



FAS - OIDC Integration Guide

Table of Contents

Table of Contents	2
About this Document	4
FAS as an Authorization Server	5
OpenID Provider	6
Endpoints	6
Scopes and Claims	7
Scopes	7
Claims	8
Role Attributes	11
Authentication Context	11
Authorize Request	13
Authorization Code Request	13
Sample Request	15
Authorization Code Response	15
Sample Response	16
Token Request	17
Access Token Request	17
Sample Request	18
Access Token Response	18
Sample Response	19
Sample IDToken	19
Introspect Request	22
User Information Request	23
UserInfo Request	23
Sample Request	23
UserInfo Response	24
Sample Response	24
Sample UserInfo	24
JWT Token Validation	26
Token Content Validation	26
Token Signature Validation	26

Obtain the Public Key	26
Validating the Signature	29
Logout Request	31
EndSession Request	31
Sample Request	32

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.

Underneath a **"user"** and the more technical words **"OIDC client"** or **"client"** (OIDC terminology) are used to refer to a customer of FPS BOSA who makes available via OIDC a secured online public application that uses the FAS for the authentication of end-users. The **"end-user"** is indeed the person that makes use of this secured online public application of the above-mentioned user. See also the schema underneath. A user might consist of several Relying Parties. To avoid confusion with the different terminologies a "user" is therefore also called a **"Relying Party"**.

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>.

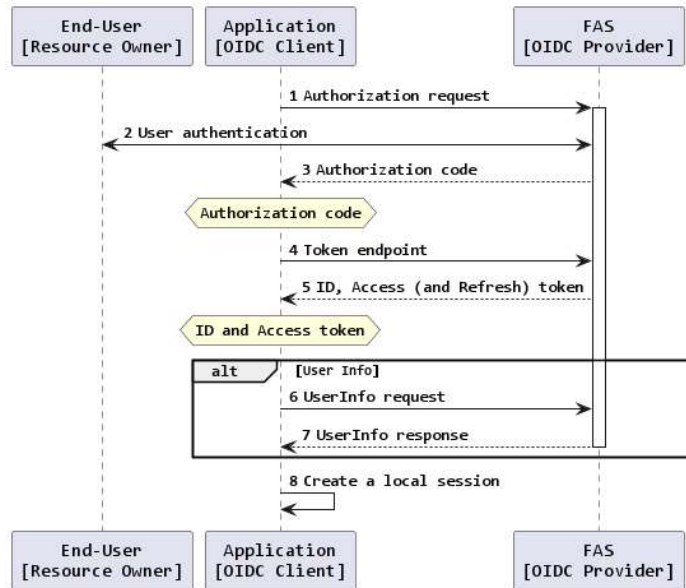


Figure 1 OIDC authorization request schema

FAS as an Authorization Server

1. The client application sends an *Authorization Code Request* towards the FAS Authorization server via the browser.
2. The end-user authenticates using one of the authentication methods available.
3. The 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 methods `client_secret_basic` or `client_secret_post` are used.
5. The FAS sends an Access token, Refresh token¹ 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. The 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 is not supported by FAS.

¹ Usage of Refresh Token is **disabled** by default and is only authorized for specific approved use cases

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

Scopes

Scope	Description
openid	This scope is a MUST if you want an <i>ID token</i> (specific scope in OAuth to upgrade your request to an OIDC request)
profile	This scope will return the following claims: <ul style="list-style-type: none"> ▪ surname ▪ givenName² ▪ fedid ▪ prefLanguage² (SMA profile is required) ▪ mail² (SMA profile is required)
egovnrn	This scope will only return the claim: <ul style="list-style-type: none"> ▪ NRN or BIS number (egovNRN)
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):

² This field may not have a value and might therefore not be returned by the system.

