MANDATORY. The authorization code received from the authorization server.
redirect_uri	MANDATORY if the "redirect_uri" parameter was included in the authorization request. Their values MUST be identical.
client_id	MANDATORY if the client is not authenticating with the authorization_server

If the client is a confidential client then he MUST authenticate to the token endpoint using the authentication method registered for its *client_id.* Extra fields must be added in the request :

Field name	Description
client_assertion_type	MANDATORY. Value must be set to “urn:ietf:params:oauth:client-assertion-type:jwt-bearer”
client_assertion	MANDATORY. JWT token signed with the certificate of the partner’s client.

The JWT token must contain some elements as defined in ***https://tools.ietf.org/html/rfc7523#section-3.*** See section 5.3.3 Confidential client authentication for more information.

5.3.1.2.2 Input arguments for refresh token request

Endpoint: Token endpoint

HTTP method: POST.

If the client is a confidential client, then he MUST authenticate to the token endpoint using the authentication method registered for its *client_id*. See section 5.3.3 Confidential client authentication for more information.

Field name	Description
client_id	MANDATORY if the client is not authenticating with the authorization_server
grant_type	MANDATORY. Value MUST be set to “refresh_token”.
refresh_token	MANDATORY. The refresh token issued to the client
Scope	OPTIONAL. The scope of the access request. The requested scope MUST NOT include any scope not originally granted by the resource owner. Thus it is not possible to increase the range of the scope following a refresh it can only be narrowed. If the scope is omitted, it is treated as equal to the scope originally granted by the resource owner.

5.3.1.2.3 Output arguments for access/refresh token (success)

HTTP status code: 200 OK.

The parameters are included in the entity-body of the HTTP response using the "application/json" media type. The parameters are serialized into a JSON structure by adding each parameter at the highest structure level. The order of parameters is irrelevant and can vary.

Field name	Description
access_token	MANDATORY. The access token issued by the authorization server
token_type	MANDATORY. The type of the token issued. Always “bearer”.
expires_in	MANDATORY. The lifetime in seconds of the access token.
refresh_token	OPTIONAL. The refresh token, which can be used to obtain new access tokens using the same authorization grant.
Scope	The scope of the access token. OPTIONAL, if identical to the scope requested by the client; REQUIRED otherwise.

ID Token validation

See Sections 3.1.3.7[4] of the OpenID Connect Core 1.0 specification for requirements regarding client-side validation of ID Tokens. Clients MUST validate ID Tokens.

Access token validation

If the ID token contains an `at_hash` Claim, the client MUST use it to validate the access token as described in section 3.1.3.8[5] of the OpenID Connect Core 1.0 specification.

If the accessToken is not suitable for the client, the client MUST propose the end user to change his/her profile. This operation can be performed with the parameter prompt=login (see section 5.3.1.1.1).

5.3.1.2.4 Output arguments for Access/Refresh Token (error)

HTTP status code: 400 (Bad Request) (unless specified otherwise).

[4] ***https://openid.net/specs/openid-connect-core-1_0.html#IDTokenValidation***

[5] ***http://openid.net/specs/openid-connect-core-1_0.html#CodeFlowTokenValidation***

Field name	Description
Error	MANDATORY. An error code.
error_description	OPTIONAL. Human-readable description of the error.
error_uri	OPTIONAL. A URI identifying a human-readable web page used to provide the client developer with additional information about the error.

For the list of errors: see section 8.

