FOD Beleid & Ondersteuning

SPF Stratégie & Appui

FAS
-
OIDC
Integration Guide

14 June 2024

Version 6.5

Table of Contents

About this Document

This guide is an OpenID Connect (OIDC) integration guide created to help Relying Parties working with FPS BOSA's Federal Authentication Service (FAS). This document is not a substitution to OIDC documentation and will not document the whole standard. Its scope is limited to what is required to work with the FAS.

Consequently, the knowledge of the OIDC standard is mandatory and statements like "*Clients SHOULD ignore unrecognized response parameters*", *"Any parameters used that are not understood MUST be ignored by the Client" or "Any members used that are not understood MUST be ignored"* are still relevant. And you should expect for instance, **more attributes** than initially requested, or **unordered attributes,** in the FAS responses.

The official specification remains the reference document. If you want to learn more about OpenID Connect, you may consult the following URL: https://openid.net/specs/openid-connect-core-1_0.html. Additionally, here is the list of Certified OIDC Implementations: https://openid.net/developers/certified-openid-connect-implementations.

FAS as an Authorization Server

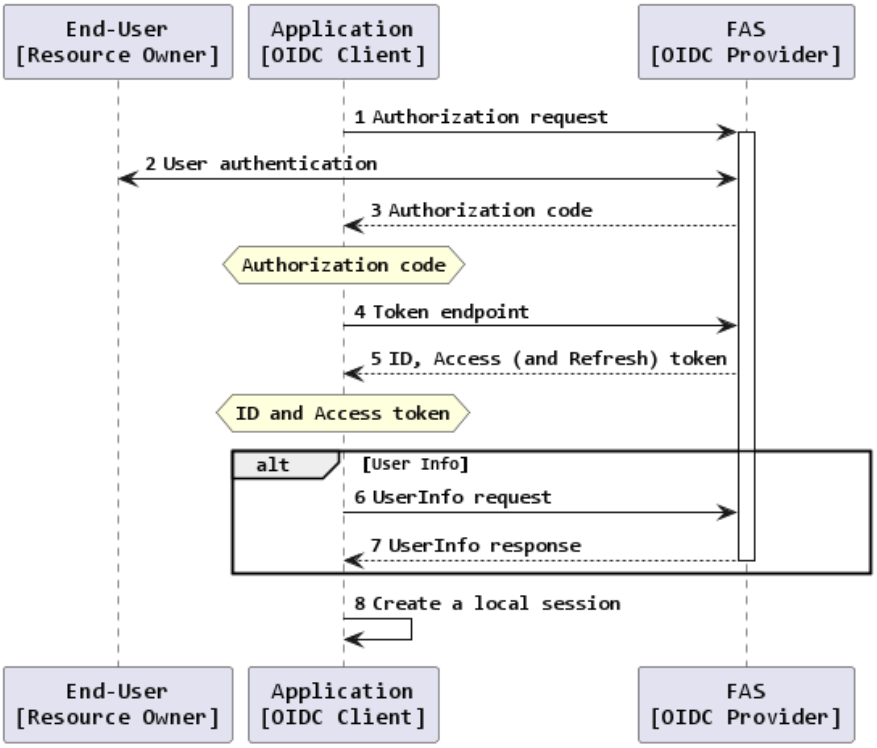

Figure 1 OIDC authorization request schema

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 available.
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. Server to server communication and OIDC client authentication method `client_secret_basic` are used.
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.

Remark: iframes technology was never supported by FAS and is not compatible actually.

OpenID Provider

Configuration information about the OpenID Provider is published on the following endpoints:

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

Table 1 OpenID provider's configuration information links

This is static information for machines and programs. It is recommended to host a local copy of this file when your application relies on constant availability of this endpoint data.

Endpoints

Endpoint	Environment	URL
Authorize	Integration	https://idp.iamfas.int.belgium.be/fas/oauth2/authorize
	Production	https://idp.iamfas.belgium.be/fas/oauth2/authorize
Token	Integration	https://idp.iamfas.int.belgium.be/fas/oauth2/access_token
	Production	https://idp.iamfas.belgium.be/fas/oauth2/access_token
Introspect	Integration	https://idp.iamfas.int.belgium.be/fas/oauth2/introspect
	Production	https://idp.iamfas.belgium.be/fas/oauth2/introspect
UserInfo	Integration	https://idp.iamfas.int.belgium.be/fas/oauth2/userinfo
	Production	https://idp.iamfas.belgium.be/fas/oauth2/userinfo

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

Table 2 FAS OIDC endpoints

Scopes and Claims

Given below is the list of supported scopes that can be requested during the authorization code request. These scopes will allow the Relying Party to retrieve claims.

