DG Digitale Transformatie
**FOD Beleid en Ondersteuning**
DG Transformation digitale
**SPF Stratégie et Appui**

# FAS
# Authorization Server
# -
# OpenID Connect
# Onboarding

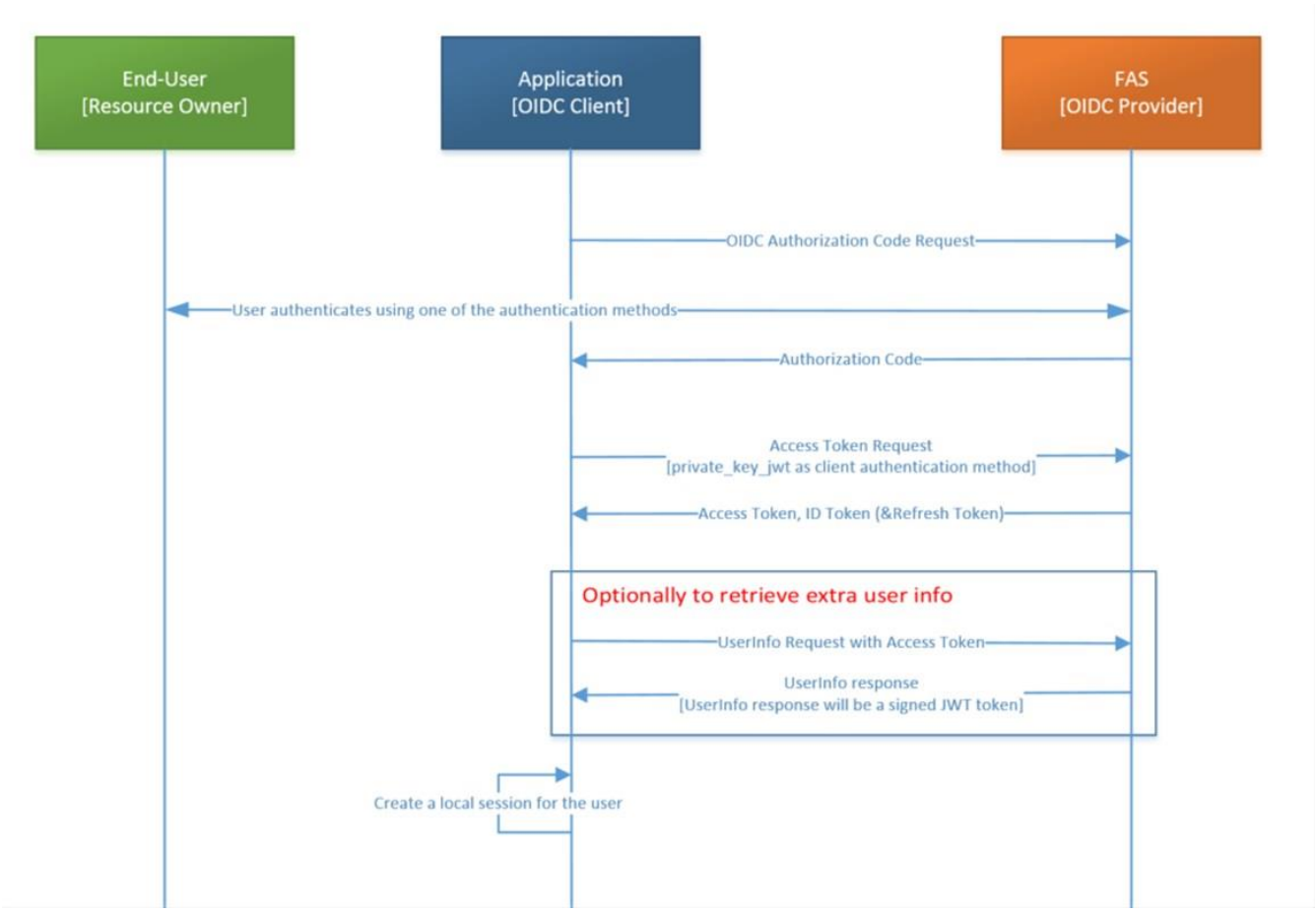
**Siège social / Maatschappelijke zetel**
WTCIII • Boulevard Simon Bolivarlaan 30
1000 Bruxelles / Brussel

