FOD Beleid & Ondersteuning
SPF Stratégie & Appui

FAS
-
OIDC
Integration guide

Versie 1.1

Table of Contents

List of Figures

1 FAS as an authorization server

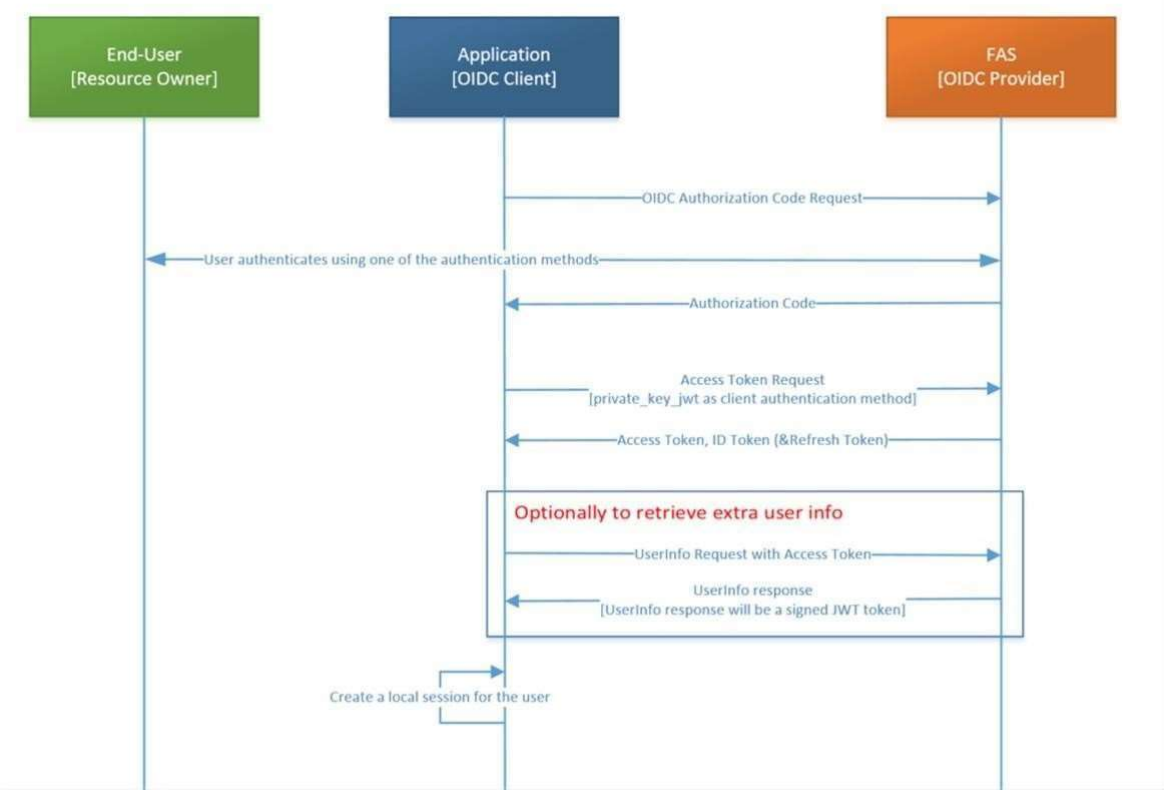

Figure 1: OpenID Connect flow diagram

1. The client application sends *an Authorization Code Request* towards the FAS Authorization server via the browser.
2. User authenticates using one of the authentication methods.
3. FAS authorization server sends an *Authorization Code* via the browser to the redirect-URI of the client application.
4. The client application exchanges the *Authorization Code* for an *Access token*, *Refresh token* and *ID token* using server to server communication and OIDC client authentication method *client_secret_basic*.
5. FAS sends an *Access token*, *Refresh token* and *ID token* via server to server communication to the client application.
6. The client application calls the *userinfo* endpoint of the FAS using the *Access Token*, also known as *Bearer Token*.
7. FAS sends a signed JWT with extra user information to the client application.
8. The client application should be able to create a local session for the user.

2 OpenID Provider Configuration

We statically provide configuration information about the OpenID Provider's configuration.
A default configuration is published on the following endpoints:

Integration environment:
https://idp.iamfas.int.belgium.be/fas/oauth2/.well-known/openid-configuration

Production environment:
https://idp.iamfas.belgium.be/fas/oauth2/.well-known/openid-configuration

It is recommended to host a local copy of this file when your application relies on constant availability of this endpoint data.

3 OpenID Connect Authorization Code Request and Response

FAS utilizes the Authorization Code Grant Type to obtain an access token to grant application to retrieve user data after authenticating. This chapter describes how to format the authorization code request and which data is returned from the authorization endpoint.

