Logius Standard - Definitive version

# OpenID NLGov 1.0.1

## Logius Standard
## Definitive version September 18, 2023

**This version:**
https://gitdocumentatie.logius.nl/publicatie/api/oidc/1.0.1

**Latest published version:**
https://gitdocumentatie.logius.nl/publicatie/api/oidc

**Latest editor's draft:**
https://logius-standaarden.github.io/OIDC-NLGOV/

**Editors:**
Remco Schaar (Logius)
Frank van Es (Logius)
Pieter Hering (Logius)
Martin van der Plas (Logius)
Alexander Green (Logius)

**Authors:**
Remco Schaar (Logius)
Frank van Es (Logius)
Joris Joosten (VZVZ)
Jan Geert Koops (Dictu)

**Participate:**
GitHub Logius-standaarden/OIDC-NLGOV
File an issue
Commit history
Pull requests

This document is also available in these non-normative format: pdf



## Abstract

The OpenID Connect protocol defines an identity federation system that allows a Relying Party to request and receive authentication and profile information about an End-User.

This specification profiles the OpenID Connect protocol to increase baseline security, provide greater interoperability, and structure deployments in a manner specifically applicable to (but not limited to) government and public service domains in The Netherlands.

This profile builds on top of, and inherits all properties of, the NL GOV Assurance profile for OAuth 2.0 [OAuth2.NLGov].

## Status of This Document

This is the definitive version of this document. Edits resulting from consultations have been applied.

## Table of Contents

# 1. Introduction

Government regulations for permitting users (citizens and non-citizens) online access to government resources vary greatly from country to country. There is a strong desire to leverage federated authentication and identity services for public access to government resources online to enable the development of safe and innovative applications for e-government services, increase overall account security, reduce cost, and provide reliable identity assurances from established and trusted sources when applicable.

OpenID Connect is a protocol enabling such federated identity and authentication protocol. OpenID Connect supports a variety of Use Cases and offers a range of features and (security) options. This specification aims to define an OpenID Connect profile that provides Dutch governments with a foundation for securing federated access to public services online when applying OpenID Connect.

## 1.1 Requirements Notation and Conventions

The key words "*MUST*", "*MUST NOT*", "*REQUIRED*", "*SHALL*", "*SHALL NOT*", "*SHOULD*", "*SHOULD NOT*", "*RECOMMENDED*", "*NOT RECOMMENDED*", "*MAY*", and "*OPTIONAL*" in this document are to be interpreted as described in [RFC2119].

All uses of "JSON Web Signature (JWS)" [RFC7515] and "JSON Web Encryption (JWE)" [RFC7516] data structures in this specification utilize the JWS Compact Serialization or the JWE Compact Serialization; the JWS JSON Serialization and the JWE JSON Serialization are not used.

## § 1.2 Terminology

This specification uses the following terms:

- "Access Token", "Authorization Code", "Authorization Endpoint", "Authorization Grant", "Authorization Server", "Client", "Client Authentication", "Client Identifier", "Client Secret", "Grant Type", "Protected Resource", "Redirection URI", "Refresh Token", "Resource Server", "Response Type", and "Token Endpoint" defined by 'OAuth 2.0' [RFC6749];
- "Claim Name", "Claim Value", and "JSON Web Token (JWT)" defined by 'JSON Web Token (JWT)' [RFC7519];
- "Introspection Endpoint" defined by [RFC7662];
- "Revocation Endpoint" defined by [RFC7009];
- "Browser-based application" defined by [OAuth2.Browser-Based-Apps];
- "Native app", "Hybrid app", "External user-agent", "Embedded user-agent", "In-app browser tab", "Web-view", "Claimed 'https' scheme URI", "Private-use URI scheme" defined by 'OAuth 2.0 for Native Apps' [RFC8252];
- "User-agent" defined by 'Hypertext Transfer Protocol' [RFC2616]; and
- the terms defined by 'OpenID Connect Core 1.0' [OpenID.Core].

In addition to the above terminology, this profile defines the following terms:

- "Representation", "Representation Relationship", "eIDAS".

Definitions for these terms as well as for the abbreviations used throughout this specification are listed in the Glossary.

## § 1.3 Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

This profile is based upon the 'International Government Assurance Profile (iGov) for OpenID Connect 1.0' [OpenID.iGov] as published by the OpenID Foundation. It should be considered a

fork of this profile, as the iGov profile is geared more towards a United States context and this NL GOV profile towards a Dutch context with European Union regulations applicable.

This specification defines requirements for the following components:

- OpenID Connect 1.0 Relying Parties (also known as OpenID Clients, or RP)
- OpenID Connect 1.0 Identity Providers (also known as OpenID Providers, IdP or OP)

The specification also defines features for interaction between these components:

- Relying Party to Identity Provider

When an NL GOV-compliant component is interacting with other NL GOV-compliant components, in any valid combination, all components *MUST* fully conform to the features and requirements of this specification. All interaction with non-NL GOV components is outside the scope of this specification.

An NL GOV-compliant OpenID Connect Identity Provider *MUST* support all features as described in this specification. A general-purpose Identity Provider *MAY* support additional features for use with non-NL GOV Clients.