Scope	Description
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[1] ▪ fedid ▪ PrefLanguage[1] (SMA profile is required) ▪ mail[1] (SMA profile is required)
egovnrn	This scope will only return the claim: ▪ NRN or BIS number (egovNRN) *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

[1] This field may not have a value and therefore not be returned by the system.

	▪ cert_serialnumber ▪ cert_cn ▪ cert_givenname ▪ cert_sn ▪ cert_mail
enterprise	States that the request is made in the name of an enterprise. **roles** and **enterprise** must be used together.
roles	This is an explicit request from roles of the authenticating end-user. **roles** and **enterprise** must be used together.
citizen	States that the end-user authenticates as a natural person. This scope is currently incompatible with the **enterprise** and **roles** scopes. This is the default scope if a relying party doesn't explicitly request the enterprise or citizen scope.

Table 3 List of available scopes

Note on citizen and enterprise scopes:

- **Roles** and **enterprise** scopes must be used together.
- The usage of **citizen** excludes **role** and **enterprise** and must be used alone.
- **Citizen** is the default scope if **enterprise** is not specified.

Role Attributes

The information returned after the role scope has been submitted is base64 encoded and has an XML structure.

The XML format uses a scheme to deliver the role information with a set attribute “role-name” and one or more possible parameters.

```
<rol:RoleResult xmlns:rol="http://be.fedict.rolemgmt/RoleXMLSchema">
  <rol:Role name="<application_role>"
xmlns:role="http://be.fedict.rolemgmt/RoleXMLSchema">
    <rol:RoleAttribute name="CompanyId">999999999</rol:RoleAttribute>
    <rol:RoleAttributename="FEDictDomain">SOMEDOMAIN</rol:RoleAttribute>
  </rol:Role>
</rol:RoleResult>
```

Authentication Context

The Authentication Context Class Reference (ACR) is mandatory so that the system allows you to login with eID, even when the FAS database would become unavailable.

The list of supported **`acr_values`** between a Relying Party 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:

Level of Assurance	**Authentication Means**	**Authentication Contract**[2]
High	eID[3]	`urn:be:fedict:iam:fas:Level500`
	eIDAS High	
	Itsme[3] High	`urn:be:fedict:iam:fas:Level450`
Substantial	eIDAS Substantial	`urn:be:fedict:iam:fas:Level400`
	Itsme[3] Substantial	
Substantial	Authenticator App	`urn:be:fedict:iam:fas:Level400`
	Mail OTP	
	SMS OTP[4]	

[2] 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.
[3] The digital keys, eID and itsme®, must be present at each onboarding.
[4] The digital key SMS OTP is disabled by default and should be requested to be used.

Low	Username/Password	`urn:be:fedict:iam:fas:Level200`
Weak	Self-registration without NRN	`urn:be:fedict:iam:fas:Level100`

Table 4 Allowed ACR with corresponding security level

Authorize Request

FAS uses the Authorization Code Grant Type to obtain an access token and allow applications to retrieve user data after the authentication. This chapter describes how to format the authorization code request and which data is returned from the authorization endpoint.

Reference:

https://openid.net/specs/openid-connect-core-1_0.html#AuthorizationEndpoint

Authorization Code Request

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:

Method	**Path**
GET	`/fas/oauth2/authorize`

Table 5 Authorization Code request

Parameter		**Value**
scope	**Mandatory**	**MUST** contain at least “openid”
response_type	**Mandatory**	<response type code>
nonce	Optional	A nonce will be used for KPI monitoring reasons
client_id	Optional	<client-id>
acr_values	Optional	Which LoA will be required by the client (see ACR Values)
redirect_uri	Optional	The onboarded https://redirect_uri scheme

state	Optional	<unique string value> Discretionary string value provided by the client, used to maintain state between the request and the callback
prompt	Optional	login
ui_locales	Optional but recommended	
response_node		*SHALL NOT be used*
display *Forbidden*		*SHALL NOT be used* *. Ignored if present.*
max_age *Forbidden*		*SHALL NOT be used* *. Ignored if present.*
id_token_hint *Forbidden*		*SHALL NOT be used* *. Ignored if present.*
claims *Forbidden*		*SHALL NOT be used* *. Ignored if present.*

Table 6 Authorization Code request parameters

Note:

- The Authorization code request and response **MUST NOT** be signed.
- It's **hightly recommended** to use the "state" parameter to avoid cross-site request forgery attacks. The state parameter is a hard to guess 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 usage is also **recommended**.

Sample Request

```
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 2 Authorization Code sample request

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`.

Parameter		Value
scope	**WILL** contain	The requested scopes
code[5]	**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

Table 7 Authorization Code response parameters

[5] The authorization code is by default only valid for 10 seconds after it has been issued.

Sample Response

```
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 3 Authorization Code sample response

Token Request

The obtained authorization code from the previous step 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.