5.3.1.3 *Example*

```
GET /auth/realms/[realm id]/protocol/openid-connect/auth

Request Headers:
        Host: localhost:8080
        Referer: http://localhost:8000/
Request DATA:
        client_id=tutorial-frontend&
        redirect_uri=http%3A%2F%2Flocalhost%3A8000%2F&
        state=2c988c4b-0fb9-4678-89b0-4de60ba44dbb&
        nonce=51ccca50-2e08-4fb9-8374-beb09df2eba3&
        response_mode=fragment&
        response_type=code&
        scope=openid

Response Headers:
        HTTP/1.1 302 Found
-------------------------------------------------------

GET /?redirect_fragment=%2Fmain
Request Headers:
        Host: localhost:8000
        User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:51.0) Gecko/20100101 Firefox/51.0
        Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
        Accept-Language: nl,en-US;q=0.7,en;q=0.3
        Accept-Encoding: gzip, deflate
        Connection: keep-alive
        Upgrade-Insecure-Requests: 1
Request DATA:
        state=234a37a1-9cb3-42b6-868c-c6b13d351c96&
        code=OTlZYsQ9bTZrdR78O3ijXSsHJV9agZ3dG4hqz2Jgu4c.9ff864ad-145d-459f-8d42-bc7c1c16b48a

Response Headers:
        HTTP/1.1 200 OK
-------------------------------------------------------

POST /auth/realms/[realm id]/protocol/openid-connect/token

Request Headers:
        Host: localhost:8080
        Referer: http://localhost:8000/
        origin: http://localhost:8000
Request DATA:
```

```
        code=uFy-rrFCc8b4InD7TSu9vfZfsrSQ6EBKdds8bmEVBgY.3c13b729-d889-4392-8bea-af7839bdf5bb
        grant_type=authorization_code
        client_id=tutorial-frontend
        redirect_uri=http://localhost:8000/

Response Headers:
        HTTP/1.1 200 OK
        Content-Type: application/json
        Content-Length: 4105
Response DATA:
        {

        "access_token":"eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJKVUNoTEVnZkYzNjBXRWg0dzJ4OV
FneEhqamhYUzN6Y0gyLW5helQ1clNnIn0.eyJqdGkiOiI2NDAxZGM3Mi02MDkwLTQ1MDgtYjEwYy0zOWMyYTJkMjY4NTAiLCJleHAiO
jE0ODYxMjg3OTksIm5iZiI6MCwiaWF0IjoxNDg2MTI4NDk5LCJpc3MiOiJodHRwOi8vbG9jYWxob3N0OjgwODAvYXV0aC9yZWFsbXMv
RGVtby1SZWFsbSIsImF1ZCI6InR1dG9yaWFsLWZyb250ZW5kIiwic3ViIjoiZWU1MWNhYWYtOTY4MC00MmU3LWJiZTQtYmRjYjE0NTc
xMWI5IiwidHlwIjoiQmVhcmVyIiwiYXpwIjoidHV0b3JpYWwtZnJvbnRlbmQiLCJub25jZSI6IjIxYzgwNWFjLTJmYTctNGQxMC1hND
YwLTFmNWVlYzA3YTFlNiIsImF1dGhfdGltZSI6MTQ4NjEyODQ5OSwic2Vzc2lvbl9zdGF0ZSI6ImI5ZmU4OWQ1LWYwNWMtNGIwNC05N
zNhLTQ5OTQxNzUxYzA3YyIsImFjciI6IjEiLCJjbGllbnRfc2Vzc2lvbiI6IjYxOTIyMTVhLTNmMWEtNGU3My1hNmNjLWEzY2E5NjQz
MzY5ZCIsImFsbG93ZWQtb3JpZ2lucyI6WyJodHRwOi8vbG9jYWxob3N0OjgwMDAiXSwicmVhbG1fYWNjZXNzIjp7InJvbGVzIjpbIm1
hbmFnZXIiLCJ1bWFfYXV0aG9yaXphdGlvbiIsInVzZXIiXX0sInJlc291cmNlX2FjY2VzcyI6eyJhY2NvdW50Ijp7InJvbGVzIjpbIm
1hbmFnZS1hY2NvdW50Iiwidmlldy1wcm9maWxlIl19fSwicGVyc29uIjp7InNzaW4iOiI3NjEyMDkwMjUyNyJ9LCJuYW1lIjoiRnJlZ
GVyaWsgTGliZXJ0IiwicHJlZmVycmVkX3VzZXJuYW1lIjoiNmZlODgyYmEtM2UwMi00NzM5LTkxZGEtNThiNmI3ZDBhNDg1IiwiZ2l2
ZW5fbmFtZSI6IkZyZWRlcmlrIiwiZmFtaWx5X25hbWUiOiJMaWJlcnQifQ.TotVCIyxwF3ef1G85Vr_8vZvTp4tE1rrr_2GaamLmODs
IIYukuIEPKHmJlkgXvkQUo8AdkUnbr9iLsm1JrmFBPbahyhMJeyvK0pPFqNZxTBxBRQAWjn0oDu7PHFz23Rnur9__kCdlnEZhUFf71E
QD8_31mGlEL5FLTtOPf6oThykiBCnTEbp4oPmRPvz06TpI0_iw2mbRnV2nmvQmvq7HXNdO_mrM9THd5gGv1zXmGzRrB1Vpu895mZBbY
ANg5La6NU0-ymijWhksB_i0jrLqkb0lcJqbsf3n4GQx-q5S9TbdDT9F7PIU53SyBmYutB8F9ZcE3bLQthBlaivR0Ky2Q",
                "expires_in":300,
                "refresh_expires_in":1800,

        "refresh_token":"eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJKVUNoTEVnZkYzNjBXRWg0dzJ4O
VFneEhqamhYUzN6Y0gyLW5helQ1clNnIn0.eyJqdGkiOiIyYWIwNGU5My03MGI1LTRjZmItOTA3Yy03YzcyZDMyZWZiZTUiLCJleHAi
OjE0ODYxMzAyOTksIm5iZiI6MCwiaWF0IjoxNDg2MTI4NDk5LCJpc3MiOiJodHRwOi8vbG9jYWxob3N0OjgwODAvYXV0aC9yZWFsbXM
vRGVtby1SZWFsbSIsImF1ZCI6InR1dG9yaWFsLWZyb250ZW5kIiwic3ViIjoiZWU1MWNhYWYtOTY4MC00MmU3LWJiZTQtYmRjYjE0NT
cxMWI5IiwidHlwIjoiUmVmcmVzaCIsImF6cCI6InR1dG9yaWFsLWZyb250ZW5kIiwibm9uY2UiOiIyMWM4MDVhYy0yZmE3LTRkMTAtY
TQ2MC0xZjVlZWMwN2ExZTYiLCJhdXRoX3RpbWUiOjAsInNlc3Npb25fc3RhdGUiOiJiOWZlODlkNS1mMDVjLTRiMDQtOTczYS00OTk0
MTc1MWMwN2MiLCJjbGllbnRfc2Vzc2lvbiI6IjYxOTIyMTVhLTNmMWEtNGU3My1hNmNjLWEzY2E5NjQzMzY5ZCIsInJlYWxtX2FjY2V
zcyI6eyJyb2xlcyI6WyJtYW5hZ2VyIiwidW1hX2F1dGhvcml6YXRpb24iLCJ1c2VyIl19LCJyZXNvdXJjZV9hY2Nlc3MiOnsiYWNjb3
VudCI6eyJyb2xlcyI6WyJtYW5hZ2UtYWNjb3VudCIsInZpZXctcHJvZmlsZSJdfX19.fUy6-m0nEWsc-VyS-
MbNZMsFQfx0VgOCvOvA3PI3Et8qYAYgcy1zz_Uk6ygVByxVt9AMGyZ3JvlZxl01IelKrCjQGkSbJx6RbMjnhqVgaQLVFWXFo6z2LFZR
Aow71_-r9T3DXqWV1g6gOoZ0Pyhj6j_0N7Vc6MloERhq5VnXlZHGNyDrEZrTTExa4lHjQ556hMzYhjkb-
3m9C0NFKrrvCssxMizC7r2MqHTZ-
SEqji3KAbW19koSbHeGoeTfccfi8XDGTBH0vt3Bn0Ep8RUc4fbwjgZh9qmD1ycnsu8qb2TPy3X068Xk0ZDtowYSArp4CpGxVyROB1Wn
5PuK8wbfFw",
                "token_type":"bearer",

        "id_token":"eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJKVUNoTEVnZkYzNjBXRWg0dzJ4OVFneE
hqamhYUzN6Y0gyLW5helQ1clNnIn0.eyJqdGkiOiI5OTY1ZDg0NS0wOGZjLTQ2M2EtODc3My1kOTBkZDhkNmIyMDYiLCJleHAiOjE0O
DYxMjg3OTksIm5iZiI6MCwiaWF0IjoxNDg2MTI4NDk5LCJpc3MiOiJodHRwOi8vbG9jYWxob3N0OjgwODAvYXV0aC9yZWFsbXMvRGVt
by1SZWFsbSIsImF1ZCI6InR1dG9yaWFsLWZyb250ZW5kIiwic3ViIjoiZWU1MWNhYWYtOTY4MC00MmU3LWJiZTQtYmRjYjE0NTcxMWI
5IiwidHlwIjoiSUQiLCJhenAiOiJ0dXRvcmlhbC1mcm9udGVuZCIsIm5vbmNlIjoiMjFjODA1YWMtMmZhNy00ZDEwLWE0NjAtMWY1ZW
VjMDdhMWU2IiwiYXV0aF90aW1lIjoxNDg2MTI4NDk5LCJzZXNzaW9uX3N0YXRlIjoiYjlmZTg5ZDUtZjA1Yy00YjA0LTk3M2EtNDk5N
DE3NTFjMDdjIiwiYWNyIjoiMSIsInBlcnNvbiE6eyJzc2luIjoiNzYxMjA5MDI1MjcifSwibmFtZSI6IkZyZWRlcmlrIExpYmVydCIs
InByZWZlcnJlZF91c2VybmFtZSI6IjZmZTg4MmJhLTNlMDItNDczOS05MWRhLTU4YjZiN2QwYTQ4NSIsImdpdmVuX25hbWUiOiJGcmV
kZXJpayIsImZhbWlseV9uYW1lIjoiTGliZXJ0In0.Q2hj5L5vyau7ztgSMnbA9E65rSg1bs3E1sqEtRQumLKOqD3FoYvSXvARK5kvZN
4wCUWEarYO82boosUS7P89rPQNDIq-
l9Mx7wCOqh3c0q8c_1Lb5IJ5o2MwQUhG0soFOpx3qD92siG9gKD_vHjHifZoJGG7t9YLkvf2vI3P5IjdvxrA3tZUlNtHuWwnH0FzovS
S8Xi_wYyZV8nwZ09vyoFgSapNZQoqFPiJPiXO2Jhy-larI0hu3-
tGbT35pXTfz8UxILeJDNuElq6_Cv6Uyn8qPmHE1dnpkxNESbeSq5zU-ZicVWzQ5C_neMSzM6REntExv9ud_tWB-IdylMhtig",
                "not-before-policy":1486050488,
                "session_state":"b9fe89d5-f05c-4b04-973a-49941751c07c"
        }
----------------------------------------------------------
```

```
GET /auth/realms/[realm id]/account

Request Headers:
        Host: localhost:8080
        Accept: application/json
        Authorization: bearer
eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJKVUNoTEVnZkYzNjBXRWg0dzJ4OVFneEhqamhYUzN6Y0gyLW5helQ
1clNnIn0.eyJqdGkiOiIwNWQ2YmNmNC0zNmI1LTQ4YzgtOTQ5MC0yNThjZjFkMzc0ZjgiLCJleHAiOjE0ODYxMzAxNDYsIm5iZiI6MC
wiaWF0IjoxNDg2MTI5ODQ2LCJpc3MiOiJodHRwOi8vbG9jYWxob3N0OjgwODAvYXV0aC9yZWFsbXMvRGVtby1SZWFsbSIsImF1ZCI6I
nR1dG9yaWFsLWZyb250ZW5kIiwic3ViIjoiZWU1MWNhYWYtOTY4MC00MmU3LWJiZTQtYmRjYjE0NTcxMWI5IiwidHlwIjoiQmVhcmVy
IiwiYXpwIjoidHV0b3JpYWwtZnJvbnRlbmQiLCJub25jZSI6IjUxY2NjYTUwLTJlMDgtNGZiOS04Mzc0LWJlYjA5ZGYyZWJhMyIsImF
1dGhfdGltZSI6MTQ4NjEyODQ5OSwic2Vzc2lvbl9zdGF0ZSI6ImI5ZmU4OWQ1LWYwNWMtNGIwNC05NzNhLTQ5OTQxNzUxYzA3YyIsIm
FjciI6IjAiLCJjbGllbnRfc2Vzc2lvbiI6IjNjMTNiNzI5LWQ4ODktNDM5Mi04YmVhLWFmNzgzOWJkZjViYiIsImFsbG93ZWQtb3JpZ
2lucyI6WyJodHRwOi8vbG9jYWxob3N0OjgwMDAiXSwicmVhbG1fYWNjZXNzIjp7InJvbGVzIjpbIm1hbmFnZXIiLCJ1bWFfYXV0aG9y
aXphdGlvbiIsInVzZXIiXX0sInJlc291cmNlX2FjY2VzcyI6eyJhY2NvdW50Ijp7InJvbGVzIjpbIm1hbmFnZS1hY2NvdW50Iiwidml
ldy1wcm9maWxlIl19fSwicGVyc29uIjp7InNzaW4iOiI3NjEyMDkwMjUyNyJ9LCJuYW1lIjoiRnJlZGVyaWsgTGliZXJ0IiwicHJlZm
VycmVkX3VzZXJuYW1lIjoiNmZlODgyYmEtM2UwMi00NzM5LTkxZGEtNThiNmI3ZDBhNDg1IiwiZ2l2ZW5fbmFtZSI6IkZyZWRlcmlrI
iwiZmFtaWx5X25hbWUiOiJMaWJlcnQifQ.hK_PKfMpHnst_ADn251xkCN3P0JwwVim5kOJMV4waOdwKd2-cfN4NAG9i9h-
KHoCnJP3ofMa5NEN-s7_HoSioBSa2X3wt-kUiDGuiZ3pQrMKv20J_en66FztJvYbcAoTFJUFucqnPzdJ-q9mVCfI-X-
bLqZGtMSbVA1rxdAPS4Wx_dRu_xcmdL20Ma3RpW36v2iO-
XiekjHdbTFeFDG4aayyYxrkllotxRYdy8clay61MNSHVKm0avM2JWsJTWe-5Az-
tufEr2BOZIr9dgKLWozpVb7iLscZBEJWV7B1wWzhrbhygsHa6iSVHk1WbcUWaymceJqSKMaj1w5MgHu59Q
        Referer: http://localhost:8000/
        origin: http://localhost:8000

Response Headers:
        HTTP/1.1 200 OK
        Content-Type: application/json
        Content-Length: 792
Response DATA:
        {
           "id":"ee51caaf-9680-42e7-bbe4-bdcb145711b9",
           "createdTimestamp":1486127148882,
           "username":"6fe882ba-3e02-4739-91da-58b6b7d0a485",
           "enabled":true,
           "totp":false,
           "emailVerified":false,
           "firstName":"John",
           "lastName":"Doe",
           "attributes":{
                  "organizationId":["0809394427"],
                  "ssin":["12345678901"],
                  "role":["user"],
                  "organizationName":["PLATE-FORME EHEALTH"],
                  "email":["do_not_reply@ehealth.fgov.be"]
           },
           "disableableCredentialTypes":[],
           "requiredActions":[]
        }


JWT
REQUEST token
POST /auth/realms/[realm id]/protocol/openid-connect/token
HEADER :
{
  "kid": "S5067472458957989586",
  "typ": "JWT",
```

```
  "alg": "RS256"
}

PAYLOAD :

{
  "sub": "client_id",
  "aud": "https://[Base_url]/auth/realms/[realm_id]",
  "iss": "client_id",
  "exp": 1653056528,
  "jti": "UCmemwCtf5AYdb8yGFE0NyfINEIiiYNV"
}

RESPONSE :
HEADER :
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "JUChLEgfF360WEh4w2x9QgxHjjhXS3zcH2-nazT5rSg"
}
PAYLOAD access_token:
        {
          "jti": "6401dc72-6090-4508-b10c-39c2a2d26850",
          "exp": 1486128799,
          "nbf": 0,
          "iat": 1486128499,
          "iss": "http://localhost:8080/auth/realms/[realm id]",
          "aud": "tutorial-frontend",
          "sub": "ee51caaf-9680-42e7-bbe4-bdcb145711b9",
          "typ": "Bearer",
          "azp": "tutorial-frontend",
          "nonce": "21c805ac-2fa7-4d10-a460-1f5eec07a1e6",
          "auth_time": 1486128499,
          "session_state": "b9fe89d5-f05c-4b04-973a-49941751c07c",
          "acr": "1",
          "client_session": "6192215a-3f1a-4e73-a6cc-a3ca9643369d",
          "allowed-origins": [
                "http://localhost:8000"
          ],
          "realm_access": {
                "roles": [
                  "manager",
                  "uma_authorization",
                  "user"
                ]
          },
          "resource_access": {
                "account": {
                  "roles": [
                        "manage-account",
                        "view-profile"
                  ]
                }
          }
```

```
        }
```