An NL GOV-compliant OpenID Connect Identity Provider *MAY* also provide NL GOV-compliant OAuth 2.0 Authorization Server functionality. In such cases, the Authorization Server *MUST* fully implement the NL GOV Assurance profile for OAuth 2.0 [OAuth2.NLGov]. If an NL GOV-compliant OpenID Connect Identity Provider does not provide NL GOV-compliant OAuth 2.0 Authorization Server services, all features related to interaction between the Authorization Server and protected resource are *OPTIONAL*.

An NL GOV-compliant OpenID Connect Client *MUST* support all required functionality described in this specification. A general-purpose Client library *MAY* support additional features for use with non-NL GOV OpenID Connect Identity Providers.

> Note that the original concept of the [OpenID.NLGov] profile was published on logius.gitlab.io as version 1.0 in February 2021 with the title "NL GOV Assurance profile for OpenID Connect 1.0".

## § 2. Use Case & context

This profile supports several Use Cases or partial aspects thereof. Design choices within this profile have been made with these Use Cases under consideration.

The generic Use Case is an End-User with the intention to consume an online service of a Service Provider. As the service requires authentication, this triggers the authentication process.

Authentication is provided in a federated manner. In other words, a Client system is relying upon another system, the OpenID Provider, for authentication. Either a shared central OpenID Provider or a (distributed) network of OpenID Providers, a.k.a. a federation or scheme is being used. The ecosystem supported by the OpenID Provider can either be a single organization (intra-organizational) or multiple organizations (inter-organizational), through either bilateral or multilateral agreements. In case a federation or scheme is being used, an Identity Broker may be applicable. Although this profile allows for usage in a federation, no explicit support for federations is *currently* included.

The service is offered by a (semi-)governmental or public Service Provider. The Use Cases therefore explicitly covers Citizen-to-Government as well as Business-to-Government contexts. Note that business-to-government is not strictly limited to businesses, these may be other governmental organisations (inter-organizational) or internal service consumers (intra-organisational). This profile is not limited to these contexts, nor intended to exclude Business-to-Consumer and Business-to-Business contexts, but additional considerations may be applicable in those contexts.

The Service Provider or OpenID Client requests either an identifier, attributes or both of an authenticated End-User from the OpenID Provider. As target End-User audiences are diverse, multiple types of identifiers can be supported. Supported Use Cases therefore span both identifiale and attribute-based authentication.

From an architectual standpoint, the Use Case can utilize a Client in the form of a hosted web-application, a mobile/native application or a browser based single-page-application (SPA). See Section 4.1 Client Types for more details.

## § 2.1 Representation

This profile supports several Use Cases for Representation Relationships, which apply when an End-User intends to consume an online service on behalf of a Natural or Juridical Person (the service consumer), where authentication and authorization is required. The End-User in these Use Cases is a Natural Person, representing the service consumer through a Representation Relationship. The relationship has to be formalized and may be either a direct relationship, either voluntarily or on legal grounds, or a chain of Representation Relationships. The formalization of these relationships is out of scope of this profile.

Example Representation Use Cases include voluntary authorization, representative assigned by court order (guardian, administrator), statutory signatory (director, president), limited authorized

signatory, etc.

## § 2.2 Misc

The OpenID Connect specification [OpenID.Core] supports self-issued OpenID Connect Providers. However, as this profile centers around (semi-)governmental and public domain Use Cases where assurance on identity verification is virtually always required, self-issued OpenID Providers *MUST NOT* be accepted by OpenID Clients under this profile.

As the Dutch identity eco-system supports multiple OpenID Providers, Identity Brokers are in common use. Brokers relieve OpenID Clients of managing multiple connections to OpenID Providers, but every additional step introduces security risks and concern with regards to privacy. Among the privacy concerns is the forming of so-called privacy hotspots, points were data collection can be concentrated. To mitigate such risks, end-to-end security is considered throughout this profile. Controls such as signing, to assure integrity, and encryption, to strengthen confidentiality, are encouraged to increase overall end-to-end security.

> Note that future versions of this profile may support use cases where Service Intermediation is applicable.

# § 3. Flow

OpenID Connect Core specifies three paths via which authentication can be performed: the *Authorization Code Flow*, the *Implicit Flow* and the *Hybrid Flow*. The flows determine how the ID Token and Access Token are returned to the Client.

This profile requires that authentication is performed using the Authorization Code Flow, in where all tokens are returned from the Token Endpoint.

The Implicit Flow and Hybrid Flow allow tokens to be obtained from the Authorization Endpoint, and thereby omitting the Token endpoint. This makes them vulnerable to token leakage and token replay and makes it impossible to cryptographically bind tokens to a certain Client.

Therefore, the Implicit Flow and Hybrid flow *MUST NOT* be used. Also, the IETF OAuth Working Group is removing support for the Implicit Flow from the OAuth 2.1 specification [OAuth2.1] for the same reasons.

## § 3.1 Authorization Code Flow

The Authorization Code Flow returns an Authorization Code to the Client, which can then exchange it for an ID Token and an Access Token directly. The flow comprises the following steps:

1. The Client sends an Authorization Request - containing the desired request parameters - to the OpenID Provider.
2. The OpenID Provider authenticates the End-User.
3. The OpenID Provider sends the End-User back to the Client with an Authorization Code.
4. The Client requests a response using the Authorization Code at the Token Endpoint.
5. The Client receives a response that contains an ID Token and Access Token in the response body.
6. The Client validates the ID token and retrieves Claims and Subject Identifier(s) of the authenticated End-User.