Reference: https://openid.net/specs/openid-connect-core-1_0.html#TokenEndpoint

Access Token Request

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 supports `'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:

Method	Path
POST	`/fas/oauth2/access_token`

Table 8 Access Token request

Header		Value
Authorization	**MUST** contain	Basic base64(client id:client secret)
Content-Type	**MUST** contain	application/x-www-form-urlencoded

Table 9 Access Token request headers

Parameter		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

Table 10 Access Token request parameters

Note:

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

Sample Request

```
POST https://idp.iamfas.int.belgium.be/fas/oauth2/access_token
?grant_type=authorization_code
&code=HusR0KJVsNVHJ84myuMn5taiqi4
&redirect_uri=https://redirecturi.be

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

Figure 4 Access Token sample request

Access Token Response

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

- ID Token
- Access Token
- Refresh Token

Attributes		**Value**
scope	**WILL** contain	The original requested scopes
access_token	**WILL** contain	The requested access token
refresh_token	**WILL** contain	A refresh token that can be used to acquire a new access token
token_type	**WILL** contain	The type of the token (defaults to Bearer)

expires_in	**WILL** contain	A duration before the token is expired and no longer usable in seconds
id_token	**WILL** contain	An ID token, identifying the client

Table 11 Access Token response attributes

Sample Response


```
{
    "scope": "profile openid",
    "access_token": "SlAV32hkKG",
    "refresh_token": "absdgzegsfvsd",
    "token_type": "Bearer",
    "expires_in": 3600,
    "id_token":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8vaWRwLmlhbWZhcy5pbnQuYmVsZ2
l1bS5iZTo4MC9mYXMvb2F1dGgyIiwiaWF0IjoxNTc4MzkwNDQ4LCJleHAiOjE2MDk5MjY0NDgsImF1ZCI6In
JwX2NsaWVudF9pZF9uYW1lIiwic3ViIjoiNjQwODA3MzU5NzUiLCJhdWRpdFRyYWNraW5nSWQiOiI5YzMyZj
UzYS1iZTk0LTRjZmItOWZjNy1lYzFiMDFiZTA4MGEtMTQxNDU0MyIsInRva2VuTmFtZSI6ImlkX3Rva2VuIi
wiYW1yIjoidXJuOmJlOmZlZGljdDppYW06ZmFzOkxldmVsNTAwIiwibm9uY2UiOiIxMjQ0NTQ2IiwiY19oYX
NoIjoiN1hMa1R4cklqcmRCWUZiQUczaHg2ZyIsInNfaGFzaCI6ImJPaHRYOEY3M0lNalNQZVZBcXh5VFEiLC
JhenAiOiJycF9jbGllbnRfaWRfbmFtZSIsImF1dGhfdGltZSI6IjE1NzgzOTA0MzciLCJyZWFsbSI6Ii8iLC
J0b2tlblR5cGUiOiJKV1RUb2tlbiIsIm9yZy5mb3JnZXJvY2sub3BlbmlkY29ubmVjdC5vcHMiOiJrNkgydk
E1aUJUQ3hlWEphTUNIWUxTSVdqckIifQ.OnAtLjFh4vY2wvhUYDc6kinqyPDvfDkirVx2qAudVL4"
}
```


Figure 5 Access Token sample response

Sample IDToken

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 token:


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


Figure 6 Sample ID Token header

Content of the 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 7 Sample ID Token content

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.

Signature of the token:


```
TV9NNJH2FE2I_BWFQ0BGSKXASFJNVZRPTK88TJNV9FM
```


Figure 8 Sample ID Token signature

Introspect Request

The usage of introspect request is limited since the refresh token is not returned by default. This feature is used only by a small number of Relying Parties.

Please reach out to the Service Management of BOSA if you wish to use this functionality.

User Information Request

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.

Reference: https://openid.net/specs/openid-connect-core-1_0.html#UserInfo

UserInfo Request

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

Method	Path
GET	/fas/oauth2/userinfo

Table 12 User Info request

Header		Value
Authorization	MUST contain	Bearer 'Access-Token'

Table 13 User Info request headers

Sample Request

```
GET https://idp.iamfas.int.belgium.be/fas/oauth2/userinfo
headers: Authorization: Bearer qY2xTpn8xhAC6dRnsiVZ5zbbsnI
```

Figure 9 User Info sampe request

UserInfo 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.

Sample Response