3.1 OPENID CONNECT AUTHORIZATION CODE REQUEST

Endpoint Production: https://idp.iamfas.belgium.be/fas/oauth2/authorize
Endpoint Integration: https://idp.iamfas.int.belgium.be/fas/oauth2/authorize

The authorization code can be obtained by performing a HTTP **GET** request towards the Authorization Code endpoint of the FAS (.../fas/oauth2/authorize) with the following query string parameters embedded in the request:

Parameter		
scope	MUST contain	openid
response_type	MUST be	code
client_id	SHALL be	nativeAppClientID
redirect_uri	SHALL include	The https scheme
state	MUST contain	An opaque value used to maintain state between the request and the callback
response_mode	SHALL NOT be used	
nonce	MUST be used	A nonce will be used for KPI monitoring reasons
display	SHALL NOT be used	
prompt	CAN be used (optional)	login
max_age	SHALL NOT be used	
ui_locales	CAN be used (optional)	“en” “de” “fr” “nl”
id_token_hint	SHALL NOT be used	
acr_values	SHALL be used	Which LoA will be required by the client (See chapter 3.1.2)
claims	SHALL NOT be used	

Figure 2: Table of parameters for authorize request

- The Authorization code request and response MUST NOT be signed.
- The “state” parameter SHALL be used to avoid cross-site request forgery attacks. The state parameter is an unguessable string known only to your application and check to make sure that the value has not changed between requests and responses. This is clearly stated in the “Security considerations” section 10.12 “Cross-Site Request Forgery” of RFC 6749 (OAuth 2.0) and in section 3.6 of RFC6819.
- The nonce parameter SHALL be used.

3.1.1 Scopes and Claims

Given below is the list of supported scopes that can be requested during the authorization code request. This will enable the RP later to retrieve claims from the **userinfo endpoint, returned in a signed JWT token** (see chapter 5).

- **openid**
  - This scope is a MUST if you want an ID token (specific scope in OAuth to upgrade your request to an OIDC request)
- **profile**
  - This scope will return the following claims:
    - surname
    - givenName
    - fedid
    - prefLanguage (SMA profile required for this field to be populated)
    - mail (SMA profile required for this field to be populated)
- **egovnrn**
  - This scope will only return the RRN/NRN or BIS number claim of the authenticated user. (Legacy scope: value can be derived from the ID token sub claim)
- **certificateInfo**
  - If the user authenticates using eID and the scope certificateInfo is requested FAS will return the following claims (if present in the eID certificate):
    - cert_issuer
    - cert_subject
    - cert_serialnumber
    - cert_cn
    - cert_givenname
    - cert_sn
    - cert_mail
- **citizen**
  - States that the end-user authenticates as a natural person
  - This scope is currently incompatible with the enterprise and roles scope.
  - This scope is default if a RP doesn't request the enterprise or citizen scope.
- **enterprise**
  - States that the request is made in the name of an enterprise
  - (roles and enterprise should be combined)
- **roles**
  - This is an explicit request from roles of the authenticating end-user
  - (roles and enterprise should be combined)

For the citizen, enterprise and roles scopes, the following pairs are allowed:

- roles + enterprise
- citizen only

Note: citizen is the default scope if a RP doesn't request a citizen or enterprise scope.

3.1.2 Acr_values

We made this value mandatory because that way the system allows you to login with eID, even in case our database becomes unavailable.

The list of supported acr_values between a RP and the FAS is given below:

- urn:be:fedict:iam:fas:Level500
- urn:be:fedict:iam:fas:Level450
- urn:be:fedict:iam:fas:Level400
- urn:be:fedict:iam:fas:Level300
- urn:be:fedict:iam:fas:Level200
- urn:be:fedict:iam:fas:Level100

Below you find a table, containing an overview of all authentication means offered and supported by FOD BOSA:

Security level	Authentication Means	Authentication contract
High	eID **(1)**	urn:be:fedict:iam:fas:Level500
	itsme® **(1)**	urn:be:fedict:iam:fas:Level450
	myID.be® Authenticator App Mail OTP SMS OTP **(2)**	urn:be:fedict:iam:fas:Level400
Low	Token **(3)**	urn:be:fedict:iam:fas:Level300
	Username/Password	urn:be:fedict:iam:fas:Level200
Without identification	Self-registration without usage of NRN	urn:be:fedict:iam:fas:Level100

Figure 3: Overview of authentication means

(1) The digital keys, **eID and itsme®, must be present** at each onboarding.
If the customer chooses a certain level, the keys with a higher technical level must also be offered as well.
(e.g.: the customer chooses level 400, then the FAS screen displays the keys of levels 400, 450 and 500).

(2) SMS OTP: the optional digital key.
The customer pays the cost of sending the SMS when authenticating with this key. This requires an agreement between the customer and the mobile operator, which charges the cost directly to the customer.
If you are interested in the offer of this digital key 'code by SMS', please ask BOSA for the corresponding annex.

(3) The token is in 'phase out' mode.
No new tokens can be activated by end users since 25/09/2020.

```
GET https://idp.iamfas.int.belgium.be/fas/oauth2/authorize
?response_type=code
&client_id=myclientid
&scope=openid%20profile
&acr_values=urn:be:fedict:iam:fas:Level500
&redirect_uri=https://www.google.com
&state=af0ifjsldkj
&nonce=1244542
```

Figure 4: Example of an authorize request

3.2 OPENID CONNECT AUTHORIZATION CODE RESPONSE

The response to the authorize request will be a HTTP redirect that results in a HTTP **GET** request to the redirect_uri that was provided for the authorization code request with the following parameters:

Parameter		Value
scope	WILL contain	The requested scopes
code	WILL contain	The authorization code
state	WILL contain	Must be the same value as in the authorization request
client_id	WILL contain	The client ID
iss	WILL contain	The issuer of the authorization code

```
GET redirect_uri (http(s)://…)
?code=31308323-c08e-431a-a5dc-2e7335795b43
&scope=openid%20profile
&iss=https%3A%2F%2Fidp.iamfas.int.belgium.be%2Ffas%2Foauth2
&state=af0ifjsldkj
&client_id=myclientid
```

Figure 5: Example of an authorize response

Note: The authorization code is by default only valid for 10 seconds after it has been issued.