The flow described by these steps is illustrated as follows:

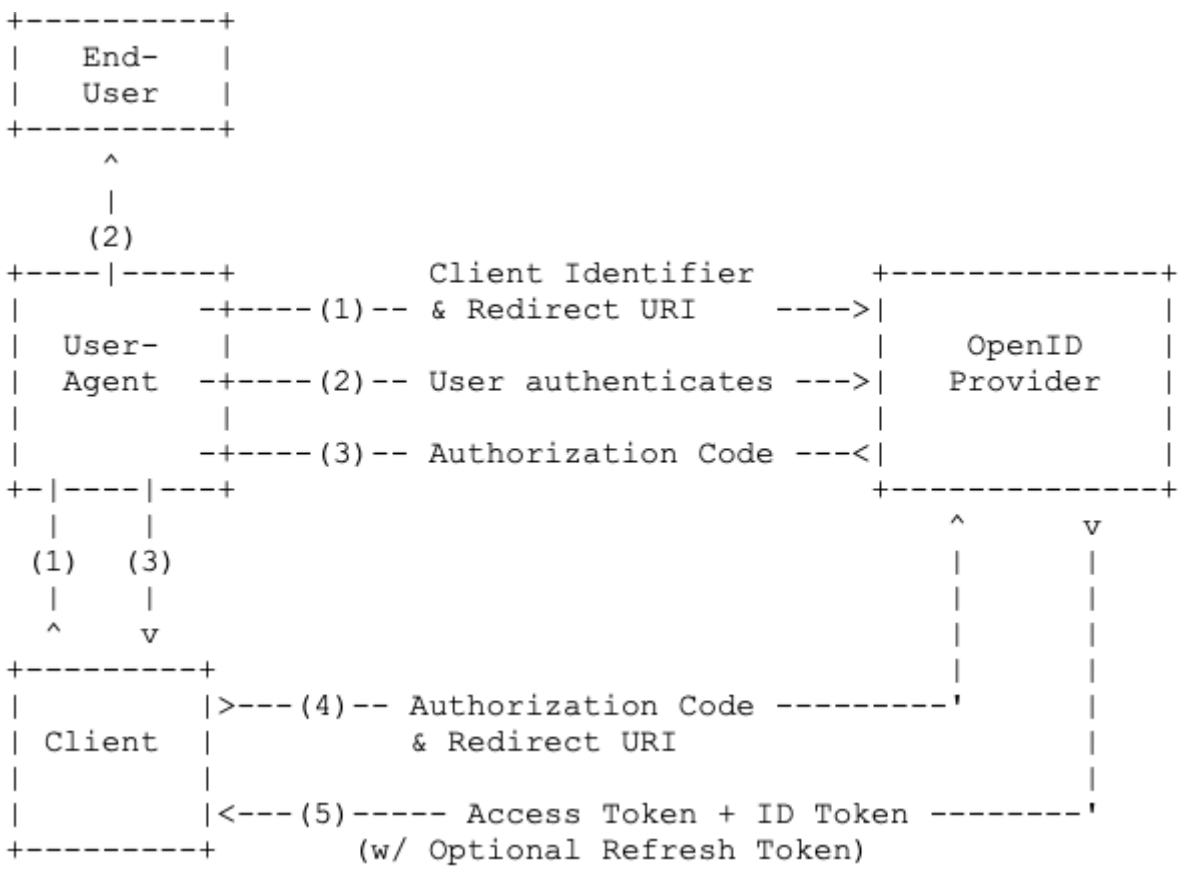


*Figure 1 Authorization Code Flow*

# § 4. OpenID Client profile

## § 4.1 Client Types

OAuth 2.0 defines two Client Types (*confidential* and *public* Clients) and three Client Profiles (*Web Applications*, *Browser / User-Agent based Applications*, and *Native Applications*).

This profile includes specific design considerations related to security and platform capabilities for these different Client Types and Profiles.

> **Note:** The iGov and NL GOV Assurance profiles for OAuth 2.0 use a slightly different segregation of Client Types: *Full Clients* and *Native Clients* act on behalf of a End-User and *Direct Access Clients* act on behalf of themselves (e.g. those Clients that facilitate bulk transfers). *Direct Access Clients* are out of scope for this profile; *Full Clients* and *Native Clients* are treated as *Web applications* and *Native applications* respectively. This profile follows the OAuth 2.0 specification [RFC6749] instead, as it allows for better provisioning of specific security considerations specific to the different Client types and it aligns better to the Security Best Practices for the different Client profiles.

The following design considerations apply to all Clients:

- Clients *MUST* use 'Proof Key for Code Exchange' [RFC7636] to protect calls to the Token Endpoint.
- Clients *SHOULD* restrict its Client-Side script (e.g. JavaScript) execution to a set of statically hosted scripts via a 'Content Security Policy' [CSP].
- Clients *SHOULD* use 'Subresource Integrity' [SRI] to verify that any dependencies they include (e.g. via a Content Delivery Network) are not unexpectedly manipulated.

### § 4.1.1 Web Applications

*Web applications* are applications that run on a web server and are consumed through the user-agent ("browser") by the End-User. Web applications are capable of securely authenticating themselves and of maintaining the confidentiality of secrets (e.g. Client credentials and tokens) and are therefore considered *confidential* Clients (OAuth 2.0 [RFC6749], Section 2.1).