PAYLOAD refresh_token:


```
        {
          "jti": "2ab04e93-70b5-4cfb-907c-7c72d32efbe5",
          "exp": 1486130299,
          "nbf": 0,
          "iat": 1486128499,
          "iss": "http://localhost:8080/auth/realms/[realm id]",
          "aud": "tutorial-frontend",
          "sub": "ee51caaf-9680-42e7-bbe4-bdcb145711b9",
          "typ": "Refresh",
          "azp": "tutorial-frontend",
          "nonce": "21c805ac-2fa7-4d10-a460-1f5eec07a1e6",
          "auth_time": 0,
          "session_state": "b9fe89d5-f05c-4b04-973a-49941751c07c",
          "client_session": "6192215a-3f1a-4e73-a6cc-a3ca9643369d",
          "realm_access": {
                "roles": [
                  "manager",
                  "uma_authorization",
                  "user"
                ]
          },
          "resource_access": {
                "account": {
                  "roles": [
                        "manage-account",
                        "view-profile"
                  ]
                }
          }
        }
```


PAYLOAD id_token:


```
        {
          "jti": "9965d845-08fc-463a-8773-d90dd8d6b206",
          "exp": 1486128799,
          "nbf": 0,
          "iat": 1486128499,
          "iss": "http://localhost:8080/auth/realms/[realm id]",
          "aud": "tutorial-frontend",
          "sub": "ee51caaf-9680-42e7-bbe4-bdcb145711b9",
          "typ": "ID",
          "azp": "tutorial-frontend",
          "nonce": "21c805ac-2fa7-4d10-a460-1f5eec07a1e6",
          "auth_time": 1486128499,
          "session_state": "b9fe89d5-f05c-4b04-973a-49941751c07c",
         "at_hash": "7TKOKVSv1UuWpuGwRoABCg",
          "sid": "8d4fbc41-7417-4eab-8f39-7f929c66af07",
          "name": "John Doe",
          "preferred_username": "123456789",
          "locale": "en",
          "given_name": "John",
```

```
  "family_name": "Doe",
  "userProfile": {
    "lastName": "John",
    "firstName": "Doe",
    "ssin": "12345678910"
  }
}
```


PAYLOAD bearer token:


```
{
  "jti": "05d6bcf4-36b5-48c8-9490-258cf1d374f8",
  "exp": 1486130146,
  "nbf": 0,
  "iat": 1486129846,
  "iss": "http://localhost:8080/auth/realms/[realm id]",
  "aud": "tutorial-frontend",
  "sub": "ee51caaf-9680-42e7-bbe4-bdcb145711b9",
  "typ": "Bearer",
  "azp": "tutorial-frontend",
  "nonce": "51ccca50-2e08-4fb9-8374-beb09df2eba3",
  "auth_time": 1486128499,
  "session_state": "b9fe89d5-f05c-4b04-973a-49941751c07c",
  "acr": "0",
  "client_session": "3c13b729-d889-4392-8bea-af7839bdf5bb",
  "allowed-origins": [
        "http://localhost:8000"
  ],
  "realm_access": {
        "roles": [
          "manager",
          "uma_authorization",
          "user"
        ]
  },
  "resource_access": {
        "account": {
          "roles": [
                "manage-account",
                "view-profile"
          ]
        }
  },
  "given_name": "John",
  "family_name": "Doe",
  "userProfile": {
    "lastName": "John",
    "firstName": "Doe",
    "ssin": "12345678910"
  }
}
```

5.3.2 Client credentials flow

Client credentials are used when clients (applications and services) want to obtain access on behalf of themselves rather than on behalf of a user.

The client credentials flow, in abstract, contains the following steps.

1. The client authenticates with the authorization server and requests an access token from the token endpoint.
2. The authorization server authenticates the client, and if valid, issues an access token.

Confidential (server-side) clients can only use the client credentials grant type.

5.3.2.1 *Input arguments for Access Token request*

Endpoint: Token endpoint

HTTP method: POST.

The client makes a request to the token endpoint by presenting the credentials.

Field name	Description
grant_type	MANDATORY. Value MUST be set to “client_credentials”
client_assertion_type	MANDATORY. Value must be set to “urn:ietf:params:oauth:client-assertion-type:jwt-bearer”
client_assertion	MANDATORY. JWT token signed with the certificate of the partner’s client.

The JWT token must contain some elements as defined in ***https://tools.ietf.org/html/rfc7523#section-3.*** See section 5.3.3 Confidential client authentication for more information.

5.3.2.2 *Output arguments for access token (success)*

HTTP status code: 200 OK.

The parameters are included in the entity-body of the HTTP response using the "application/json" media type. They are serialized into a JSON structure by adding each parameter at the highest structure level. The order of parameters is irrelevant and can vary.

Field name	Description
access_token	MANDATORY. Access token.
token_type	MANDATORY. The type of the token issued. Always “bearer”.
expires_in	MANDATORY. The lifetime in seconds of the access token.
refresh_token	OPTIONAL. The refresh token, which can be used to obtain new access tokens using the same authorization grant.
refresh_expires_in	OPTIONAL. The lifetime in seconds of the refresh token.

5.3.2.3 *Output arguments for access token (error)*

For the list of errors: see section 8.

5.3.3 Confidential client authentication

If the client is a confidential client then he MUST authenticate to the token endpoint using the authentication method registered for its *client_id.*

A signed JWT will be used. The signed JWT is defined as value of the "client_assertion" parameter. This parameter MUST NOT contain more than one JWT.

A signed JWT is composed of 3 parts :

- Header
- Payload
- Signature

5.3.3.1 *header*

The header contains the following claims :

Field name	Description
Typ	MANDATORY. Value MUST be set to “JWT”
Alg	MANDATORY. The value must correspond to the algorithm used (usually RS256).
Kid	OPTIONAL. The id of the key used, it can be retrieved within the JWKS URL. When a JWKS URL is configured for a confidential client, new keys will be automatically reloaded by eHealth IAMConnect. If the JWKS URL contains more than one kid, the kid is mandatory in the header.

5.3.3.2 *payload*

The payload contains the following claims :

Field name	Description
Iss	MANDATORY. Value MUST correspond to the clientID.
Sub	MANDATORY. Value MUST correspond to the clientID.
Aud	MANDATORY Value must correspond to the identification URL of the realm in the correct environment. (example : “https://[BaseUrl]/auth/realms/[realm_id]”)
Jti	MANDATORY. Unique identifier for the token. The authorization server will ensure that JWTs are not replayed by maintaining the set of used "jti" values for the length of time for which the JWT would be considered valid based on the applicable "exp" instant.
Exp	MANDATORY. Expiration time. Limits the time window during which the JWT can be used, the expiration time has a maximum of 60 seconds.

5.4 Realm keys

The authentication protocols used by IAM Connect require cryptographic signatures. Asymmetric key pairs (a private and a public key) are used for this purpose.

Each realm has only one single active key pair at a time, but it may have more than one passive key pair as well. The active key pair is used to create new signatures, while the passive key pair may be used to verify previous signatures. This allows key rollover without any downtime/interruption.

The keys used to sign tokens (**all tokens are signed by the Realm key**) and all endpoints can be consulted online for each Realm: ***https://[BaseUrl]/auth/realms/[realm id]/.well-known/openid-configuration***

To view the active keys for a realm, you can use the well know endpoints: *jwks_uri*
https://[BaseUrl]/auth/realms/[realm id]/protocol/openid-connect/certs

5.4.1 Key rollover

For key rollover, keys should change at least with each major release at the eHealth platform.

For each realm, there is only a single active key pair at a time, but it may have more than one passive key pair. The active key pair is used to create new signatures, while the passive key pairs may be used to verify previous signatures.

When we add a new key, this key is considered as passive during a few days (1 to 7 days). After this period older keys are set as passive and the new one is active, meaning all new tokens will be signed with the new key. When a user authenticates to an application, the cookie is updated with the new signature.

When tokens are refreshed, new tokens are signed with the new keys.

All cookies and tokens will use the new key. Old keys are deleted one month after the new one has been created.

All clients are advised to retrieve the list of active certificates at least one time per day.