4 OpenID Connect Access Token request and Response

The obtained authorization code obtained in chapter 2 can be used to receive an access token. This chapter describes how to format the access token request and which data is returned from the access token endpoint.

4.1 OPENID CONNECT ACCESS TOKEN REQUEST

Endpoint Production: https://idp.iamfas.belgium.be/fas/oauth2/access_token
Endpoint Integration: https://idp.iamfas.int.belgium.be/fas/oauth2/access_token

The OpenID Connect RFC states that there are 4 possible client authentication methods (used by Clients to authenticate to the Authorization Server when using the Token Endpoint)
FAS only support 'client_secret_basic' as client authentication method.

The access token can be obtained by performing a HTTP **POST** request to the token endpoint of the FAS (.../fas/oauth2/access_token) with the following query string parameters:

Parameters		**Value**
grant_type	MUST contain	authorization_code
code	MUST contain	The authorization code you received from the FAS
redirect_uri	MUST contain	Same redirect uri as used in the authorization code request
Headers		
Authorization	MUST contain	Basic base64(client id:client secret)
Content-Type	MUST contain	application/x-www-form-urlencoded

URL with query string parameters:
https://idp.iamfas.int.belgium.be/fas/oauth2/access_token?grant_type=authorization_code&code=HusR0KJVsNVHJ84myuMn5taiqi4&redirect_uri=https://redirecturi.be

type: POST
headers: Authorization: Basic v2xGZW50aWN6Y2xpZW50c2VjcmV0
Content-Type: application/x-www-form-urlencoded

Figure 6: Example of an access token request

The /oauth2/access_token endpoint requires authentication, supports basic authorization (a base64-encoded string of client_id / client_secret), client_id and client_secret passed as header values.

4.2 OPENID CONNECT ACCESS TOKEN RESPONSE

After receiving and validating a valid and authorized token request from the client, the Authorization Server returns a successful response that includes:

- **ID Token**
- **Access Token**
- **Refresh Token**

The response is returned in the `application/json` media type. An example of such response can be found in Figure 7.


```
{
  "scope":"profile openid",
 "access_token": "SlAV32hkKG",
 “refresh_token”: “absdgzegsfvsd”,
 "token_type": "Bearer",
 "expires_in": 3600,
  "id_token":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8vaWRwLmlhbWZhcy5pb
nQuYmVsZ2l1bS5iZTo4MC9mYXMvb2F1dGgyIiwiaWF0IjoxNTc4MzkwNDQ4LCJleHAiOjE2
MDk5MjY0NDgsImF1ZCI6InJwX2NsaWVudF9pZF9uYW1lIiwic3ViIjoiNjQwODA3MzU5NzUi
LCJhdWRpdFRyYWNraW5nSWQiOiI5YzMyZjUzYS1iZTk0LTRjZmItOWZjNy1lYzFiMDFiZTA4
MGEtMTQxNDU0MyIsInRva2VuTmFtZSI6ImlkX3Rva2VuIiwiYW1yIjoidXJuOmJlOmZlZGljd
DppYW06ZmFzOkxldmVsNTAwIiwibm9uY2UiOiIxMjQ0NTQ2IiwiY19oYXNoIjoiN1hMa1R4
cklqcmRCWUZiQUczaHg2ZyIsInNfaGFzaCI6ImJPaHRYOEY3M0lNalNQZVZBcXh5VFEiLCJhe
nAiOiJycF9jbGllbnRfaWRfbmFtZSIsImF1dGhfdGltZSI6IjE1NzgzOTA0MzciLCJyZWFsbSI6Ii8i
LCJ0b2tlblR5cGUiOiJKV1RUb2tlbiIsIm9yZy5mb3JnZXJvY2sub3BlbmlkY29ubmVjdC5vcHMi
OiJrNkgydkE1aUJUQ3hlWEphTUNIWUxTSVdqcklifQ.OnAtLjFh4vY2wvhUYDc6kinqyPDvfD
kirVx2qAudVL4"
 }
```


Figure 7: Example of an access token response

The response of the server contains:

- The original requested scopes
- The requested access token
- A refresh token that can be used to acquire a new access token (see c5)
- The type of the token (defaults to Bearer)
- A duration before the token is expired and no longer usable in seconds
- An ID token, identifying the client (see below)

The ID token is a JWT and is created (and thus signed, RS256 by default) by the Authorization Server itself. The structure of this token is header – content – signature.