### § 4.1.2 Browser-based Applications

*Browser-based applications* are applications that are dynamically downloaded and executed in a web browser that are also sometimes referred to as *user-agent-based applications* or *single-page applications*. Browser-based applications are considered to be not capable of maintaining the confidentiality of secrets, as they may be vulnerable to several types of attacks, including Cross-Site Scripting (XSS), Cross Site Request Forgery (CSRF) and OAuth token theft. Browser-based applications are considered *public* Clients (OAuth 2.0 [RFC6749], Section 2.1).

- Browser-based applications *SHOULD* follow the best practices specified in [OAuth2.Browser-Based-Apps].

### § 4.1.3 Native and Hybrid Applications

*Native applications* are applications installed and executed on the device used by the End-User (i.e. desktop applications, native mobile applications). Native applications can sufficiently protect dynamically issued secrets, but are not capable of maintaining the confidentiality of secrets that are statically included as part of an app distribution. Therefore, Native applications are considered *public* Clients, except when they are provisioned per-instance secrets via mechanisms like Dynamic Client Registration (OAuth 2.0 [RFC6749], Section 2.1).

*Hybrid applications* are applications implemented using web-based technology but distributed as a native app; these are considered equivalent to native applications for the purpose of this profile.

- Native applications *MUST* follow the best practices as specified in OAuth 2.0 for Native Apps [RFC8252].
- The use of *confidential* Native applications (which are provisioned per-instance secrets) is *RECOMMENDED* over *public* Native applications, as *confidential* Clients provide better means to perform secure Client Authentication.
- Native applications *MUST* use an external user-agent or "in-app browser tab" to make authorization requests; an "embedded user-agent" or "web-view" components *MUST NOT* be used for this purpose. See 'OAuth 2.0 for Native apps' [RFC8252] for more information on the "in-app browser tab" feature and support on various platforms.

## § 4.2 Authorization Endpoint

### § 4.2.1 Authentication Request

The following describes the supported OpenID Connect Authorization Code Flow parameters for use with a NL Gov compatible OpenID Provider. Some of these requirements are inherited as specified in Section 2.1.1 of [OAuth2.NLGov].

Request Parameters:

`client_id`

- *REQUIRED*. Valid OAuth 2.0 Client Identifier. *MUST* have the value as obtained during registration. Identical as in [OAuth2.NLGov].

`response_type`

- *REQUIRED*. *MUST* have value `code` for the Authorization Code Flow. Identical as in [OAuth2.NLGov].

`scope`

- *REQUIRED*. Indicates the access privileges being requested. *MUST* contain at least the value `openid` and *SHOULD* contain a specific scope for which access is requested.

`redirect_uri`

- *REQUIRED*. Indicates a valid endpoint where the Client will receive the authentication response. *MUST* be an absolute HTTPS URL unless the Client is a native application operating on a desktop device. In case of a native application on a desktop, this *MAY* be an absolute HTTP URL with the literal loopback IP address and port number the Client is listening on as hostname. *MUST NOT* use `localhost` for loopback addresses, see [RFC8252] Sections 7.3 and 8.3. *MUST* exactly match one of the Redirection URI values for the Client pre-registered at the OpenID Provider, except for the port URI component on loopback addresses for native applications on desktops. Inter-app redirect URIs for Native applications on mobile devices *MUST* use Claimed `https` Scheme URI Redirection, as specified in Section 7.2 of [RFC8252].

`state`

- *REQUIRED*. Unguessable random string generated by the Client, used to protect against Cross-Site Request Forgery (CSRF, XSRF) attacks. Must contain at least 128 bits of cryptographic random to avoid guessing. Returned to the Client in the Authentication Response. Identical as in [OAuth2.NLGov].

`nonce`

- *REQUIRED*. Unguessable random string generated by the Client, used to associate a Client session with an ID Token and to protect against replay attacks. Must contain at least 128 bits of cryptographic random to avoid guessing. Returned to the Client in the ID Token. See also [OpenID.Core], Section 15.5.2 for implementation notes.

`acr_values`

- *OPTIONAL*. Lists the acceptable LoAs for this authentication. Under this profile, `acr_values` takes precedence over `vtr`. See also Section 5.2.3. Identical as in [OpenID.Core].

`vtr`

- *OPTIONAL*. *MUST* be set to a value as described in Section 6.1 of Vectors of Trust [RFC8485]. *MUST NOT* be used when `acr_values` is set or when the `acr` Claim is requested via the `claims` parameter. See also Section 5.2.4.

`claims`

- *OPTIONAL*. This parameter is used to request specific Claims. The value is a JSON object listing the requested Claims, as specified in section 5.5 of [OpenID.Core].

`code_challenge`

- *REQUIRED*. Code challenge as in PKCE [RFC7636].

`code_challenge_method`

- *REQUIRED*. *MUST* use the value of `S256`.