Numéro d'entreprise / Zakelijke nummer
671.516.647

T +32 (0)2 740 74 74
**www.bosa.be**

# Table of Content

## FAS as an authorization server




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 private_key_jwt.
5. FAS sends an Access token (, Refresh token) and ID token via server to server communication to the client application.
6. The client application optionally calls the userinfo endpoint of the FAS using the Access Token.
7. FAS sends a signed JWT with extra user information to the client application.
8. The client application is now able to create a local session for the user.

# 1 OpenID Connect Authorization Code Request and Response

## 1.1 OPENID CONNECT AUTHORIZATION CODE REQUEST

The Authorization code request and response MUST NOT be signed.

However 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 3.5 of RFC6819.

The nonce parameter SHALL be used.

**GET towards the Authorization Code endpoint of the FAS (*.../fas/oauth2/authorize)*:**

| **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** | SHALL NOT be used | |
| **max_age** | SHALL NOT be used | |
| **ui_locales** | SHALL NOT be used | |
| **id_token_hint** | SHALL NOT be used | |
| **acr_values** | SHALL be used | Which LoA will be required by the client |
| **claims** | SHALL NOT be used | |

### 1.1.1 acr_values

**Possible acr_values between a RP and the FAS:**

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

### 1.1.2 Scopes and Claims

- **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 contain the following claims:
    - egovNRN
    - surname
    - givenName
    - fedid
    - PrefLanguage
    - mail
- **egovNRN**
  - This scope will only contain the RRN/NRN or BIS number claim of the authenticated user.
- **certficateInfo**
  - If the user authenticates using eID and the scope certificateInfo is requested we'll return the following claims (if present in the eID certificate):
    - cert_issuer
    - cert_subject
    - cert_serialnumber
    - cert_cn
    - cert_givenname
    - cert_sn
    - cert_mail

*Example request:*

```
GET TO HTTPS://…/FAS/OAUTH2/AUTHORIZE
?RESPONSE_TYPE=CODE
&CLIENT_ID=MYCLIENTID
&SCOPE=OPENID%20PROFILE
&ACR_VALUES=URN:BE:FEDICT:IAM:FAS:STAR:LEVEL500
&REDIRECT_URI=HTTPS://FEDICT.BE
&STATE=AF0IFJSLDKJ
&NONCE=1244542
```

## 1.2 OPENID CONNECT AUTHORIZATION CODE RESPONSE

**GET to the redirect_uri of the authorization code request with the following parameters:**

| **Parameter** | | |
|---|---|---|
| **scope** | SHOULD contain | The requested scopes |
| **code** | MUST contain | The authorization code |
| **state** | MUST contain | Must be the same value as in the authorization request |
| **client_id** | MUST contain | The client ID |
| **iss** | MUST contain | The issuer of the authorization code |

*Example response:*

```
REDIRECT_URI
?CODE=31308323-C08E-431A-A5DC-2E7335795B43
&SCOPE=OPENID%20PROFILE
&ISS=HTTP%3A%2F%2FIDP-POC.IAMFAS.QA.BELGIUM.BE%3A80%2FFAS%2FOAUTH2
&STATE=AF0IFJSLDKJ
&CLIENT_ID=OPENIDDEMOPOC
```

# 2 OpenID Connect Access Token request and Response

## 2.1 OPENID CONNECT 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):

- Client_secret_basic
- Client_secret_post
- Client_secret_jwt
- Private_key_jwt

FAS will only support **private_key_jwt** as client authentication method, it's the most secure client authentication method.