	<ul style="list-style-type: none"> ▪ cert_issuer ▪ cert_subject ▪ cert_serialnumber ▪ cert_cn ▪ cert_givenname ▪ cert_sn ▪ cert_mail
enterprise	<p>States that the request is made in the name of an enterprise.</p> <p>roles and enterprise must be used together.</p>
roles	<p>This is an explicit request from roles of the authenticating end-user.</p> <p>roles and enterprise must be used together.</p>
citizen	<p>States that the end-user authenticates as a natural person.</p> <p>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.</p>

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.

Claims

Claims	Endpoint	Description
acr (Type: string)	ID Token only	<p>Authentication Class Reference: OIDC standard claim containing the authentication level (LoA) used by the end user.</p> <p>IMPORTANT: Customer SHOULD validate the acr and/or amr claims in the FAS response according to their own policies (e.g. only eID and itsme is allowed as authentication method)</p>

amr: (type: Array of string)	ID Token only	Authentication Methods References: OIDC standard claim containing the list of authentication methods used by the end user. NOTE: Currently the FAS also includes the Level in the list of amr for legacy/backward compatibility reason. Usage of the Level from the amr claims is deprecated in favor of the acr claims IMPORTANT: Customer SHOULD validate the acr and/or amr claims in the FAS response according to their own policies (e.g. only eID and itsme is allowed as authentication method)
sub (type: string)	Both ID token and userinfo	Subject Identifier: A unique identifier for the end user's profile, typically the NRN of the profile has a valid NRN but could be set to another user id if the profile does not have a valid NRN (e.g. Foreigner). NOTE: Use the claim egovNRN if your application only expects valid NRN.
egovNRN (type: string)	Both ID token and userinfo	The NRN (or Bis Number) of the end user (if present in the user profile)
surname (type: string)	Both ID token and userinfo	The family name of the end user (if present in the user profile)
givenName (type: string)	Both ID token and userinfo	The first name of the end user (if present in the user profile)
mail (type: string)	Both ID token and userinfo	The SMA email of the end user (if present in the user profile)

prefLanguage (type: string)	Both ID token and userinfo	The SMA preferred language of the end user (if present in the user profile)
fedid (type: string)	Both ID token and userinfo	Opaque user identifier of the end user
cert_issuer (type: string)	Both ID token and userinfo	The Issuer of the eID certificate (if present for eID authentication) NOTE: legacy claim
cert_subject (type: string)	Both ID token and userinfo	The subject of the eID certificate (if present for eID authentication) NOTE: legacy claim
cert_serialnumber (type: string)	Both ID token and userinfo	The serial number of the eID certificate (if present for eID authentication) NOTE: legacy claim
cert_cn (type: string)	Both ID token and userinfo	The common name of the eID certificate (if present for eID authentication) NOTE: legacy claim
cert_givenname (type: string)	Both ID token and userinfo	The given name of the eID certificate (if present for eID authentication) NOTE: legacy claim
cert_surname (type: string)	Both ID token and userinfo	The surname of the eID certificate (if present for eID authentication) NOTE: legacy claim

roles (type: string)	Userinfo only	Extra attributes fetched from external sources defined in RMA
weak_profile_id (type: string)	Both ID token and userinfo	Unique identifier of the weak profile. NOTE: only present for low profile level 100 authentication method

NOTE: The ID Token and/or Userinfo MAY contain additional claims for internal use or beta features. Customers SHOULD NOT base their implementation on claims not explicitly documented above.

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 list of supported **acr_values** between a Relying Party and the FAS is given below:

- urn:be:fedict:iam:fas:Level1500
- urn:be:fedict:iam:fas:Level1450
- urn:be:fedict:iam:fas:Level1400
- urn:be:fedict:iam:fas:Level1300
- urn:be:fedict:iam:fas:Level1200
- urn:be:fedict:iam:fas:Level1100

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

Level of Assurance	Authentication Means	Authentication Contract ³
High	eID ⁴	urn:be:fedict:iam:fas:Level1500
	eIDAS High	
	Itsme ⁴ High	urn:be:fedict:iam:fas:Level1450
Substantial	eIDAS Substantial	urn:be:fedict:iam:fas:Level1400
	Itsme ⁴ Substantial	
Substantial	Authenticator App	urn:be:fedict:iam:fas:Level1400
	Mail OTP	
	SMS OTP ⁵	
Low	Username/Password	urn:be:fedict:iam:fas:Level1200
Weak	Self-registration without NRN	urn:be:fedict:iam:fas:Level1100

Table 4 Allowed ACR with corresponding security level

³ If the customer chooses a certain level, the keys with a higher technical level must (will) 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.