> EXAMPLE 1
> A sample request may look like:
>
> ```
> https://idp-p.example.com/authorize?
> client_id=55f9f559-2496-49d4-b6c3-351a586b7484
> &nonce=cd567ed4d958042f721a7cdca557c30d
> &response_type=code
> &scope=openid+email
> &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
> &state=481e9c0c52e751a120fd90f7f4b5a637
> &acr_values=http%3a%2f%2feidas.europa.eu%2fLoA%2fsubstantial
> &code_challenge=E9Melhoa2OwvFrEMTJguCHaoeK1t8URWbuGJSstw-cM
> &code_challenge_method=S256
> ```

### 4.2.2 Request Objects

Clients *MAY* optionally send requests to the Authorization Endpoint using the `request` or `request_uri` parameter as defined by OpenID Connect [OpenID.Core], section 6. Passing a Request Object by reference using the `request_uri` is preferred because of browser limits and network latency.

Request Objects *MUST* be signed by the Client's registered key. Request Objects *MAY* be encrypted to the OpenID Provider's public key. When sending Request Objects by reference, Clients *MUST* pre-register `request_uri` values with the OpenID Provider at registration and *MUST* only use pre-registered values for `request_uri`.

### 4.2.3 Authentication Response Validation

All Clients *MUST* validate the following in received Authentication Responses:

`state`

- The `state` response parameter *MUST* be present and *MUST* equal the `state` request parameter sent in the Authentication Request.

This in line with OpenID Connect Core ([OpenID.Core], Section 3.1.2.7), which equals to OAuth 2.0 ([RFC6749], Section 4.1.2 and 10.12). Verifying the `state` returned in the Authorization Response is part of CSRF mitigation measures and will help prevent attacks with late or stale responses, among others.

## 4.3 Token Endpoint

### 4.3.1 Client Authentication

Confidential Clients, as defined in Section 4.1, *MUST* authenticate to the OpenID Provider using either:

- a JWT assertion as defined by the 'JWT Profile for OAuth 2.0 Client Authentication and Authorization Grants' [RFC7523] using only the `private_key_jwt` method defined in [OpenID.Core]; or
- mutually authenticated TLS, as specified in [RFC8705]. In case of a mutual TLS connection (mTLS) between the Client and the server, the JWT assertion *SHOULD* be omitted and the

`client_id` parameter *MUST* be included.

Public Clients *MAY* authenticate to the OpenID Provider. However, the OpenID Provider *MUST NOT* rely on public Client Authentication for the purpose of identifying the Client.

Clients *MUST NOT* use more than one authentication method in each request.

### § 4.3.2 Token Request

The following describes the supported parameters for the Token Request. Some of these requirements are inherited as specified in Section 2.3.1 of [OAuth2.NLGov].

The following parameters are specified:

`grant_type`

- *REQUIRED*. *MUST* contain the value `authorization_code`. Identical as in [OAuth2.NLGov].

`code`

- *REQUIRED*. The value of the `code` parameter returned in the Authorization Response. Clients *MUST NOT* use the same authorization code more than once. Identical as in [OAuth2.NLGov].

`client_assertion`

- *REQUIRED*, in case `private_key_jwt` is used for Client Authentication. The value of the signed Client Authentication JWT generated as described in [OAuth2.NLGov]. The OpenID Client *MUST* generate a new assertion JWT for each call to the Token Endpoint.

`client_assertion_type`

- *REQUIRED*, in case `client_assertion` is present. *MUST* be set to `urn:ietf:params:oauth:client-assertion-type:jwt-bearer`.

`client_id`

- *REQUIRED*, in case mutually authenticated TLS is used for Client Authentication.

`code_verifier`

- *REQUIRED*. Code verifier as in PKCE [RFC7636].

### § 4.3.3 Token Response Validation

All Clients *MUST* validate the following in received Token Responses:

- Follow the Token Response validation rules in [RFC6749], Sections 5.1 and 10.12.
- Validate the Access Token according to [OpenID.Core], Section 3.1.3.8.
- Validate the ID Token according to [OpenID.Core], Section 3.1.3.7, as well as the below mentioned requirements for validating the ID Token.

This in line with [OpenID.Core], Section 3.1.3.5.

### § 4.3.4 ID Tokens

All Clients *MUST* validate the signature of an ID Token before accepting it. Validation can be done using the public key of the issuing server, which is published in JSON Web Key (JWK) format. ID Tokens *MAY* be encrypted using the appropriate key of the requesting Client.

Clients *MUST* verify the following in received ID tokens:

`iss`

- The `issuer` Claim is the Uniform Resource Locater (URL) of the expected Issuer. Identical as in [OpenID.iGov].

`aud`

- The `audience` Claim contains the Client ID of the Client. Identical as in [OpenID.iGov].

`nonce`

- The `nonce` parameter in the ID Token *MUST* equal the `nonce` request parameter sent in the Authentication Request. This is in line with [OpenID.Core], Section 3.1.3.7.

`exp`, `iat`, `nbf`

- The `expiration`, `issued at`, and `not before` timestamps for the token are within acceptable ranges. These Claims are formatted as Unix Time Stamps (number of seconds since 1970-01-01T00:00:00Z UTC). Values for `iat` and `nbf` *MUST* lie in the past and `exp` *MUST* lie in the future; the acceptable range for how far away `iat` is in the past is specific to the Client. This is in line with [OpenID.iGov].

`acr`

- The Level of Assurance received in the `acr` Claim is at least the Level of Assurance requested. See also Section 5.2.3. This is in line with [OpenID.Core], Section 3.1.3.7.