Header of the ID token:


```
{
  "typ": "JWT",
  "kid": "v1IQ0WgmnJ2sV9HKqbi1oMvRiCE=",
  "alg": "RS256"
}
```


Figure 8: Header of decoded ID token

Content of the ID token:


```
{
  "at_hash": "K8raFeErC-rgimvj_2Q-ow",
  "sub": "01022335972",
  "auditTrackingId": "3f50d393-ab53-4826-817f-83110975fff7-933565",
  "amr": [
    "eid",
    "urn:be:fedict:iam:fas:Level500"
  ],
  "iss": "https://idp.iamfas.int.belgium.be/fas/oauth2",
  "tokenName": "id_token",
  "nonce": "n-0S3_CVA2Mj",
  "aud": "client_id",
  "c_hash": "OrPoMVFrpFFyd2Ti9zeIrQ",
  "acr": "0",
  "org.forgerock.openidconnect.ops": "NNO5qN0gD2s-aw-IkgQK0BG_uH8",
  "s_hash": "bOhtX8F73IMjSPeVAqxyTQ",
  "azp": "client_id",
  "auth_time": 1636019027,
  "realm": "/",
  "exp": 1636022643,
  "tokenType": "JWTToken",
  "iat": 1636019043
}
```


Figure 9: Content of decoded ID token

The payload of the ID token contains information about the client request:

- The **sub**ject of the request (either the NRN, BIS number or an email address)
- An **amr** field containing an array with the chosen authentication method and the associated level of assurance.
- The **iss**uer of the tokens (the authorization server)
- The intended **aud**ience of the tokens (in this case the client id)
- An authorization and **exp**iry time of the token

To enforce a LOA, the **relying party should always validate** if the minimal authentication level was reached while authenticating to the OpenID Provider (FAS).
See Chapter 7 for info on validating JWT tokens.

Signature of the ID token:

```
TV9NNJH2FE2I_BWFQ0BGSKXASFJNVZRPTK88TJNV9FM
```

Figure 10: Signature of the ID token

4.2.1 Verifying the access token, refresh token and the ID token

Endpoint Production: https://idp.iamfas.belgium.be/fas/oauth2/introspect
Endpoint Integration: https://idp.iamfas.int.belgium.be/fas/oauth2/introspect

4.2.1.1 Access / refresh token

An endpoint (.../fas/oauth2/introspect) is defined to retrieve metadata about a token, such as validity, approved scopes and the context in which the token was issued.

Given an access token, a client can perform an HTTP **POST** on .../fas/oauth2/introspect?token=access_token to retrieve a JSON object indicating the following:

- active -> Is the token active.
- scope -> A space-separated list of the scopes associated with the token.
- client_id -> Client identifier of the client that requested the token.
- user_id -> The user who authorized the token.
- token_type -> The type of token.
- exp -> When the token expires, in seconds since January 1 1970 UTC.
- sub -> Subject of the token (NRN or BIS number)
- iss -> Issuer of the token.

The POST request to FAS (.../fas/oauth2/introspect) with the following parameters:

Headers		**Value**
Authorization	MUST contain	basic base64(client id:client secret)
Content-Type	MUST contain	application/x-www-form-urlencoded
Body		
token	MUST contain	access token / refresh_token

The /oauth2/introspect endpoint also requires authentication, and supports basic authorization (a base64-encoded string of client_id:client_secret), client_id and client_secret passed as header values.

The following example demonstrates the /oauth2/introspect endpoint request:

```
URL:    https://idp.iamfas.int.belgium.be/fas/oauth2/introspect

type:   POST
headers:        Authorization: basic Y6xpZW50aWQ6Y2xpZW50c2VjcmV0
                Content-Type: application/x-www-form-urlencoded
body:   {
                token: bsHy5UlFdHaKs_qd6sBTY9RfyoQ
          }
```

Figure 11 Introspect request example

Example response:

```
{
 "active": true,
 "scope": "profile egovnrn",
 "client_id": "myOAuth2Client",
 "user_id": "91112345678",
 "token_type": "Bearer",
 "exp": 1419356238,
 "sub": "https://resources.example.com/",
 "iss": "https://idp.iamfas.belgium.be/fas"
 }
```

Figure 12 Introspect example response – active true

If the access token is no longer valid you'll receive:

```
{
 "active": false
}
```

Figure 13 introspect example response – active false

4.2.1.2 ID Token

See chapter 'Validating a JWT token' for the general validation of a JWT token.

5 OpenID Connect User Info request and Response

Endpoint Production: https://idp.iamfas.belgium.be/fas/oauth2/userinfo
Endpoint Integration: https://idp.iamfas.int.belgium.be/fas/oauth2/userinfo

The user info endpoint can be called to retrieve additional user info using the access token.
The user info response is a signed JWT token which will contain claims based on the requested scopes in the authorization code request.

5.1 OPENID CONNECT USER INFO REQUEST

Additional claims can be requested by performing a **GET** request to the user info endpoint of the FAS server (…/fas/oauth2/userinfo).

The following parameters can/should be added:

Headers		**Value**
Authorization	MUST contain	Bearer *'Access-Token'*

The following example demonstrates the /oauth2/introspect endpoint request:

URL: https://idp.iamfas.int.belgium.be/fas/oauth2/userinfo

type: GET
headers: Authorization: Bearer qY2xTpn8xhAC6dRnsiVZ5zbbsnI

Figure 14: Example of a user info request

5.2 OPENID CONNECT USER INFO RESPONSE

If the access token is still valid, the FAS will return a signed (RS256 by default) JWT user info token.
An example of such token and its contents can be found below.