⁴ The digital keys, eID and itsme®, must be present at each onboarding.

⁵ The digital key SMS OTP is disabled by default and should be requested to be used.

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 executing 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 (but strongly recommended)	A nonce will be used for KPI monitoring reasons
client_id	Mandatory	<client-id>
acr_values	Mandatory	Which LOA should be used by the client (see ACR Values)

redirect_uri	Mandatory	The onboarded https://redirect_uri scheme NOTE: MUST match one of the provided values during the onboarding process
state	Optional (but strongly recommended)	<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	One of: “en” “de” “fr” “nl”
<i>response_node</i>	<i>Forbidden</i>	<i>SHALL NOT be used. Ignored if present.</i>
<i>display</i>	<i>Forbidden</i>	<i>SHALL NOT be used. Ignored if present.</i>
<i>max_age</i>	<i>Forbidden</i>	<i>SHALL NOT be used. Ignored if present.</i>
<i>id_token_hint</i>	<i>Forbidden</i>	<i>SHALL NOT be used. Ignored if present.</i>
<i>claims</i>	<i>Forbidden</i>	<i>SHALL NOT be used. Ignored if present.</i>

Table 6 Authorization Code request parameters

Note:

- The Authorization code request and response **MUST NOT** be signed.
- It is **highly 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:Level1500
&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
<code>scope</code>	MAY contain	The requested scopes
<code>code</code> ⁶	WILL contain	The authorization code
<code>state</code>	WILL contain	Must be the same value as in the authorization request
<code>client_id</code>	MAY contain	The client ID
<code>iss</code>	MAY contain	The issuer of the authorization code

Table 7 Authorization Code response parameters

⁶ 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).

The access token can be obtained by executing a HTTP **POST** request to the token endpoint of the FAS (.../fas/oauth2/access_token) with the following parameters using the "application/x-www-form-urlencoded" format with a character encoding of UTF-8 in the HTTP request entity-body (and not as 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 NOTE: may contain additional scopes used for internal use cases
access_token	WILL contain	The requested access token

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": "NN05qN0gD2s-aw-IkgQK0BG_uH8",
  "s_hash": "b0htX8F73IMjSPeVAqxyTQ",
  "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 **subject** 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 **issuer** of the tokens (the authorization server).
- The intended **audience** of the tokens (in this case the client id).
- An authorization and **expiry** 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 or plain JSON 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

User Info Request

Additional claims can be requested by executing 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 sample request

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.

Sample Response

```
EYJ0EXAI0IJKV1QILCJHBGCI0IJIUZI1NIJ9.EYJWCMVMTGFUZ3VHZ2UI0IJUBCISIM1HAWWIOIJZDXJUYW1LLMDPDMVUBMFTZUBIB3NHLMZNB3YUYMUJLCJZDXJUYW1LIJ0IUGHPBCISIMDPDMVUTMFTZSI6IKNVDWXZB24ILCJPC3MIOIJODHRW0I8VAWRWMLLHBWZHCY5PBNQUYMVSSZ2L1BS5IZT04MC9MYXMVB2F1DGGYIIWIZWDVVK5STII6IJKXMDGYMJM10TC1IIWIZMVKAQIOIJSYZGZNFMBDHI ZGVMNGQZM2IXOGE1YWQ4YTAX0TGXMJDJN2U1M2VLZCISIMLHDCI6MTU3ODM4NTU0MSWIZXHWIJOXNTC4MZG5NZM3LCJQDGKIOIJMZTC3MTK2MI02YTYXLTRMM2ITYTQ1YS1J0GJMMDC5NJE0ZDYIFQ.DLR8IIPHK5OCN-JYJ8BXR8J0F2C0LN-KECUB7GLHJ9G
```