`represents`

- The `represents` Claim, if applicable, identifies the represented service consumer on behalf of which the End-User intends to authenticate. Any Client *MUST* be able to process `represents` Claims. As an exception, `represents` Claims *MAY* be ignored by the Client if, and only if, it is explicitly agreed upon beforehand that no Representation will be provided.

## § 4.4 Discovery

All Clients *SHOULD* use OpenID Provider discovery to avoid manual configuration and risk of mistakes.

Clients *SHOULD* acquire OpenID Provider metadata using either 'OpenID Connect Discovery 1.0' ([OpenID.Discovery] Section 4) or 'OAuth 2.0 Authorization Server Metadata' ([RFC8414] Section 3) via one of the Discovery endpoints provided by the OpenID Provider. See also Section 5.4.

Clients *SHOULD NOT* use OpenID Provider Issuer Discover using WebFinger (as described in [OpenID.Core], Section 2) to avoid privacy issues such as leaking information to unknown locations.

Clients *SHOULD* follow caching directives provided by the OpenID Provider via HTTP headers [RFC7234] for the OpenID Provider's Discovery and `jwks` endpoints. This to avoid having to unnecessarily re-retrieve these documents while getting fresh updates of these documents when they have changed.

Clients *SHOULD* support `signed_metadata` as specified in [RFC8414] Section 2.1. In case signed metadata is available, this *MUST* be used over non-signed metadata and the signature *MUST* be verified prior to further utilizing any contents.

Clients *MUST* use the public keys obtained from the `jwks` endpoint to validate the signature on tokens or to encrypt Request Objects to the OpenID Provider.

## § 4.5 Registration

All Clients *MUST* register with the OpenID Provider.

Native Clients *MUST* either be provisioned a unique per-instance Client identifier or be registered as *public* Clients by using a common Client identifier; browser-based Clients *MUST* be registered

as *public* Clients.

Clients *SHOULD* use Dynamic Registration as per [RFC7591] to reduce manual labor and the risks of configuration errors. Dynamic Client Registration Management Protocol [RFC7592] *MAY* be used by Clients.

In case a native Client is using per-instance registration, the Client *MUST* use Dynamic Registration.

## § 5. OpenID Provider profile

For OpenID Providers the following items are applicable:

- OpenID Providers *MUST* implement all *Mandatory to Implement Features for All OpenID Providers* (Section 15.1) and all *Mandatory to Implement Features for Dynamic OpenID Providers* (Section 15.2) of [OpenID.Core]. Note that these Mandatory to Implement features include required support for the Hybrid Flow for authentication (Response Types `id_token` and `id_token token`). This profile deviates from this requirement, as this profile specifically forbids the use of the Hybrid Flow (see also Chapter 3).
- OpenID Providers *MUST* support and require the use of 'Proof Key for Code Exchange' ([RFC7636]) using only the `S256` verification method and a code verifier with at least 43 and at most 128 cryptographically random characters to allow Clients to protect calls to the Token Endpoint.
- OpenID Providers *MUST* apply the necessary 'Cross-Origin Resource Sharing' ([CORS]) headers to allow browsers to protect requests to its endpoints and *SHOULD NOT* use wildcard origins.
- OpenID Providers that support Web Applications *SHOULD* follow the best practices specified in [OAuth2.Browser-Based-Apps].
- OpenID Providers that support Native Applications *MUST* follow the best practices specified in OAuth 2.0 for Native Apps [RFC8252].

## § 5.1 Authorization Endpoint of the Provider profile

### § 5.1.1 Request Objects of the Provider profile

OpenID Providers *MUST* accept requests containing a Request Object signed by the Client's private key. OpenID Providers *MUST* validate the signature on such requests against the Client's registered public key. OpenID Providers *MUST* accept Request Objects encrypted to the OpenID Provider's public key.

OpenID Providers *SHOULD* accept Request Objects by reference using the `request_uri` parameter. The Request Object can be either hosted by the Client or pushed to the OpenID Provider prior to the Authentication Request. OpenID Providers *MUST* verify that the `request_uri` parameter exactly matches one of the `request_uri` values for the Client pre-registered at the OpenID Provider, with the matching performed as described in Section 6.2.1 of [RFC3986] (Simple String Comparison).

Using Request Objects allows for Clients to create a request that is protected from tampering through the browser, allowing for a higher security and privacy mode of operation for Clients and applications that require it. Clients are not required to use Request Objects, but OpenID Providers are required to support requests using them.

> Note that when a Request Object is used (either passed by value or by reference), the Client *MAY* send the parameters included in the Request Object duplicated in the query parameters as well for backwards compatibility (so that the request is a valid OAuth 2.0 Authorization Request). However, the OpenID Provider *MUST* only consider the parameters included in the Request Object and ignore the duplicated query parameters.

## § 5.2 Token Endpoint of the Provider profile

### § 5.2.1 Token Request Validation

OpenID Providers *MUST* validate all incoming Token Requests according to [OpenID.Core], Section 3.1.3.2.

In addition, OpenID Providers *MUST* validate the `code_verifier` value against the `code_challenge` and `code_challenge_method` values specified by the Client in the Authorization Request according to [RFC7636], Section 4.6.

### § 5.2.2 ID Tokens of the Provider profile

All ID Tokens *MUST* be signed by the OpenID Provider's private signature key. ID Tokens *MAY* be encrypted using the appropriate key of the requesting Client.