5.5 Validate access token

Once an access token has been obtained by the client and passed on to the target resource server, the latter must check whether the access token received is valid.

The keys used to sign tokens can be consulted online for each Realm. Access tokens are JSON Web Token (JWT).

General validation access token:

- JWT tokens can be validated offline:
  The signing key may be retrieved using the well-known endpoints : *jwks_uri*
  https://[BaseUrl]/auth/realms/[realm id]/protocol/openid-connect/certs
- In addition, clients can send the token to IAM Connect for validation. We use the following standard:
  OAuth 2.0 token Introspection (based on RFC 7662)
  This allows the client to verify that a token has not been revoked.
- With offline validation only, you obviously cannot perform this verification.
  Clients wanting to check revocation must use the following endpoint:
  https://[BaseUrl]/auth/realms/[realm id]/protocol/openid-connect/token/introspect
  URL therefor can also be retrieved via the well-known endpoints: *token_introspection_endpoint*
  Bearer-only clients, who want to use this endpoint, must be registered (with their public keys or their JWKS URL).

5.5.1 Token introspection

5.5.1.1 *Input arguments*

Endpoint: token_introspection_endpoint

HTTP method: POST.

The access token is passed on using the "application/x-www-form-urlencoded" format.

Field name	Description
Token	MANDATORY. The string value of the token. For access tokens, this is the "access_token" value returned from the token endpoint defined in OAuth 2.0 (RFC6749), Section 5.1.

5.5.1.2 *Output arguments (success)*

HTTP status code: 200 OK.

The server responds with a JSON object in “application/json” format with the following top-level members:

Field name	Description
Active	MANDATORY. Boolean indicator of whether or not the presented token is currently active.

Sub	OPTIONAL. Subject Identifier. A locally unique and never reassigned identifier within the Issuer for the End-User, which is intended to be consumed by the client, e.g., `24400320` or `AItOawmwtWwcT0k51BayewNvutrJUqsvl6qs7A4`. It MUST NOT exceed 255 ASCII characters in length. The `sub` value is a case sensitive string.
Aud	OPTIONAL. Audience(s) that this access token is intended for. It MUST be the OAuth 2.0 `client_id` of the Client.
Iss	OPTIONAL. Issuer Identifier for the Issuer of the response. The `iss` value is a case sensitive URL using the `https` scheme that contains scheme, host, and optionally, port number and path components and no query or fragment components.
Exp	OPTIONAL. The lifetime in seconds of the access token. For example, the value "3600" denotes that the access token will expire in one hour from the time the response was generated.
Scope	OPTIONAL. The scope of the access token (space-separated).
client_id	OPTIONAL. Client identifier for the OAuth 2.0 client that requested this token.
token_type	OPTIONAL. Type of the token.
Iat	OPTIONAL. Integer timestamp indicating when this token was originally issued.
Nbf	OPTIONAL. Integer timestamp indicating when this token is not to be used before
Jti	OPTIONAL. Provides a unique identifier for the JWT.

5.5.1.3 *Output arguments (error)*

HTTP status code: 401 Unauthorized.

If the protected resource uses OAuth 2.0 client credentials to authenticate to the introspection endpoint and its credentials are invalid, the authorization server responds with an HTTP 401 (Unauthorized).

If the protected resource uses an OAuth 2.0 bearer token to authorize its call to the introspection endpoint and the token used for authorization does not contain sufficient privileges or is otherwise invalid for this request, the authorization server responds with an HTTP 401 code.

5.5.1.4 *Example*

```
HTTP/1.1 200 OK
     Content-Type: application/json

     {
      "active": true,
      "client_id": "l238j323ds-23ij4",
      "username": "jdoe",
      "scope": "read write dolphin",
      "sub": "Z5O3upPC88QrAjx00dis",
      "aud": "https://protected.example.net/resource",
      "iss": "https://server.example.com/",
      "exp": 1419356238,
      "iat": 1419350238,
      "extension_field": "twenty-seven"
     }
```

```
HTTP/1.1 200 OK
     Content-Type: application/json

     {
      "active": false
     }
```

5.6 Consent

Consent may be enabled. When consent is enabled for a client, a user must give his/her permission before the client can participate in the authentication process.

After a user provides his credentials, the authorization server will display a screen requesting the client for a login and the identity information. The user can decide whether to grant the request.

From a technical perspective, consent is optional and can be configured by the client. In some cases (for example: when a client uses client credentials flow), the consent will not be prompted. However, in other cases (for example: when a client performs token eXchange on behalf of an end user to access other protected resources in his/her name), the consent MUST be prompted.

Consent revocation is available through a default client called 'account'. This 'account' client is available for each realm involving end user authentication:

https://[BaseUrl]/auth/realms/[realm id]/account

5.7 Token specification

5.7.1 IDToken

An ID token is a JSON Web Token (JWT) that contains claims about the authentication event. It MAY contain other Claims.

The generated IDTokens will contain the default info any IDToken should have according to the specs. You will find a description of the default info below.

Field name	Description
Iss	Identifier for the issuer of the response. The iss value is a case sensitive URL using the https scheme that contains scheme, host, and optionally, port number and path components and no query or fragment components.
Sub	Subject identifier. A locally unique and never reassigned identifier within the issuer for the End-User, which is intended to be consumed by the client, e.g., 24400320 or AItOawmwtWwcT0k51BayewNvutrJUqsvl6qs7A4. It MUST NOT exceed 255 ASCII characters in length. The sub value is a case sensitive string.
Aud	Audience(s) this ID token is intended for. It MUST contain the OAuth 2.0 client_id of the relying party as an audience value. It MAY also contain identifiers for other audiences. In the general case, the aud value is an array of case sensitive strings. In the common special case when there is one audience, the aud value MAY be a single case sensitive string.

Exp	Expiration time on or after which the ID Token MUST NOT be accepted for processing. The processing of this parameter requires that the current date/time MUST be before the expiration date/time listed in the value. Implementers MAY provide for some small leeway, usually no more than a few minutes, to account for clock skew. Its value is a JSON number representing the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time. See RFC 3339 for details regarding date/times in general and UTC in particular.
Iat	Time at which the JWT was issued. Its value is a JSON number representing the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time.
nonce	String value used to associate a Client session with an ID Token, and to mitigate replay attacks. The value is passed on through unmodified from the authentication request to the ID token. If present in the ID token, clients MUST verify that the nonce claim value is equal to the value of the nonce parameter sent in the authentication request. If present in the authentication request, authorization servers MUST include a nonce claim in the ID Token with the claim value being the nonce value sent in the authentication request. Authorization servers SHOULD perform no other processing on nonce values used. The nonce value is a case sensitive string.
Acr	Authentication context class reference. String specifying an authentication context class reference (ARC) value, identifying the authentication context class that the performed authentication is satisfied. The value "0" indicates the End-User authentication did not meet the requirements of ISO/IEC 29115 [ISO29115] level 1. Authentication using a long-lived browser cookie, for instance, is one example where the use of "level 0" is appropriate. Authentications with level 0 SHOULD NOT be used to authorize access to any resource of any monetary value. (This corresponds to the OpenID 2.0 PAPE [OpenID.PAPE] `nist_auth_level` 0.) An absolute URI or an RFC 6711 registered name SHOULD be used as the `acr` value; registered names MUST NOT be used with a different meaning than the registered one. Parties using this claim will need to agree upon the meanings of the values used, which may be context-specific. The `acr` value is a case sensitive string.
Azp	Authorized party - the party to whom the ID Token was issued. If present, it MUST contain the client ID of this party. This claim is only needed when the ID token has a single audience value and that audience is different from the authorized party. It MAY be included even when the authorized party is the same as the sole audience. The `azp` value is a case sensitive string containing a StringOrURI value.
Nbf	The "nbf" (not before) claim identifies the time before which the JWT MUST NOT be accepted for processing. The processing of the "nbf" claim requires that the current date/time MUST be after or equal to the not-before date/time listed in the "nbf" claim. Implementers MAY provide for some small leeway, usually no more than a few minutes, to account for clock skew. Its value MUST be a number containing a NumericDate value.
Jti	The "jti" (JWT ID) claim provides a unique identifier for the JWT. The identifier value MUST be assigned in a way, which ensures a negligible probability that the same value will be accidentally assigned to a different data object; if the application uses multiple issuers, collisions MUST be prevented among values produced by different issuers as well. The "jti" claim can be used to prevent the JWT from being replayed. The "jti" value is a case-sensitive string.

name	End-user's full name in displayable form including all name parts, possibly including titles and suffixes, ordered according to the end-user's locale and preferences.
given_name	Given name(s) or first name(s) of the end-user. Note that in some cultures, people can have multiple given names; all can be present, with the names being separated by space characters.
family_name	Surname(s) or last name(s) of the end-user. Note that in some cultures, people can have multiple family names or no family name; all can be present, with the names being separated by space characters.

IDToken :

For client involving IDP, the ID token may contain other claims depending on the profile chosen by the authenticated user (within eHealth IDP). You can find more information about extra claims in the document IAM Connect - Claim mappers. Here is a not exhaustive example:

```
{
  "jti": "9965d845-08fc-463a-8773-d90dd8d6b206",
  "exp": 1486128799,
  "nbf": 0,
  "iat": 1486128499,
  "iss": "http://localhost:8080/auth/realms/[realm id]",
  "aud": "tutorial-frontend",
  "sub": "ee51caaf-9680-42e7-bbe4-bdcb145711b9",
  "typ": "ID",
  "azp": "tutorial-frontend",
  "nonce": "21c805ac-2fa7-4d10-a460-1f5eec07a1e6",
  "auth_time": 1486128499,
  "session_state": "b9fe89d5-f05c-4b04-973a-49941751c07c",
  Authorization: Bearer  "at_hash": "7TKOKVSv1UuWpuGwRoABCg",
  "sid": "8d4fbc41-7417-4eab-8f39-7f929c66af07",
  "name": "john doe",
  "preferred_username": "xxxx",
  "locale": "en",
  "given_name": "John",
  "family_name": "Doe",
   "userProfile": {
        "lastName": "Doe",
        "firstName": "John",
        "ssin": "12345678910"
  }
}
```

5.7.2 AccessToken

An access token is a JSON web token (JWT) that contains credentials used to access protected resources. This token is issued to the client.