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJwcmVmTGFuZ3VhZ2UiOiJubCIsIm1haWwiOiJzdXJuYW1lLMD
PDMVubmFtZUBib3NhLmZnb3YuYmUiLCJzdXJuYW1lIjoiUGhpbCIsIMDPDMVuTmFtZSI6IkNvdWxzb24iLCJp
c3MiOiJodHRwOi8vaWRwLmlhbWZhcy5pbnQuYmVsZ2L1bS5iZTo4MC9mYXMvb2F1dGggyIiwiZWdvdk5S
TiI6IjkxMDgyMjM1OTc1IiwiZmVkaWQiOiJsYzgzNjFmbDhiZGVmNGQzM2IxOGE1YWQ4YTAxOTgxMjdjN
2U1M2VlZCIsIMLHdCI6MTU3ODM4NTU0MSwiZXhwIjoxNTc4MzG5NzM3LCJqdGkiOiJmZTc3MTk2Mi02
YTYxLTRmM2ItYTQ1YS1jOGJmMDc5NjE0ZDYifQ.DLr8iiPHk5ocN-JYJ8BXr8J0f2C0ln-KecUb7glhJ9g

Figure 15: Example of a user info Response


```
{
 "alg": "RS256",
 "typ": "JWT",
 "kid": "7Or5FbLaxRPxtXafRQxn+o7GK9s="
}
```


Figure 16: Decoded header of an example user info token

```
{
  "sub": "6408075384",
  "prefLanguage": "en",
  "mail": phil.coulson@bosa.fgov.be,
  "cert_issuer": "SERIALNUMBER=201805, CN=Citizen CA, O=Certipost N.V./S.A.,
L=Brussels, C=BE",
  "givenName": "Phil",
  "iss": "https://idp.iamfas.belgium.be/fas/oauth2",
  "egovNRN": "6408075384",
  "fedid": "lc8361fl8bdef4d33b18a5ad8a0198127c7e53eed",
  "cert_subject":
"SERIALNUMBER=6408075384,GIVENNAME=Phil,SURNAME=Coulson,CN=Phil
Coulson (Authentication),C=BE",
  "aud": "your_rp_client_id",
  "cert_serialnumber": "6408075384",
  "surname": "Coulson",
  "cert_cn": "Phil Coulson (Authentication)",
  "cert_givenname": "Phil"
}
```


Figure 17: Decoded payload of example user info token


```
DLR8IIPHK5OCN-JYJ8BXR8J0F2C0LN-KECUB7GLHJ9G
```


Figure 18: Signature of example user info token

The claims requested for this example JWT;

- egovNRN
- profile
- roles
- enterprise
- certficateInfo

5.3 USER INFO RESPONSE VALIDATION

See chapter 'Validating a JWT token' for the general validation of a JWT token.

6 Requesting a new access token with a refresh token

Endpoint Production: https://idp.iamfas.belgium.be/fas/oauth2/access_token
Endpoint Integration: https://idp.iamfas.int.belgium.be/fas/oauth2/access_token

To prevent excessive calls made to the token endpoint, you can **validate** your access / refresh token against the **introspect endpoint** as described in chapter 3.2.1 before retrieving a new **set** of tokens. This will reduce the amount of calls made from / to the FAS.

Note: swapping a refresh token for a new access token also invalidates the current refresh token. Upon refresh, a new ID, refresh and access token is returned.

A new (access) token can be obtained by performing a HTTP **POST** request to the token endpoint of the FAS (.../fas/oauth2/access_token) with the following parameters:

Headers		**Value**
Authorization	MUST contain	Basic base64(client id:client secret)
Content-Type	MUST contain	application/x-www-form-urlencoded
Body		
grant_type	MUST contain	refresh_token
refresh_token	MUST contain	The refresh token of the user you want to obtain a new access token for.

Example request:

```
URL:    https://idp.iamfas.int.belgium.be/fas/oauth2/access_token
type:   POST
header: Authorization: Basic moxpZW50aWK6Y2xpZW50c2VjcmV0
body:   {
                "grant_type”: "refresh_token",
                "refresh_token”: refresh-token
        }
```

Figure 19 Refresh token request example

After receiving and validating a valid and authorized refresh token request from the client, the Authorization Server returns a successful response that includes a new Access Token and a new Refresh Token. The response uses the `application/json` media type.


```
{
   "access_token": "34847c38-cc17-46fe-991e-698711aa856e ",
   "refresh_token": " bdcffbb4-6609-438b-bea3-5ff7dcf07d31",
   "scope": "enterprise openid profile roles egovnrn",
   "id_token":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8vaWRwLmlhbWZhcy5pbnQuYmVsZ2l
1bS5iZTo4MC9mYXMvb2F1dGgyIiwiaWF0IjoxNTc4MzkwNDQ4LCJleHAiOjE2MDk5MjY0NDgsImF1ZCI
6InJwX2NsaWVudF9pZF9uYW1lIiwic3ViIjoiNjQwODA3MzU5NzUiLCJhdWRpdFRyYWNraW5nSWQiOiI
5YzMyZjUzYS1iZTk0LTRjZmItOWZjNy1lYzFiMDFiZTA4MGEtMTQxNDU0MyIsInRva2VuTmFtZSI6ImlkX3
Rva2VuIiwiYW1yIjoidXJuOmJlOmZlZGljdDppYW06ZmFzOkxldmVsNTAwIiwibm9uY2UiOiIxMjQ0NTQ2I
iwiY19oYXNoIjoiN1hMa1R4cklqcmRCWUZiQUczaHg2ZyIsInNfaGFzaCI6ImJPaHRYOEY3M0lNalNQZVZ
BcXh5VFEiLCJhenAiOiJycF9jbGllbnRfaWRfbmFtZSIsImF1dGhfdGltZSI6IjE1NzgzOTA0MzciLCJyZWFsbSI
6Ii8iLCJ0b2tlblR5cGUiOiJKV1RUb2tlbiIsIm9yZy5mb3JnZXJvY2sub3BlbmlkY29ubmVjdC5vcHMiOiJrNkg
ydkE1aUJUQ3hlWEphTUNIWUxTSVdqckIifQ.OnAtLjFh4vY2wvhUYDc6kinqyPDvfDkirVx2qAudVL4",
   "token_type": "Bearer",
   "expires_in": 3599
}
```