The ID Token *MUST* expire and *SHOULD* have an active lifetime no longer than five minutes. Since the ID Token is consumed by the Client and not presented to remote systems, it is *RECOMMENDED* that expiration times are kept as short as possible.

The Token Response includes an Access Token (which can be used to make a UserInfo request) and ID Token (a signed and optionally encrypted JSON Web Token). This profile imposes the following requirements on the Claims used in ID Tokens:

`iss`

- *REQUIRED*. The `issuer` field is the Uniform Resource Locator (URL) of the expected Issuer. Identical as in [OpenID.iGov].

`aud`

- *REQUIRED*. The `audience` field contains the Client ID of the Client. Identical as in [OpenID.iGov].

`sub`

- *REQUIRED*. The identifier of the authenticated End-User, also known as the subject. OpenID Providers *MUST* support a pairwise identifier in accordance with the OpenID Connect specification [OpenID.Core], section 8.1. See Pairwise Identifiers on when it may be useful to relax this requirement. Identical as in [OpenID.iGov].

`sub_id_type`

- *OPTIONAL*. The type of identifier passed in the `sub` Claim. In order to support multiple types of identifiers in an interoperable way, the type of identifier used for the identifier in the `sub` Claim *SHOULD* be explicitly included. The value of the `sub_id_type` *MUST* be a URI. Values supported by the OpenID Provider are provided via the Discovery endpoint.

`acr`

- *OPTIONAL*. The LoA the End-User was authenticated at. *MUST* be at least the requested Level of Assurance value requested by the Client (either via the `acr_values` or `claims` parameters) or - if none was requested - a Level of Assurance established through prior agreement. See also Section 5.2.3. As eIDAS is leading in most scenarios targeted by this profile, using the `acr` Claim to express the Level of Assurance is preferred over Vectors of Trust (`vot`).

`nonce`

- *REQUIRED*. *MUST* contain the `nonce` value that was provided in the Authentication Request. Identical as in [OpenID.iGov].

`jti`

- *REQUIRED*. A unique identifier for the token, which can be used to prevent reuse of the token. The value of `jti` *MUST* uniquely identify the ID Token between sender and receiver for at least 12 months.

`auth_time`

- *REQUIRED* if `max_age` was specified in the request or when `auth_time` was requested as an Essential Claim. Otherwise `auth_time` is *OPTIONAL* and *SHOULD* be included if the OpenID Provider can assert an End-User's authentication intent was demonstrated. For example, a login event where the End-User took some action to authenticate. See also Section 15.1 of [OpenID.Core].

`exp`, `iat`, `nbf`

- *REQUIRED*. The `expiration`, `issued at`, and `not before` timestamps indicate when the token expires, was issued and becomes valid, respectively. The expiration time for ID Tokens is specific to the OpenID Provider. In line with [OpenID.iGov].

`represents`

- *REQUIRED* in case Representation is applicable, the `represents` Claim provides information about the effective authorization due to a Representation Relationship for the End-User.

`alt_sub`

- *OPTIONAL*. Describes alternative Subject Identifiers for the authenticated End-User in the context of a specific audience. The value of `alt_sub` is an array of objects, each of which *MUST* contain `sub` and `aud` Claims to uniquely identify the authenticated End-User and the audience for the alternative Subject Identifier and *SHOULD* contain a `sub_id_type` Claim to explicitly indicate the type of identifier used in the `sub` claim if the OpenID Provider supports multiple types of subject identifiers.

`vot`

- *OPTIONAL*. The vector value as specified in Vectors of Trust. *MUST NOT* be included when `acr` is included. See also Section 5.2.4.

`vtm`

- *REQUIRED* if `vot` is provided. The trustmark URI as specified in Vectors of Trust. See also Section 5.2.4.

Other Claims *MAY* be included. See Claims Request below on how such Claims *SHOULD* be requested by the Client to be provided by the OpenID Provider.

EXAMPLE 2

This example ID Token has been signed using the server's RSA key:

```
eyJhbGciOiJSUzI1NiJ9.eyJleHAiOjE0MTg2OTk0
MTIsInN1YiI6IjZXWlFQcG5ReFYiLCJzdWJfaWRfd
HlwZSI6InVybjpubC1laWQtZ2RpOjEuMDppZDpwc2
V1ZG9ueW0iLCJub25jZSI6IjE4ODYzN2IzYWYxNGE
iLCJhdWQiOlsiYzFiYzg0ZTQtNDdlZS00YjY0LWJi
NTItNWNkYTZjODFmNzg4Il0sImFsdF9zdWIiOlt7I
mF1ZCI6IjM3OWIwMjJkLWQ5ZDAtNGM0My1iN2RlLT
I5MGEwMjNlYjQ2MSIsInN1YiI6InhTSENyRm05Qkc
iLCJzdWJfaWRfdHlwZSI6InVybjpubC1laWQtZ2Rp
OjEuMDppZDpwc2V1ZG9ueW0ifV0sImlzcyI6Imh0d
HBzOi8vaWRwLXAuZXhhbXBsZS5jb20vIiwiYWNyIj
oiaHR0cDovL2VpZGFzLmV1cm9wYS5ldS9Mb0Evc3V
ic3RhbnRpYWwiLCJpYXQiOjE0MTg2OTg4MTIsImp0
aSI6ImE2NWM1NjBkLTA4NWMtNDY2ZS05N2M1LWY4N
jM5ZmNhNWVhNyIsIm5iZiI6MTQxODY5OTExMn0
```