**POST to the token endpoint of the FAS (*.../fas/oauth2/access_token)*:**

| **Parameters** | | |
|---|---|---|
| **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 in the authorization code request |
| **client_assertion_type** | MUST contain | *urn:ietf:params:oauth:client-assertion-type:jwt-bearer* |
| **client_assertion** | MUST contain | *A signed JWT token* |

*Example:*

```
URL:    HTTPS://.../ FAS/OAUTH2/ACCESS_TOKEN
TYPE:   POST
DATA:   {
            "GRANT_TYPE": "AUTHORIZATION_CODE",
            "CODE": CODE,
            "CLIENT_ASSERTION_TYPE": "URN:IETF:PARAMS:OAUTH:CLIENT-ASSERTION-TYPE:JWT-BEARER",
            "CLIENT_ASSERTION": JWTTOKEN
        }
```

Where jwtToken :

eyJraWQiOiJkYzY4YmNhMC1kMzc5LTQxMjMtYmJhYi01ZjU0Y2IyNmZlYTciLCJhbGciOiJSUzI1NiJ9.eyJqdGkiOiI2ZWM5N2ZhZS01ZmYyLTQyYWItOTk0MS1hNGI0ZDEyMDgwNzEiLCJpYXQiOjE1MTA2NTE2MDUsInN1YiI6Ik9wZW5JRFByaXZhdGVLZXlKV1QiLCJpc3MiOiJPcGVuSURQcml2YXRlS2V5SldUIiwiYXVkIjoiaHR0cDovL2lkcC1wb2MuaWFtZmFzLnFhLmJlbGdpdW0uYmU6ODAvZmFzL29hdXRoMi9hY2Nlc3NfdG9rZW4iLCJleHAiOjE1MTA2NTUyMDV9.FFB81t7XHH3siqquBksbLOQl0__pWO5pGK35s_3deW9ZswylxwDklrvWPYpUzkk9WnSYjG1E_54UdmZ6x5jkWaNmjRTkEdDLug0ehsq3gXWjXOFyr7Lnmk3f8dXIGpD_4KFu-LkctZU4FdiUeIaYdleeIcFCf9eg9tgfGHIXxZvVktCj3V59g1ek2o_L44z1j6azQGxbVZ34i7CD29lNSn05KTO6aTbrG5R4fqAfQpTHMGf8wGNfYxApK8t_CTScPQhYnqFVrDnPQj_g5jKYoPWX-u7D75yxfTSRnAu1fDD1zuDOLdTuZ6dS-IrBOM4FdpWtzqS3wG7nPV0teAEFCw

```
{
        "kid": "dc68bca0-d379-4123-bbab-5f54cb26fea7",
        "alg": "RS256"
}.
{
        "jti": "6ec97fae-5ff2-42ab-9941-a4b4d1208071",
        "iat": 1510651605,
        "sub": "clientID",
        "iss": "clientID",
        "aud": "http://idp-poc.iamfas.qa.belgium.be:80/fas/oauth2/access_token",
        "exp": 1510655205
}.
{
        Signature not shown
}
```


**Note**: Notice the difference between the requested URL and the URL in the audience field of the JWT token. This is the bug by OpenAM we introduced earlier.
For more information about the private_key_jwt client authentication mechanism and how to create the JWK URI see …

## 2.2 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 an ID Token an Access Token and a Refresh Token. The response uses the `application/json` media type.

*Example response:*


```
{
  "SCOPE":"PROFILE OPENID",
  "ACCESS_TOKEN": "SLAV32HKKG",
 “REFRESH_TOKEN”: “ABSDGZEGSFVSD”,
  "TOKEN_TYPE": "BEARER",
  "EXPIRES_IN": 3600,
  "ID_TOKEN": "EYJHBGCIOIJSUZI1NIISIMTPZCI6IJFLOWDKAZCIFQ.EWOGIMLZC
   YI6ICJODHRWOI8VC2VYDMVYLMV4YW1WBGUUY29TIIWKICJZDWIIOIAIMJQ4MJG5
   NZYXMDAXIIWKICJHDWQIOIAICZZCAGRSA3F0MYISCIAIBM9UY2UIOIAIBI0WUZZ
   FV3PBMK1QIIWKICJLEHAIOIAXMZEXMJGXOTCWLAOGIMLHDCI6IDEZMTEYODA5NZ
   AKFQ.GGW8HZ1EUVLUXNUUIJKX_V8A_OMXZR0EHR9R6JGDQROOF4DAGU96SR_P6Q
   NQEGPE-GCCMG4VFKJKM8FCGVNZZUN4_KSP0AAP1TOJ1ZZWGJXQGBYKHIOTX7TPD
   QYHE5LCMIKPXFEIQILVQ0PC_E2DZL7EMOPWOAOZTF_M0_N0YZFC6G6EJBOEORO SD
   K5HODALRCVRYLSRQAZZKFLYUVCYIXEOV9GFNQC3_OSJZW2PAITHFUBEEBLUVVK4
   XUVRWOLRLL0NX7RKKU8NXNHQ-RVKMZQG"
 }
```