```
EYJ0EXAIOIJKV1QILCJHBGCIOIJIUZI1NIJ9.EYJWCMVMTGFUZ3VHZ2UIOIJUBCISIM1HAWWIOIJZDXJUYW1
LLMDPDMVUBMFTZUBIB3NHLMZNB3YUYMUILCJZDXJUYW1LIJOIUGHPBCISIMDPDMVUTMFTZSI6IKNVDWXZB24
ILCJPC3MIOIJODHRWOI8VAWRWLMLHBWZHCY5PBNQUYMVSZ2L1BS5IZTO4MC9MYXMVB2F1DGGYIIWIZWDVDK5
STII6IJKXMDGYMJM1OTC1IIWIZMVKAWQIOIJSYZGZNJFMBDHIZGVMNGQZM2IXOGE1YWQ4YTAXOTGXMJDJN2U
1M2VLZCISIMLHDCI6MTU3ODM4NTU0MSWIZXHWIJOXNTC4MZG5NZM3LCJQDGKIOIJMZTC3MTK2MI02YTYXLTR
MM2ITYTQ1YS1JOGJMMDC5NJE0ZDYIFQ.DLR8IIPHK5OCN-JYJ8BXR8J0F2C0LN-KECUB7GLHJ9G
```

Figure 10 User Info sample response

Sample UserInfo

The claims requested for this example JWT are:

- `egovNRN`
- `profile`
- `roles`
- `enterprise`
- `certficateInfo`

Header of the token:

```
{
    "ALG": "RS256",
    "TYP": "JWT",
    "KID": "7OR5FBLAXRPXTXAFRQXN+O7GK9S="
}
```

Figure 11 User Info sample header

Body of the 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 12 User Info sample content

Signature of the token:


```
DLR8IIPHK5OCN-JYJ8BXR8J0F2C0LN-KECUB7GLHJ9G
```


Figure 13 User Info sample signature

JWT Token Validation

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

Token Content Validation

- 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.

Token Signature Validation

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.

Obtain the Public Key

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

Method	Path
GET	`/fas/oauth2/connect/jwk_uri`

Table 14 JWK request

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

Attribute	Description
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

Table 15 JWK response attributes

Sample Response:


```
{
    "keys": [
        {
            "kty": "RSA",
            "kid": "GDeFBZrcfqxZ2CIPBYYMgjRhwnY=",
            "use": "sig",
            "x5t": "tSrE6n0UOizCaNTrIA_KyWi4SSQ",
            "x5c":
["MIIIUDCCBjigAwIBAgIUIQ1JHx9W6UdzDvicCqdhuNk2BegwDQYJKoZIhvcNAQELBQAwUzELMAkGA1UEBh
MCQkUxIDAeBgNVBAoTF1F1b1ZhZGlzIFRydXN0bGluayBCVkJBMSIwIAYD<...>jYOoxsosjS0wYcxtBrfB2
1ZbkxkgVA="],
            "n": "urssM6BlGKux9zfTwMj-FLBJFmxik-
GIPr<...>7cj0vrpld1dONjq6GiqA_84ajrN7cGlC0hwQIic1TuqQ13wIn8wxGqAQ",
            "e": "AQAB",
            "alg": "RS256"
        },
        ...
    ]
}
```


Figure 14 JWK sample 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 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 15 Sample code to verify a JWT in Java

Validating the Signature

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

Example given in JAVA:

```
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);
```

Figure 16 Sample code to re-build a public key in Java

Logout Request

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

EndSession Request

Method	Path
GET	`/fas/oauth2/connect/endSession`

Table 16 Logout request

Parameter		Value
id_token_hint	**MUST** contain	The session ID_token
post_logout_redirect_uri	**SHALL** include	The https scheme

Table 17 Logout request parameters

Notes:

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

Sample Request

```
GET https://idp.iamfas.int.belgium.be/fas/oauth2/connect/endSession
?id_token_hint=EYJHBGCIOIJSUZI1NIISIMTPZCI6IJFLOWDKAZCIFQ.EWOGIMLZCYI6ICJODHRWOI8VC2
VYDMVYLMV4YW1WBGUUY29TIIWKICJZDWIIOIAIMJQ4MJG5NZYXMDAXIIWKICJHDWQIOIAICZZCAGRSA3F0MY
ISCIAIBM9UY2UIOIAIBI0WUZZFV3PBMK1QIIWKICJLEHAIOIAXMZEXMJGXOTCWLAOGIMLHDCI6IDEZMTEYOD
A5NZAKFQ.GGW8HZ1EUVLUXNUUIJKX_V8A_OMXZR0EHR9R6JGDQROOF4DAGU96SR_P6QNQEGPE-
GCCMG4VFKJKM8FCGVNZZUN4_KSP0AAP1TOJ1ZZWGJXQGBYKHIOTX7TPDQYHE5LCMIKPXFEIQILVQ0PC_E2DZ
L7EMOPWOAOZTF_M0_N0YZFC6G6EJBOEOROSDK5HODALRCVRYLSRQAZZKFLYUVCYIXEOV9GFNQC3_OSJZW2PA
ITHFUBEEBLUVVK4XUVRWOLRLL0NX7RKKU8NXNHQ-RVKMZQG
&post_logout_redirect_uri=https://www.google.com
```

Figure 17 Logout sample request