Figure 20 Refresh token response example

7 Validating a JWT token

A JWT received from the authorization server has to be validated to ensure the content, sender and validity of the token.

7.1 CHECKING THE CONTENT OF THE TOKEN

- The Issuer (the appropriate FAS url) MUST exactly match the value of the iss (issuer) Claim.
- The Client MUST validate that the aud (audience) Claim contains its client_id value registered at the Issuer identified by the iss (issuer) Claim as an audience. The ID Token MUST be rejected if the ID Token does not list the Client as a valid audience, or if it contains additional audiences not trusted by the Client.
- The alg value SHOULD be the default of RS256.
- The current time MUST be before the time represented by the exp Claim.
- The iat Claim can be used to reject tokens that were issued too far away from the current time, limiting the amount of time that nonces need to be stored to prevent attacks. The acceptable range is Client specific.
- If a nonce value was sent in the Authentication Request, a nonce Claim MUST be present and its value checked to verify that it is the same value as the one that was sent in the Authentication Request. The Client SHOULD check the nonce value for replay attacks.

7.2 CHECKING THE SIGNATURE OF THE TOKEN

As mentioned before, a JWT consists of three different parts separated by a dot: the header, the payload and the signature. In order to check the authenticity of the token we will need all three parts of the token.

The signature must be checked to validate if this token was issued by the FAS OpenID Provider and not a malicious party. During this check the public key of the authorization server will be used.

7.2.1 Obtain the public key

The public key of the authorization server can be retrieved from the JWK URI:

Production: https://idp.iamfas.belgium.be/fas/oauth2/connect/jwk_uri
Integration: https://idp.iamfas.int.belgium.be/fas/oauth2/connect/jwk_uri

A **get** request to the above URL will respond with an `application/json` object containing an array of json objects containing the following:

Kty: The type of the public key
Kid: the id of this key object
use: The use of this key (e.g.: sig(nature))
x5t: base64url-encoded SHA-1 thumbprint (a.k.a. digest) of the DER encoding of an X.509 certificate
x5c: contains a chain of one or more PKIX certificates. The certificate chain is represented as a JSON array of certificate value strings. Each string in the array is a base64-encoded DER PKIX certificate value. The JWK uri endpoint from FAS returns only 1 certificate.
n: The modulus of the public key
e: The exponent of the public key
Alg: The specific signing algorithm


```
{
"keys":[
{
"kty":"RSA",
"kid":"GDeFBZrcfqxZ2CIPBYYMgjRhwnY=",
"use":"sig",
"x5t":"tSrE6n0UOizCaNTrIA_KyWi4SSQ",
"x5c":[
"MIIIUDCCBjigAwIBAgIUIQ1JHx9W6UdzDvicCqdhuNk2BegwDQYJKoZIhvcNAQELBQAwUzELMAkGA1UEBh
MCQkUxIDAeBgNVBAoTF1F1b1ZhZGlzIFRydXN0bGluayBCVkJBMSIwIAYDVQQDExlRdW9WYWRpcyBFdXJvc
GUgU1NMIENBIEcxMB4XDTE4MDQwNjA5MTUwM1oXDTIwMDQwNjA5MjQwMFowgboxCzAJBgNVBAYTAkJFMScw
JQYDVQQIDB5CcnVzc2VscyBIb29mZHN0ZWRlbGlqayBHZXdlc3QxEDAOBgNVBAcMB0JydXNzZWwxOTA3BgN
VBAoMMEZlZGVyYWxlIE92ZXJoZWlkc2RpZW5zdCBCZWxlaWQgZW4gT25kZXJzdGV1bmluZzEQMA4GA1UECw
wHQk9TQSBEVDEjMCEGA1UEAwwac2FtbC5pYW1mYXMuaW50LmJlbGdpdW0uYmUwggEiMA0GCSqGSIb3DQEBA
QUAA4IBDwAwggEKAoIBAQC6uywzoGUYq7H3N9PAyP4UsEkWbGKT4Yg+vVsgOubPsEaRtu+seUqFnsjDmSTI
uKs9ghuiudf5h6L9GRZnTd7FwEUDzoL3qcENYWNjj0VZls4rXQD92/1A7iUf6wSLEl1Y52gznRf8vb3n5lV
mpw4dwHn+Q5P9E8z3sTKHf1mxIHJ4myIIOGmTElHhbAFYmJGEn4F6xdep2ZtgN0CU20rhUxqa/VT1S3Q+5k
bwYDV4Ztdezk4OTiAc0VJy6ypXKiIONbj3RR8TQiSN7pH8sPYb6B29agnQDtyPS+umV3V042OroaKoD/zhq
Os3twaULSHBAiJzVO6pDXfAifzDEaoBAgMBAAGjggOyMIIDrjB0BggrBgEFBQcBAQRoMGYwOAYIKwYBBQUH
MAKGLGh0dHA6Ly90cnVzdC5xdW92YWRpc2dsb2JhbC5jb20vcXZldXNzbDEuY3J0MCoGCCsGAQUFBzABhh5
odHRwOi8vb2NzcC5xdW92YWRpc2dsb2JhbC5jb20wHQYDVR0OBBYEFEVrFf+MDMmlqPC6XfcMrTJQ+3UxMA
wGA1UdEwEB/wQCMAAwHwYDVR0jBBgwFoAUQtOHc+Ge0z5c/rBwu8reIbokeTYwWwYDVR0gBFQwUjBGBgwrB
gEEAb5YAAJkAQEwNjA0BggrBgEFBQcCARYoaHR0cDovL3d3dy5xdW92YWRpc2dsb2JhbC5jb20vcmVwb3Np
dG9yeTAIBgZngQwBAgIwOwYDVR0fBDQwMjAwoC6gLIYqaHR0cDovL2NybC5xdW92YWRpc2dsb2JhbC5jb20
vcXZldXNzbDEuY3JsMA4GA1UdDwEB/wQEAwIFoDAdBgNVHSUEFjAUBggrBgEFBQcDAgYIKwYBBQUHAwEwJQ
YDVR0RBB4wHIIac2FtbC5pYW1mYXMuaW50LmJlbGdpdW0uYmUwggH2BgorBgEEAdZ5AgQCBIIB5gSCAeIB4
AB3ALvZ37wfinG1k5Qjl6qSe0c4V5UKq1LoGpCWZDaOHtGFAAABYppGCegAAAQDAEgwRgIhAIlG1K+YFPD0
IwLVV6MI7jI5bz1XZg41Nmwl0QHeufk/AiEA1gIRW2J5doX4xrXMoYPtuki3Hoj1zXo4ate9qR9H4j0AdQC
kuQmQtBhYFIe7E6LMZ3AKPDWYBPkb37jjd80OyA3cEAAAAWKaRgn7AAAEAwBGMEQCIFh3VvagYu1QnRVulk
fMH5W8HcY6vAYhC4BqMBIpeE3iAiAPZIud+XUY0lIBlNuySi7EafSFEsJHZDza6G8dmpUZ0QB1AFYUBpov1
8Ls0/XhvUSyPsdGdrm8mRFcwO+UmFXWidDdAAABYppGC9kAAAQDAEYwRAIgG/4K5DGh6jA0wSGqQw6BR7jy
0i4wxAROf9bmyQcB0yUCIFW901oY/L/C2dHU+4vRdsbbDGI750ioneJEm+Kd6Z9KAHcAb1N2rDHwMRnYmQC
kURX/dxUcEdkCwQApBo2yCJo32RMAAAFimkYMeQAABAMASDBGAiEA/562+cD0hnYj/GKHGAqDW1zCX/e+sI
hX+48p6rOha3sCIQDOrNdNeVS/2xWDvZ90zyZcl87Myb6OtRX+z4ZGpYPmbTANBgkqhkiG9w0BAQsFAAOCA
gEAMMLlr4P7sug/wi8Dvdx792dglntuk6ULXVZ16BbxF2aL9HXWp6T5SKfEhXBhtToG2Uv81cX4jwbynhyr
```

```
XNZdgHHoT0bMkWY8/rXDaFICqp9xCuLb1wvl1KHI85beuqADJNO+n91No85MJukwLchlZdhfp6HgKNWi6YC
PPVE9IFqSJK/TjV+rTbnutF6Kz45d7xGhz7iNTDKfQnd9tqwn94QlUsT9dG0hqLU0f6Hbf1vB7U5RU2UN/f
cMyEqhuYc4FqLy798TS0/7+LQ/kMDT0ATIxECQdOOa8W38x93ulFHrgfZz5SdvETsmI5x4RY18sYV53Kaqu
vbrHIFY2PAiJXMLXVIVe4xFDQv2+lbXpBcQIyaSu7wY2WJu5QlDnWU862n4uocBHreGYK7N7cjWoOy651eK
qr5/JnLtRXdU51GKvcPY8BpQB1XDtuO1DBotxmi206uab+9W+tClRguZHqmjhoD4AJAfhE1PgNIN6MAEik4
Q3fuwAJQMZmV+2zBWwuW0mytwfXgvafKvmXOS9UjIkQgeHiAlo56HzY/yAA53YOF64ZUzslqcAr5BQLGBbc
OB5UA5tTksMVg532z8W1wK0TG+A7DPtwx7uM+inzO+fy2vdfqZxoiuzD8WRDEUR3W+tNrYkvXy/jYOoxsos
jS0wYcxtBrfB21ZbkxkgVA="
],
"n":"urssM6BlGKux9zfTwMj-FLBJFmxik-GIPr1bIDrmz7BGkbbvrHlKhZ7Iw5kkyLirPYIbornX-
Yei_RkWZ03excBFA86C96nBDWFjY49FWZbOK10A_dv9QO4lH-sEixJdWOdoM50X_L295-
ZVZqcOHcB5_kOT_RPM97Eyh39ZsSByeJsiCDhpkxJR4WwBWJiRhJ-
BesXXqdmbYDdAlNtK4VMamv1U9Ut0PuZG8GA1eGbXXs5ODk4gHNFScusqVyoiDjW490UfE0Ikje6R_LD2G-
gdvWoJ0A7cj0vrpld1dONjq6GiqA_84ajrN7cGlC0hwQIic1TuqQ13wIn8wxGqAQ",
"e":"AQAB",
"alg":"RS256"
},
…
```


Figure 21 JWK_uri response

With this information, the public key can be reconstructed with the modulus and the exponent. Since the jwk URL returns an array of key objects, we have to find the correct key. We do this with the `kid` field in the header of the JWT token. This value should match with one of the `kid` fields of the key objects in the array.