*Example decoded id_token:*

- Decoded id_token header:


```
{
 "TYP": "JWT",
 "ALG": "HS256"
}
```


- Decoded id_token token data:


```
{
  "AT_HASH": "J3_BKFYWTWGHMKKD6UT10W",
  "SUB": "91111325384",
  "AUDITTRACKINGID": "201109D4-437C-4C77-9886-2FA130B1EBAB-7490",
  "ISS": "HTTP://IDP-POC.IAMFAS.QA.BELGIUM.BE:80/FAS/OAUTH2",
  "TOKENNAME": "ID_TOKEN",
  "AUD": "CLIENTID",
  "C_HASH": "WSKELV-N8ZCGTT2RG66KMQ",
  "ACR": "URN:BE:FEDICT:IAM:FAS:CITIZEN:LEVEL500",
  "ORG.FORGEROCK.OPENIDCONNECT.OPS": "CB3BA15E-0A75-4A66-8F1F-   F2E4B50807A9",
  "AZP": "CLIENTID",
  "AUTH_TIME": 1482230162,
  "REALM": "/",
  "EXP": 1482233763,
  "TOKENTYPE": "JWTTOKEN",
  "IAT": 1482230163
}
```


- id_token signature:


```
TV9NNJH2FE2I_BWFQ0BGSKXASFJNVZRPTK88TJNV9FM
```

# 3 OpenID Connect User Info request and Response

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

## 3.1 OPENID CONNECT USER INFO REQUEST

The user info request is POST request towards the FAS userinfo endpoint with one extra header which will contain the access token.

**POST to the userinfo endpoint of the FAS (*.../fas/oauth2/userinfo)*:**

| **Parameters** | | |
|---|---|---|
| **Authorization** | MUST contain | Bearer 'Access-Token' |

*Example:*

```
https://.../fas/oauth2/userinfo?access_token=ea9ef5f4-c96e-48b6-8629-618e62e51c52
```

## 3.2 OPENID CONNECT USER INFO RESPONSE

If the access token is still valid, the FAS will return a signed JWT token.

*Example response:*

- Decoded user info JWT header:

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

- Decoded user info JWT data:

```
{
 "eGovNRN": "91211325384",
 "cert_issuer": "SERIALNUMBER=201504, CN=Citizen CA, C=BE",
 "cert_subject":
"SERIALNUMBER=91121325384,GIVENNAME=John,SURNAME=snow,CN=John
snow (Authentication),C=BE",
 "cert_serialnumber": "9121325384",
 "cert_cn": "John Snow (Authentication)",
 "cert_givenname": "John",
 "cert_sn": "snow",
 "cert_mail": null
}
```

- User info JWT signature:

```
JpU5lGucLxP/RHH0MQFWmSy/Y8UbfYXQ8I7ey9eXvDmDXoujeRlmJl+aa7sqZPGW6MwfqZSDNCLGn7
Oks92bRX5mn1ODxMCiKkuPmZwRHWVeecq23hTFffrf1eaSH3Tc6I4ak+oIOsZLCeLsBsV9km3RToISn9+Y
RrCuaZUmKKTwhIjh8FlTA1JVFtfO+LDdDvUqcrrTmHs6mN/b9Gbt0fPlhkSvSKRCElgKHAmyo6zWc8Hds
```