Tokens represent specific scopes and durations of access, granted by the resource owner, and enforced by the resource server and authorization server.

As for ID Token, it contains registered claim names (iss, sub, aud, exp, nbf, iat, jti). Other claims may be provided. You can find more information about extra claims in the document IAM Connect - Claim mappers.

Access Token may contain realm_access and resource_access. Realm_access contains realm roles (global roles applicable for all clients in the realm). Resource_access contains specific client roles. Client roles are grouped by client.

AccessToken example:

```
access_token:
       {
         "jti": "6401dc72-6090-4508-b10c-39c2a2d26850",
         "exp": 1486128799,
         "nbf": 0,
         "iat": 1486128499,
         "iss": "http://localhost:8080/auth/realms/[realm id]",
         "aud": "tutorial-frontend",
         "sub": "ee51caaf-9680-42e7-bbe4-bdcb145711b9",
         "typ": "Bearer",
         "azp": "tutorial-frontend",
         "nonce": "21c805ac-2fa7-4d10-a460-1f5eec07a1e6",
         "auth_time": 1486128499,
         "session_state": "b9fe89d5-f05c-4b04-973a-49941751c07c",
         "acr": "1",
         "client_session": "6192215a-3f1a-4e73-a6cc-a3ca9643369d",
         "allowed-origins": [
             "http://localhost:8000"
         ],
         "realm_access": {
             "roles": [
               "manager",
               "uma_authorization",
               "user"
             ]
         },
         "resource_access": {
             "account": {
               "roles": [
                    "manage-account",
                    "view-profile"
               ]
             }
         }
       }
```

5.8 UserInfo endpoint

The userinfo endpoint returns standard claims about the authenticated user and is protected by a bearer token. The endpoint can be found in the well-known endpoints: *userinfo_endpoint*

https://[BaseUrl]/auth/realms/[realm id]/protocol/openid-connect/userinfo.

For more details, refer to Userinfo Endpoint[6] section in the OpenID Connect specification.

5.9 Pseudonymization

In some use cases, the requesting client or the called API should not be able to uniquely identify the end user but he does need some kind of identifier, linkable to the end-user for further handling of the data.

In that case, we can work with pseudonyms[7]. This flow is only applicable for clients using authorization code flow.

Below is a description of possible pseudonymisation flows between a requesting client and a called API.

1. Client application sends request for authentication/authorization on behalf of an end-user (resource owner) to the auth endpoint of the AuthorizationServer
2. IAM Connect (AuthorizationServer) authenticates end-user and returns authorization_code to client application
3. Client application uses authorization_code to send request for a token to the token endpoint of the AuthorizationServer: an idtoken, accesstoken and refreshtoken is returned
4. (optional) Client application uses id token to show user identity in the available UI
5. Client application uses access token to send request to API
6. API uses access token to send request to the userinfo endpoint of AuthorizationServer
7. AuthorizationServer generates pseudonyms for claims with data that contain identifiers that can uniquely identify the end user and returns userinfo to the API (what the API does with the pseudonym is out of scope here)
8. (optional) Client application uses refresh token every x minutes to get a fresh access token

Important: during client registration, it is necessary to specify that your client must be able to use pseudonymized tokens: the client scope pseudo:profile is required for this purpose and it is not available by default.

5.9.1 Access token

The access token is sent to the API(s). It does not contain full identity in clear. Sensitive info is not provided.

No pseudonyms are provided in the access token because they are too long. They could cause problems when an API receives such a request as the access token is sent as a bearer token in the HTTP Header 'Authorization'. HTTP Headers are limited in length.

Content

- Identity of user for as far as the APIs, that receive it as authorization, are authorized to see this information
  - Claim 'preferred_username' is filtered out.
  - Claim 'userProfile' contains a set of claims with the current profile except for the claims that uniquely identify the end user
- Claim 'sub' is a pairwise subject identifier

Example :

```
{
  "exp": 1677843148,
```

[6] ***http://openid.net/specs/openid-connect-core-1_0.html#UserInfo***

[7] https://www.ehealth.fgov.be/ehealthplatform/fr/service-codage-anonymisation-et-ttp

```
  "iat": 1677842848,
  "auth_time": 1677842848,
  "jti": "8ff4eafe-9d2b-4c61-bc34-c505923f6a8b",
  "iss": "https://api.ehealth.fgov.be/auth/realms/healthcare",
  "sub": "968affcc-d59f-4c1a-bf1d-7a3f690a20d3",
  "typ": "Bearer",
  "azp": "sometestclient",
  "session_state": "78ea1f3c-8916-4a2f-be71-5a0b3b8feaa1",
  "acr": "urn:be:fgov:ehealth:1.0:acr:40",
  "allowed-origins": [
    "*"
  ],
  "scope": "openid pseudo:profile",
  "sid": "78ea1f3c-8916-4a2f-be71-5a0b3b8feaa1",
  "name": "John Doe",
  "given_name": "John",
  "locale": "nl",
  "family_name": "Doe",
  "userProfile": {
    "lastName": "Doe",
    "firstName": "John"
  }
}
```


5.9.2 Id token

The id token follows the same concept as the access token, sensitive info is not provided.

5.9.3 UserInfo

UserInfo endpoint can be called by anyone with a valid access token to receive user information linked to that access token.

Pseudonyms are used for all claims that contain sensitive information that could uniquely identify the end-user.

Content

- Contains filtered identity. Identifiers that can uniquely identify the end-user are replaced by a pseudonym in transit for the domain IAM. User info that is already in AccessToken are left out
  - Claim 'preferred_username' is pseudonymized
  - Claim 'userProfile' contains a set of claims with the current profile. The claims that uniquely identify the end-user are pseudonymized
- Claim 'sub' is a pairwise subject identifier

Example :


```
{
   "sub": "968affcc-d59f-4c1a-bf1d-7a3f690a20d3",
   "name": "John Doe",
   "preferred_username":
"eyJpZCI6IjJlMTM4ZTdmLTQyMDYtNDEzOC04OGVhLTMwZThlOTNjOGQ4MyIsImRvbWFpbiI6ImVoZWFsd
GhfdjEiLCJjcnYiOiJQLTUyMSIsImlhdCI6MTY3Nzg0Mjk3NCwiZXhwIjoxNjc3ODQzNTc0LCJ4IjoiQU5
HMDd3Vk5BakxDNU00L1Z4WGhJSENib3ZraGhpN1gzclVDZlpSZ3BTVDlSbGdIOW44U2xaMjdWNHdkWC83a
mlSeUMrV2UvVWlhOWFteXVOc21mdW5TTiIsInkiOiJKbjJuWE9qVnJ0TmU4clU3dzNxNUdiK2lxR2tLamh
```

jNTVuNnFLc0NJb21mL0N5THd5Yi9GVFRWOWZFZnhZNlZ2aFBuUGFVTXcrcXM5NVpONWFtcGwxdzQ9Iiwid
HJhbnNpdEluZm8iOiJleUpoYkdjaU9pSmthWElpTENKbGJtTWlPaUpCTWpVMlIwTk5JaXdpYTJsa0lqb2l
NakF5TXkwd01TSXNJbUYxWkNJNkltaDBkSEJ6T2k4dllYQnBMV2x1ZEM1bGFHVmhiSFJvTG1abmIzWXVZb
VV2Y0hObGRXUnZMM1l4TDJSdmJXRnBibk12WldobFlXeDBhRjkyTVNKOS4ud0VLX2lkX0dvYWhtdDlmOS5
UTWVqTGhWTTE3WGphZ2NoZmNYNlVCSks2dHJnNmpabVdrY1dWdHJ4Ym9Cd3JnX1R4ODhRU0VaRUZjVUVme
U9hSnRnUGVvbGpJaktaVlQyUHJyT3pvWXBiZkNSOUsyNGxIcWN5V0s4N29mSnZBbm9jOUpUd0NRR21oUUM
wMlVrX09HMEp3bXc5eHdRLXhYTXlFaHRKSTlrQ1U1WlNydHYwUWZVUjR0NHRyb3FrODd2eEZjVjM1VFJPT
FJPTUQ4a1VUSUJSd0UwcmRfNkpPTjVCN3k0Nlo4TEdDMEpYR3RFb2QxYklBX1p4N2pnRjh3OUZHUlhaVG1
xLWFhSHZHTEowdE9QMVlhNFAtN09GZmdOMi5KWWNWclpneTBnNXRUckxQa091UmVBIn0=",
"given_name": "John",
"locale": "nl",
"family_name": "Doe",
"userProfile": {"ssin":
"eyJpZCI6IjlhNmZlMjZjLWQxN2ItNGMyZS1hY2JiLTQwMDA2Y2UxYTc4NCIsImRvbWFpbiI6ImVoZWFsd
GhfdjEiLCJjcnYiOiJQLTUyMSIsImlhdCI6MTY3Nzg0Mjk3NCwiZXhwIjoxNjc3ODQzNTc0LCJ4IjoiQWR
CcHpMTW5OdlJQOU9JOTlHNXN2aVFIMFV4cTRvb256aWJQVlFiYkxZYUNMejVEdnBzdWowT0FZZjNhSDY2U
3B5bVo2ZDhWeEtzcm9ZSW5KRjBkTDZ0cyIsInkiOiJBYmlRY3djN2kxNkFNTThveHM2d1NYWUNhblRaTk9
yV0dpQkpRY2NNS2lxWTJ5dU96bUNGelJ4Q3pYcU4wR3hjOGEzMURlSHhnRXk2Q3k3T09MaG5ZN1F1Iiwid
HJhbnNpdEluZm8iOiJleUpoYkdjaU9pSmthWElpTENKbGJtTWlPaUpCTWpVMlIwTk5JaXdpYTJsa0lqb2l
NakF5TXkwd01TSXNJbUYxWkNJNkltaDBkSEJ6T2k4dllYQnBMV2x1ZEM1bGFHVmhiSFJvTG1abmIzWXVZb
VV2Y0hObGRXUnZMM1l4TDJSdmJXRnBibk12WldobFlXeDBhRjkyTVNKOS4uYUZINktiTkx5QWZnNVlZVi4
1SWozZ2ZMdDNQWlJOQWRZM0FIcnZDZUFyRHczZlJXLTU0dXE4RGNyNmpieTVIMU8zSkZ6QXJ0M1h2b29jc
01CX2tvYkVrdU45ZzRMcG5yUW5GYzBBMnVmOU9xR2lMVmdnSzN6VGI0Wmx4NFJWWlF0b2xiU1oza182MU5
PdGZHS2ZYb2dyNHZZWGRScXdtd0R2dURzMF9RaVJ4TWNIcnJjN1RwRFVSVWllZWx5Q2g2ZWZLOGNWNUtQd
GZTSHo5WnJLYVN3b2VaZzdiaHVVeXY1QlBhYmRDTTgwY1llZE5ZRnZhQ280bmxCNG00c1Q2VlpqZThfUDl
lY2kwa3BCS1o5Z3M1NFVlSG82VDF1QTdxdi45d2FwbHdINHdkRHhsOU1Yd0xzcDJBIn0="}
}