In the past we used the exponent and modulus to calculate the public key, now this value is directly available DER encoded as the x5c value.
The exponent and modulus are still available for legacy applications.

Example given in Java code:

```
public static PublicKey getPublicKey(String modulusB64u, String exponentB64u) throws Exception {
    byte exponentB[] = Base64.getUrlDecoder().decode(exponentB64u);
    byte modulusB[] = Base64.getUrlDecoder().decode(modulusB64u);
    BigInteger exponent = new BigInteger(toHexFromBytes(exponentB), 16);
    BigInteger modulus = new BigInteger(toHexFromBytes(modulusB), 16);

    //Build the public key
    RSAPublicKeySpec spec = new RSAPublicKeySpec(modulus, exponent);
    KeyFactory factory = KeyFactory.getInstance("RSA");
    PublicKey pub = factory.generatePublic(spec);

    return pub;
}
```

Figure 22: Example of a public key builder

7.2.2 Validating the signature

Once the public key is obtained, it can be used to verify the signature of the token.

Example given in JAVA:

```
public boolean verifyJWT (String exponentB64u, String modulusB64u,String jwt) throws Exception{
    //Build the public key from modulus and exponent
    PublicKey publicKey = getPublicKey (modulusB64u,exponentB64u);

    //print key as PEM (base64 and headers)
    String publicKeyPEM =
        "-----BEGIN PUBLIC KEY ---- \n"
      + Base64.getEncoder().encodeToString(publicKey.getEncoded()) +"\n"
      + "-----END PUBLIC KEY---- ";
    System.out.println( publicKeyPEM);

    //get signed data and signature from JWT
    String signedData = jwt.substring(0, jwt.lastIndexOf("."));
    String signatureB64u = jwt.substring(jwt.lastIndexOf(".")+1,jwt.length());
    byte signature[] = Base64.getUrlDecoder().decode(signatureB64u);

    //verify Signature
    Signature sig = Signature.getInstance("SHA256withRSA");
    sig.initVerify(publicKey);
    sig.update(signedData.getBytes());
    boolean v = sig.verify(signature);
    return v;
}
```

Figure 23: Example of a JWT verifier

If the signature is valid we can assume the signature belongs to the JWT token and that is was signed by the authorization server. In that case we can continue.
If not, an error message should be returned to the client.

8 OIDC Logout

Terminating an OIDC session on the authorization server is done via a single GET call to the endSession endpoint:

Endpoint Production: https://idp.iamfas.belgium.be/fas/oauth2/connect/endSession
Endpoint Integration: https://idp.iamfas.int.belgium.be/fas/oauth2/connect/endSession

The following parameters can/should be added:

Parameters		Value
id_token_hint	MUST contain	The session ID_token (see 4.2)
post_logout_redirect_uri*	SHALL include	The https scheme

(*)Make sure the post_logout_redirect_uri is communicated during the onboarding process. We need to whitelist the post logout redirect uri for each relying party.

```
https://idp.iamfas.int.belgium.be/fas/oauth2/connect/endSession?id_token_hint
=EYJHBGCIOIJSUZI1NIISIMTPZCI6IJFLOWDKAZCIFQ.EWOGIMLZC
   YI6ICJODHRWOI8VC2VYDMVYLMV4YW1WBGUUY29TIIWKICJZDWIIOIAIMJQ4MJG5
   NZYXMDAXIIWKICJHDWQIOIAICZZCAGRSA3F0MYISCIAIBM9UY2UIOIAIBI0WUZZ
   FV3PBMK1QIIWKICJLEHAIOIAXMZEXMJGXOTCWLAOGIMLHDCI6IDEZMTEYODA5NZ
   AKFQ.GGW8HZ1EUVLUXNUUIJKX_V8A_OMXZR0EHR9R6JGDQROOF4DAGU96SR_P6Q
   NQEGPE-GCCMG4VFKJKM8FCGVNZZUN4_KSP0AAP1TOJ1ZZWGJXQGBYKHIOTX7TPD
   QYHE5LCMIKPXFEIQILVQ0PC_E2DZL7EMOPWOAOZTF_M0_N0YZFC6G6EJBOEOROSD
   K5HODALRCVRYLSRQAZZKFLYUVCYIXEOV9GFNQC3_OSJZW2PAITHFUBEEBLUVVK4
   XUVRWOLRLL0NX7RKKU8NXNHQ-RVKMZQG&post_logout_redirect_uri=https://www.google.com
```

Figure 24: Example of a user endSession request

Make sure you terminate the local session before redirecting towards the endSession endpoint of the FAS. We'll redirect the user to the post_logout_redirect_uri after we've terminated the user's FAS session.