# 4 Private_key_jwt client authentication mechanism

## 4.1 JWK URI

The jwk uri provides a json object that will be used by the FAS to validate the private key JWT token. The json object needs at least the following fields:

- kty: the type of key that is represented in the token
  - Value should be: “RSA”
- use: the operation for which the key will be use
  - Value should be: “sig” (signature check)
- kid: the id of the key
  - Value should be unique
  - Value will be used during the creating of the private_key_jwt token
- n: the modulus part of the public key.
  - The public key should be part of a key pair of which the private part will be used to sign the private_key_jwt token
  - Value should be base64 encoded
- e: the exponent part of the public key
  - Value will often be: “AQAB” (base64 encoding of 65537)

Remark: The jwk_uri can contain multiple public keys. The json object should then consist of a single *keys* field. This field can contain an array of any number of json object as described above. Every object in the array should have a unique *kid*.
*Example:*

## 5.2 PRIVATE_KEY_JWT_TOKEN

The private_key_jwt_token will be used as client authentication mechanism. The jwt token consists of a header, a payload and a signature.

### 5.2.1 Header

The header of the jwt token should be the base64 encoding of a json object containing at least the following fields:

- kid: The id of the key that will be used to sign the token
  - The value should be the *kid* value of the public key from the jwk_uri whose private key counterpart is used to sign the token
- alg: The algorithm that is used to sign the token
  - Value should be: “RS256” (RSA)

### 5.2.2 Payload

The payload of the jwt token should be the base64 encoding of a json object containing at least the following fields:

- jti: A random number that identifies the jwt token
- sub: The subject of the token (i.e. for who this token was issued)
  - Value is the same as *iss*
  - Value is name of client (client_id)
- iss: The issuer of the token (i.e. Who created the token)
  - Value is the same as *sub*
- aud: The targeted audience of the token
  - Value is the url of the token endpoint of the openam instance
  - A bug exists in the current version of openam where the audience is different than expected (see the examples in section 2)
- exp: The expiration date of the token
  - Value is in milliseconds

### 5.2.3 Signature

The signature consists of the signature of the token. It is created with a private key:

```
SIGN(BASE64URLENCODE(HEADER) + "." + BASE64URLENCODE(PAYLOAD),PRIVATE_KEY)
```

*Example:*

**A private_key_jwt token and its decoded values (using jwt.io):**


eyJraWQiOiJkYzY4YmNhMC1kMzc5LTQxMjMtYmJhYi01ZjU0Y2IyNmZlYTciLCJhbGciOiJSUzI1NiJ9.eyJqdGkiOiI3ZWY4ZjQ2NC1kODE0LTRkNWUtOGE3Yy1kMjY3NzE5MGE5ZjciLCJpYXQiOjE1MTA2NDk5MzcsInN1YiI6Ik9wZW5JRFByaXZhdGVLZXlKV1QiLCJpc3MiOiJPcGVuSURQcml2YXRlS2V5SldUIiwiYXVkIjoiaHR0cDovL29wZW5hbTEuZmVkaWN0bGFiLmJlOjgwL2Zhcy9vYXV0aDIvYWNjZXNzX3Rva2VuIiwiZXhwIjoxNTEwNjUzNTM3fQ.Sn_dqWBD0ZgYfTDlPIJFXtd8R79kB-FXql9Q3mrsUDhuRJqaelQzD0JbviEP3H3-CaCbzjmgf2fPWYAGdRHbv1ja684pYj79NxyNqZRje-fKD5mXXEzbleMxHFIC9K-sakSse20Z2kQgbocBFIfiy8xS-cUsPO2Gq8EVm2KhE6YfjFlYHTydPol-XUdKPjb4Y6L_GYzwL10bft_Tb7I88ccfMTK4OZA78QreOWoFD0J6lGn72iCXSF9-dEDZd6M8D7zgJCorp9cjPMelkpxQgOGL1hGI8tDmiUXkbEcg82RQ2M4jS5xBURjdX7wLH