5.10 Token exchange

Exchanging tokens from one client to another is based on RFC-8693 [8].

Exchanging tokens can be initiated from the client that received the original token (i.e. source client) or from the client/API that received the token in the authorization header of one of its services.

Tokens can be exchanged by a client for himself as the audience or for another registered client/API as audience.

Exchanging tokens requested by other clients or requesting another audience than itself requires configuration of permissions in IAM Connect. Without these permissions, an error will be returned.

Furthermore, if the token that will be issued is for a client that requires end-user consent, the user must have given this consent beforehand as the exchange request is executed in a direct connection between the requesting client and IAM Connect. There is no UI available and no user interaction is involved. This means that an end user must at least have once authenticated to such a target client in a UI (e.g. a browser) at which point consent will have been requested and stored.

To use this functionality, security commitments must be signed (for more information, see section 6.2).

[8] ***https://datatracker.ietf.org/doc/html/rfc8693***

5.10.1 Exchange token from source

Client A, receiving a token from the authorization server after authentication of the end user, wants to exchange the token for another one with a specific content.

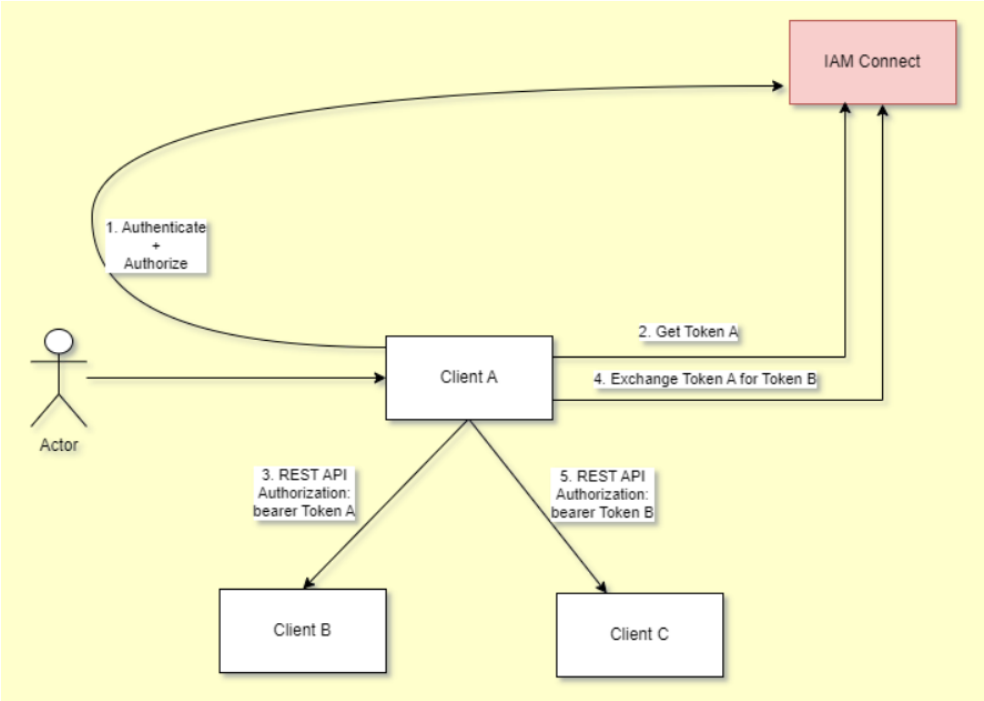


Figure 1 token exchange from source

1. Client A requests authorization from IAM Connect (standard OIDC authorization_code flow). During this step the user authenticates (if no active session available) and consents to the scope of the request (if consent was not yet given). IAM Connect returns an authorization_code to client A.
2. Client A requests the token (A), based on the authorization_code it received in previous step.
3. Client A sends token A as authorization with an API call to Client B.
4. Client A requests IAM Connect to exchange token A for token B.
   Client A must be configured in IAM Connect with the permission to use client C as audience of the tokens it requests.
5. Client A sends token B as authorization with an API call to Client C.

5.10.1.1 *Input arguments*

Endpoint: Token endpoint

HTTP method: POST.

It accepts form parameters (application/x-www-form-urlencoded) as input.

Field name	Description
grant_type	MANDATORY. Value MUST be set to “urn:ietf:params:oauth:grant-type:token-exchange”
requested_token_type	MANDATORY. Value must be set to “urn:ietf:params:oauth:token-type:access_token”
subject_token_type	MANDATORY. Value must be set to “urn:ietf:params:oauth:token-type:access_token”

subject_token	MANDATORY. A security token (active) that represents the identity of the party on behalf of whom the request is being made. It corresponds to the token issued to the client that requested authorization to IAM Connect (i.e. client A in Figure 1 token exchange from source)
Audience	MANDATORY. This parameter specifies the target client you want the new token minted for. It identifies the target client, registered at eHealth, that will receive the issued token in the authorization header of its API (i.e. client C in Figure 1 token exchange from source)

For public clients: add the following parameter :

Field name	Description
client_id	MANDATORY. It identifies the requesting client, registered at eHealth (i.e. client A in Figure 1 token exchange from source).

For confidential clients: the client (client A in Figure 1 token exchange from source) MUST authenticate to the token endpoint using the authentication method registered for his *client_id*. For authentication with a signed JWT, add the following parameters :

Field name	Description
client_assertion_type	MANDATORY. Value MUST be set to “urn:ietf:params:oauth:client-assertion-type:jwt-bearer”
client_assertion	MANDATORY. See section 5.3.3 Confidential client authentication

5.10.1.2 *Output arguments for access token (success)*

HTTP status code: 200 OK.

The parameters are included in the entity-body of the HTTP response using the "application/json" media type. They are serialized into a JSON structure by adding each parameter at the highest structure level. The order of parameters is irrelevant and can vary.

Field name	Description
access_token	MANDATORY. Access token.
token_type	MANDATORY. The type of the token issued. Always “Bearer”.
expires_in	MANDATORY. The lifetime in seconds of the access token.
refresh_expires_in	OPTIONAL. The lifetime in seconds of the refresh token. The value is always “0”, no refresh token is provided during a token exchange.

5.10.1.3 *Output arguments for access token (error)*

For the list of errors: see section 8.

5.10.2 Exchange token from target

Client B, receiving a token from client A, wants to exchange the token to send it to client C.