Figure 10 User Info sample response

Sample UserInfo

The claims requested for this example JWT are:

- egovNRN
- profile
- roles
- enterprise
- certificateInfo

Header of the token:

```
{  
  "ALG": "RS256",  
  "TYP": "JWT",  
  "KID": "70R5FBLAXRPXTXAFRQXN+07GK9S="
```


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": "lc8361f18bdef4d33b18a5ad8a0198127c7e53eed",
  "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 10 User Info sample content

Signature of the token:

```
DLR8IIPHK50CN-JYJ8BXR8J0F2C0LN-KECUB7GLHJ9G
```

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	<code>/fas/oauth2/connect/jwk_uri</code>

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(signature))
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": "GDeFBZrcfqxZ2CIPBYMgjRhwnY=",
      "use": "sig",
      "x5t": "tSrE6n0U0izCaNTrIA_KyWi4SSQ",
      "x5c":
        ["MIIIUCCBjigAwIBAgIU1Q1JHx9W6UdzDvicCqdhUNk2BegwDQYJKoZIhvcNAQELBQAwUzELMAkGA1UEBhMCQkUxIDAeBgNVBAoTF1F1b1ZhZGlzIFRydXN0bGluyBCVkJBMSIwIAAYD<...>jY0oxsosjS0wYcxtBrfB21ZbkxkgVA="],
      "n": "urssM6BlGKux9zfTwmj-FLBJFmxik-GIPr<...>7cj0vrpld1dONjq6GiqA_84ajrN7cGlC0hwQIic1TuqQ13wIn8wxGqAQ",
      "e": "AQAB",
      "alg": "RS256"
    },
    ...
  ]
}
```

Figure 11 JWK sample response

With this information, the public key can be reconstructed with the modulus and the exponent. Since the `jwt` 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 12 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 13 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	<code>/fas/oauth2/connect/endSession</code>

Table 16 Logout request

Parameter		Value
<code>id_token_hint</code>	MUST contain	The session ID_token
<code>post_logout_redirect_uri</code>	MAY include	The https scheme

Table 17 Logout request parameters

Notes:

- EndSession calls can be made either via a browser redirect (i.e. using the front channel) or via a backend call (i.e. via the back channel) in which case the `post_logout_redirect_uri` is not necessary. It is recommended to use the Back Channel (backend) call to avoid leaking the ID Token via the front channel and to be able to gracefully handle the endsession response (error handling).
- 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=EYJHBGCI0IJSUZI1NIISIMTPZCI6IJFLOWDKAZCIFQ.EWOGIMLZCYI6ICJODHRWOI8VC2
VYDMVYLMV4YW1WBGUUY29TIIWKICJZDWIIOIAIMJQ4MJG5NZYXMDAXIIWKICJHDWQIOIAICZZCAGRSA3F0MY
ISCIAIBM9UY2UIOIAIBI0WUZZFV3PBMK1QIIWKICJLEHAI0IAXMZEXMJGX0TCWLAOGIMLHDCI6IDEZMTEYOD
A5NZAKFQ.GGW8HZ1EUVLUXNUUIJKX_V8A_OMXZR0EHR9R6JGDQR0OF4DAGU96SR_P6QNQEGPE-
GCCMG4VFKJKM8FCGVNZZUN4_KSP0AAP1T0J1ZZWGJXQGBYKHI0TX7TPDQYHE5LCMIKPXFEIQILVQ0PC_E2DZ
L7EMOPW0AOZTF_M0_N0YZFC6G6EJB0EOROSDK5HODALRCVRYLSRQAZZKFLYUVCYIXE0V9GFNQC3_OSJZW2PA
ITHFUBEEBLUVVK4XUVRWOLRLL0NX7RKKU8NXNHQ-RVKMZQG
&post_logout_redirect_uri=https://www.google.com
```

Figure 14 Logout sample request