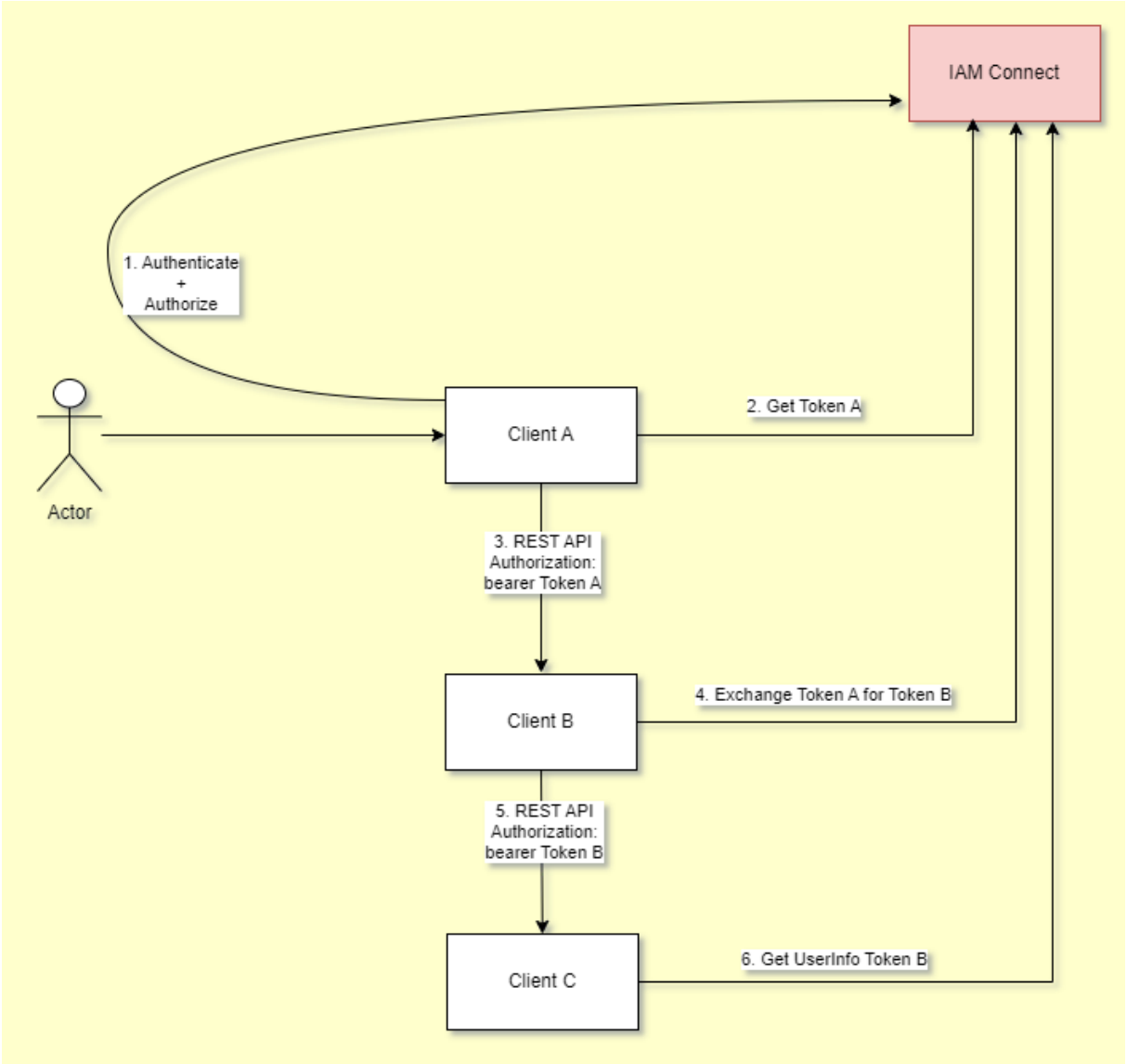


Figure 2 token exchange from target

1. Client A requests authorization from IAM Connect (standard OIDC authorization_code flow). During this step the user authenticates (if no active session available) and consents to the scope of the request (if consent not yet given). IAM Connect returns an authorization_code to client A.
2. Client A requests the token (A), based on the authorization_code it received in previous step.
3. Client A sends token A as authorization with an API call to Client B.
4. Client B requests IAM Connect to exchange token A for token B.
5. Client B sends token B as authorization with an API call to Client C.
6. (Optional) Client C requests IAM Connect for UserInfo, linked to token B.

Important notes :

- Client B must be configured as confidential client. Public clients are only authorized to exchange tokens that were issued to them (i.e. they are the holder of the token).
- Client B must be configured in IAM Connect with the permission.
  - to use tokens, issued to client A, as source to exchange to another token
  - to use client C as audience of the tokens it requests

5.10.2.1 *Input arguments*

Endpoint: Token endpoint

HTTP method: POST.

It accepts form parameters (application/x-www-form-urlencoded) as input.

Field name	Description
grant_type	MANDATORY. Value MUST be set to “urn:ietf:params:oauth:grant-type:token-exchange”
requested_token_type	MANDATORY. Value must be set to “urn:ietf:params:oauth:token-type:access_token”
subject_token_type	MANDATORY. Value must be set to “urn:ietf:params:oauth:token-type:access_token”
subject_token	MANDATORY. A security token (active) that represents the identity of the party on behalf of whom the request is being made. It corresponds to the token issued to the client that requested authorization to IAM Connect (i.e. client A in Figure 2 token exchange from target)
audience	MANDATORY. This parameter specifies the target client you want the new token minted for. It identifies the target client, registered at eHealth, that will receive the issued token in the authorization header of its API (i.e. client C in Figure 2 token exchange from target)

The client B (in Figure 2 token exchange from target) is a confidential client. The client MUST authenticate to the token endpoint using the authentication method registered for its *client_id*. For authentication with a signed JWT, add the following parameters :

Field name	Description
client_assertion_type	MANDATORY. Value MUST be set to “urn:ietf:params:oauth:client-assertion-type:jwt-bearer”
client_assertion	MANDATORY. See section 5.3.3 Confidential client authentication

5.10.2.2 *Output arguments for access token (success)*

HTTP status code: 200 OK.

The parameters are included in the entity-body of the HTTP response using the "application/json" media type. They are serialized into a JSON structure by adding each parameter at the highest structure level. The order of parameters is irrelevant and can vary.

Field name	Description
access_token	MANDATORY. Access token.
token_type	MANDATORY. The type of the token issued. Always “Bearer”.
expires_in	MANDATORY. The lifetime in seconds of the access token.
refresh_expires_in	OPTIONAL. The lifetime in seconds of the refresh token. The value is always “0”, no refresh token is provided during a token exchange.

5.10.2.3 *Output arguments for access token (error)*

Specific errors can be triggered during the switch operation (list might not be exhaustive):

Error/ErrorDescription	Explanation
invalid_token/"invalid subject_token"	The client is using an invalid subject_token_type.
invalid_request/"requested_token_type unsupported"	The client is using an invalid requested_token_type.
access_denied/"Client is not the holder of the token"	The client specifies a client_id different than the one present in the subject token.
invalid_token/"Invalid token"	The client is using an invalid token (expired token).

5.11 Profile switch

For client using authorization code flow, profile switch allows the client to switch profile for an authenticated user without using eHealth UI.

Switching profiles can only be done from/for the client that received the original token.

The client must be authorized to request these client scopes :

- *iam:exchange:profile:switch*
- *iam:exchange:profile*

Exchanging tokens that were requested by other clients or requesting another audience than itself require configuration of permissions in IAM Connect. Without these permissions, an error will be returned.

Also, the scope during exchange cannot be larger than the scope of the source token.

5.11.1 Process overview

A first accessToken can be obtained by using these client scopes *openid* and *iam:exchange:profile.*
Due to client scope *iam:exchange:profile*, the accessToken may contain a claim may_act with at least one profile. Each profile is identified by a unique key *sub* and contain a userProfile element providing information on the profile to exchange (for more information about the structure of the userProfile, see **I.AM Connect – Claim Mappers** documentation).

The client can then initiate a new call to switch the profile. The client scope *iam:exchange:profile:switch* must be requested. The input parameters are described in section 5.11.2.

If the operation succeeds, the profile selected is then the new actual one. If the first accessToken is refreshed, the new profile will be returned in the accessToken. The accessToken obtained during the switch operation might not be used.

If the operation fails, the client will receive an error. More information can be found in section 5.11.4.

5.11.2 Input arguments

Endpoint: Token endpoint

HTTP method: POST.

It accepts form parameters (application/x-www-form-urlencoded) as input.

Field name	Description
grant_type	MANDATORY. Value MUST be set to "urn:ietf:params:oauth:grant-type:token-exchange"

requested_token_type	MANDATORY. Value must be set to “urn:ietf:params:oauth:token-type:access_token”
subject_token_type	MANDATORY. Value must be set to “urn:ietf:params:oauth:token-type:access_token”
subject_token	MANDATORY. A security token (active) that represents the identity of the party on behalf of whom the request is being made.
requested_profile	OPTIONAL. This parameter specifies the unique id of a profile for the user authenticated in the subject_token. Only ids of profiles for which the client may act are acceptable. These ids will be available in the may_act claim of the subject_token when this claim was in scope when the subject_token was requested. Profile names can be requested from the IAM eXchange API (see **I.AM eXchange – Technical specification**). To switch to a citizen profile, use the value “citizen”.

5.11.3 Output arguments (success)

HTTP status code: 200 OK.

The parameters are included in the entity-body of the HTTP response using the "application/json" media type. They are serialized into a JSON structure by adding each parameter at the highest structure level. The order of parameters is irrelevant and can vary.

Field name	Description
access_token	MANDATORY. Access token.
token_type	MANDATORY. The type of the token issued. Always “Bearer”.
expires_in	MANDATORY. The lifetime in seconds of the access token.
refresh_expires_in	OPTIONAL. The lifetime in seconds of the refresh token. The value is always “0”, no refresh token is provided during a switch profile operation.

NB : The accessToken obtained might not be used to reach other API.

5.11.4 Output arguments (error)

Specific errors can be triggered during the switch operation (list might not be exhaustive):

Error/ErrorDescription	Explanation
invalid_request/”Invalid profile”	The client is using an invalid requested_profile. The value used cannot be found in the may_act of the authenticated user. The client should only use authorized values.
invalid_token/”invalid subject_token”	The client is using an invalid subject_token_type. The value must be *urn:ietf:params:oauth:token-type:access_token.*
invalid_request/”requested_token_type unsupported”	The client is using an invalid requested_token_type. The value must be *urn:ietf:params:oauth:token-type:access_token.*
access_denied/”Client is not the holder of the token”	The client specifies a client_id different than the one present in the subject token.
invalid_token/”Invalid token”	The client is using an invalid token (expired token).

5.12 Logout

A logout page asks end user confirmation to logout (unless id_token_hint is given and client is not configured for consent) and when confirmed, the logout endpoint logs out the authenticated user.

The user agent can be redirected to the endpoint, in which case the active user session is logged out. Afterwards the user agent is redirected back to the application.

The endpoint can also be invoked directly by the application. To invoke this endpoint directly the refresh token needs to be included as well as the credentials required to authenticate the client.

The endpoint can be found in the well-known endpoints: *end_session_endpoint*

Clients should use one of following options to configure the logout :

1. **Logout without parameters** => logout confirmation is requested + logout page is shown
   https://[BaseUrl]/auth/realms/[realm id]/protocol/openid-connect/logout

It is also possible to provide a redirect URI during the logout process :

2. **Logout with parameter post_logout_redirect_uri && id_token_hint** => user is redirected if uri is trusted and client not configured for consent
   https://[BaseUrl]/auth/realms/[realm id]/protocol/openid-connect/logout?id_token_hint=<id_token>&post_logout_redirect_uri=<redirect URI>
3. **Logout with parameter post_logout_redirect_uri && client_id**=> user is redirected if uri is trusted and client not configured for consent
   https://[BaseUrl]/auth/realms/[realm id]/protocol/openid-connect/logout?Client_id=<client_id>&post_logout_redirect_uri=<redirect URI>

Post logout redirect uri must match one of the patterns, configured as valid redirect uri.

id_token_hint must be a valid id_token, the client received it after authenticating the client.

If post_logout_redirect_uri given as parameter, the request must also add id_token_hint or client_id as parameter.

5.13 Timeout and lifespan

- SSO Session Idle

  If the user is inactive for longer than this timeout, the user session will be invalidated.

  Maximum time before a session is invalidated: 15 minutes.

  A client requesting authentication will bump the idle timeout. Refresh token requests will also bump the idle timeout.
- SSO Session Max

  Maximum time before a user session is expired and invalidated: 12 hours.
- Access Token Lifespan

  Maximum time before an access token is expired: 10 minutes.
- Client login timeout

Maximum time a client has to finish the access token protocol: 1 minute.

- Login timeout.

  Maximum time a user has to complete login related actions like update password or configure Time based One Time Password (TOTP): 30 minutes.
- Login action timeout.

Maximum time a user can spend on any one page in the authentication process: 5 minutes.

6 Risks and security

6.1 Security

6.1.1 Business security

If he development team adds an additional use case based on an existing integration, the eHealth platform must be informed at least one month in advance along with a detailed estimate of the expected load. This will ensure an effective capacity management.

If technical issues arise with the WS, the partner may obtain support from the contact center.

If the eHealth platform defines a bug or vulnerability in its software, we advise the partner to update his application with the latest version of the software within 10 business days.

If the partner discovers a bug or vulnerability in the software or web service provided by the eHealth platform , he is obliged to immediately contact and inform the eHealth platform. Under no circumstances is he permitted to publicly disclose this bug or vulnerability.

6.1.2 Recommendations for mobile clients

For native app, you can refer to ***https://tools.ietf.org/html/rfc8252***

For clients utilizing the authorization code grant, you should consider using Proof Key for Code Exchange (PKCE[9]).

For public clients, you must implement the PKCE.

6.2 Token exchange

When the end user gives his consent, the client (the trusted platform) might be allowed to exchange tokens (see section 5.10) during a given period.

A more comprehensive set of security requirements is given in “IAM eXchange Annex A – Security commitment from the Trusted Platform” and “IAM Connect Token eXchange – Security commitment”, both available on the portal.
(See https://www.ehealth.fgov.be/ehealthplatform/nl/service-i.am-identity-access-management)

These documents should be signed by a legal representative of the entity or by the information security consultant.

[9] ***https://tools.ietf.org/html/rfc7636***

7 Test and release procedure

7.1 Procedure

This chapter explains the procedures for testing and releasing an application in acceptation or production.

7.1.1 Initiation

If you intend to use the service of the eHealth platform, please contact ***info@ehealth.fgov.be***. The Project department will provide you with the necessary information and mandatory documents.

7.1.2 Development and test procedure

You have to develop a client in order to connect to our service. Most of the required info about the integration is published on the portal of the eHealth platform (***https://www.ehealth.fgov.be/ehealthplatform***).

In some cases, the eHealth platform provides you with a test case (see ***Request test case template***) in order for you to test your client before releasing it in the acceptance environment.

7.1.3 Release procedure

When development tests are successful, you can request to access the acceptance environment of the eHealth platform.

From this moment, you start integration and acceptance tests. The eHealth platform suggests testing during minimum one month.

For client involving end user authentication, these functionalities must be tested and validated :

- Consent page
  - Consent page displayed (for new user or consent not agreed)
  - Consent page not displayed if agreed
- Access token delivery by IAM Connect
- Access token consummation by the client
- Refresh token delivery by IAM Connect
- Refresh token consummation by the client
- Link to revoke consent must be proposed within the client
- Verify information in Account page[10]
  - Verify the list of application with consent
  - Consent can be revoked
- Logout
- Logout with redirect url

If token exchange is needed :

- Perform Token exchange
- Perform Token switch profile

If pseudonymization is needed

- Perform Pseudonymization

After successful acceptance tests, the partner sends his test results and performance results (including a sample of “eHealth request” and “eHealth answer”) by email to the designated point of contact at the eHealth platform.

[10] [BaseUrl]/realms/healthcare/account/

Then the eHealth platform and the partner agree on a release date. The eHealth platform then prepares the configuration for the production environment and provides the partner with the necessary information. During the release day, the partner provides the eHealth platform with feedback on the test and performance tests.

For further information and instructions, please contact: ***integration-support@ehealth.fgov.be***.

7.1.4 Operational follow-up

Once in production, the partner using the service of the eHealth platform for one of his applications will always test FIRST in the acceptance environment before releasing any adaptations of his application in production. In addition, he will inform the eHealth platform on the progress and test period.

8 Error and failure messages

The client MUST manage errors and MUST display in a correct way (when necessary) the error(s) to the end user.

For examples:

- the accessToken (linked to an end user) received does not match the client specifications (the chosen profile is not supported), the end user MUST be informed that his/her actual profile is not supported and the client SHOULD propose to the end user a link to change his/her profile
- the client cannot receive an error while calling the authorization endpoint or the token endpoint (http status code 503, http status code 404, http status code 400, …), the end user MUST be informed that there is a technical issue and MUST be informed about the way to introduce a support request.

8.1 Authorization endpoint error

Error	Error description
invalid_request	The request is missing a required parameter, includes an invalid parameter value, a parameter is included more than once, or is otherwise malformed.
invalid_client_credentials	The client is not correctly authenticate, issue with his credentials.
unauthorized_client	The client is not authorized to request an authorization code using this method.
access_denied	The resource owner or authorization server denied the request.
unsupported_response_type	The authorization server does not support obtaining an authorization code through this method.
invalid_scope	The requested scope is invalid, unknown, or malformed.
server_error	The authorization server encountered an unexpected condition that prevented it from fulfilling the request. (This error code is needed because a 500 Internal Server Error HTTP status code cannot be returned to the client via an HTTP redirect.)
temporarily_unavailable	The authorization server is currently unable to handle the request due to a temporary overloading or maintenance of the server. (This error code is needed because a 503 Service Unavailable HTTP status code cannot be returned to the client via an HTTP redirect.)

8.2 Token endpoint error

Error	Error description
invalid_request	The request is missing a required parameter, includes an unsupported parameter value (other than grant type), repeats a parameter, includes multiple credentials, utilizes more than one mechanism for authenticating the client, or is otherwise malformed.

invalid_client	Client authentication failed (e.g., unknown client, no client authentication included, or unsupported authentication method). The authorization server MAY return an HTTP 401 (Unauthorized) status code to indicate which HTTP authentication schemes are supported. If the client attempted to authenticate via the "Authorization" request header field, the authorization server MUST respond with an HTTP 401 (Unauthorized) status code and include the "WWW-Authenticate" response header field matching the authentication scheme used by the client.
invalid_grant	The provided authorization grant (e.g., authorization code, resource owner credentials) or refresh token is invalid, expired, revoked, and does not match the redirection URI used in the authorization request, or was issued to another client.
unauthorized_client	The authenticated client is not authorized to use this authorization grant type.
unsupported_grant_type	The authorization server does not support the authorization grant type.
invalid_scope	The requested scope is invalid, unknown, malformed, or exceeds the scope granted by the resource owner.

9 Recommendation

Here are some recommendations for the usea and integration of IAM Connect.

9.1 Refresh token

- Before sending a refresh token to the authorization server, its validity must be checked first. An expired refresh token should not be sent to the authorization server as it will result in an error.
- When requesting a new access token with the refresh token, the used credential must be registered onserver side, otherwise an error will arise.
- A refresh token can only be used once to get an access token. A new refresh token will be providedduring this process, so a next access token can be requested if needed.
- By default, the refresh token is valid for 30 minutes. Applying a sliding window mechanism to get a new refresh token in time is possible. As long as the refresh token remains valid for more than 5 minutes, no renewal is necessary. A client-side scheduled task can check the token's validity every minute, allowing for early detection and renewal attempts in the final minutes.
- By default, an access token is valid for 5 minutes. As long as it is valid (threshold of 30 seconds), it should be re-used. If a refresh token is available, a new access token can be requested therewith. This will not only result in a new access token but also in a new refresh token.. This operation will therefore impact the process of the sliding window of the refresh token (see previous point). If no valid refresh token is available, the user must request a new access token to the authorization endpoint.

9.2 Signed JWT

- The nbf claim is optional. If present, a clockSkew should be applied to prevent time synchronization issues. Use current time minus 5 seconds.
- The iat claim is optional. If present, a clockSkew should be applied to prevent time synchronization issues. Use current time minus 5 seconds.
- The exp claim should be short as signed JWT tokens are for one-time use. Use current time plus 10 minutes max.