Its Claims are as follows:


```
{
       "auth_time": 1418698782,
       "exp": 1418699412,
       "sub": "6WZQPpnQxV",
       "sub_id_type": "urn:nl-eid-gdi:1.0:id:pseudonym",
       "nonce": "188637b3af14a",
       "aud": [
         "c1bc84e4-47ee-4b64-bb52-5cda6c81f788"
       ],
       "alt_sub": [{
         "aud": "379b022d-d9d0-4c43-b7de-290a023eb461",
         "sub": "xSHCrFm9BG",
         "sub_id_type": "urn:nl-eid-gdi:1.0:id:pseudonym"
       }],
       "iss": "https://idp-p.example.com/",
       "acr": "http://eidas.europa.eu/LoA/substantial",
       "iat": 1418698812,
       "jti": "a65c560d-085c-466e-97c5-f8639fca5ea7",
       "nbf": 1418699112,
 }
```

### § 5.2.3 Pairwise Identifiers

Pairwise Subject Identifiers specified in OpenID Connect Core [OpenID.Core] Section 8 help protect an End-User's privacy by allowing an OpenID Provider to represent a single End-User with a different Subject Identifier (`sub`) for every Client the End-User connects to. This technique can help mitigate correlation of an End-User between multiple Clients and therewith tracking of End-Users between different sites and applications.

Use of pairwise identifiers does not prevent Clients from correlating data based on other identifying attributes such as names, phone numbers, email addresses, document numbers, or other attributes. However, since not all transactions require access to these attributes, but a Subject Identifier is always required, a pairwise identifier will aid in protecting the privacy of End-Users as they navigate the system.

OpenID Providers *MUST* support pairwise identifiers for cases where correlation of End-User's activities across Clients is not appropriate. OpenID Providers *MAY* support public identifiers for frameworks where public identifiers are required, or for cases where public identifiers are shared as attributes and the framework does not have a requirement for subject anonymity.

*Burgerservicenummers (BSN), Rechtspersonen en Samenwerkingsverbanden Identificatienummers (RSIN)* and *Kamer van Koophandel (KvK) nummers* are considered public sectoral identifiers and therefore *MUST NOT* be used as Subject Identifiers in case correlation of End-User's activities across Clients is not appropriate. In such cases, the use of Polymorphic Pseudonyms or Polymorphic Identities is preferred.

> Note that BSNs *MUST* only be used by Relying Parties for Services eligible for using the BSN according to Dutch Law and that the BSN, or token containing it, *SHOULD* be encrypted.

### § 5.2.4 Representation Relationships

In Use Cases that involve Representation Relationships, Representation Relationships are explicitly mentioned in the form of a `represents` Claim, analogous to the Delegation Semantics specified in [RFC8693].

> **Note**: Whereas [RFC8693] lists the End-User in the `act` or `may_act` Claims and the represented service consumer in the `sub` Claim, this is reversed in this profile: the End-User is listed in the `sub` Claim and the represented service consumer is listed in the `represents` Claim. Reason for this is to mitigate the risk that a Client that does not explicitly supports the Representation Use Cases cannot recognize the difference between an End-User that authenticates on behalf of himself or on behalf of someone else via Representation.

As such, all Clients *MUST* process `represents` Claims used, in case Representation can be applicable in the context of the OpenID Client and OpenID Provider. As an exception, `represents` Claims *MAY* be ignored by the Client if, and only if, it is explicitly agreed upon beforehand that no Representation will be provided.

This profile specifies Representation Relations in ID Tokens as follows:

- The End-User is always identified by the `sub` Claim;
- The represented service consumer is mentioned in the `represents` Claim.
- In case a chain representation is applicable, the representation chain is represented as a series of nested `represents` Claims with the represented service consumer listed as the deepest nested `represents` Claim.
- Each `represents` Claim *MUST* contain `sub` and `iss` Claims to uniquely identify the represented party and *SHOULD* contain a `sub_id_type` Claim to explicitly indicate the type of identifier used in the `sub` claim if the OpenID Provider supports multiple types of subject identifiers.
- `represents` Claims *MAY* contain additional Claims (e.g. `email`) to provide additional useful information about the represented party.
- Claims within the `represents` Claim pertain only to the identity of that party and *MUST NOT* contain Claims that are not related to the represented party, such as top-level Claims `exp`, `nbf`, and `aud`.

EXAMPLE 3

A sample chain representation for a requested scope `urn:uuid:a9e17a2e-d358-406d-9d5f-ad6045f712ba` may look like (note: the requested scope also includes the required `openid` scope; Claims that do not add to the example are omitted for readability):

```
{
  "scope": "openid urn:uuid:a9e17a2e-d358-406d-9d5f-ad6045f712ba",
  /* End-User - representing the service consumer */
  "sub": "RKyLpEVr1L",
  "sub_id_type": "urn:nl-eid-gdi:1.0:id:pseudonym",
  "iss": "urn:uuid:b556992a-e233-4fdc-915a-e2b52d3cc355",
  "represents": {
    /* Intermediary in representation chain - an organization in this
    "sub": "492099595",
    "sub_id_type": "urn:nl-eid-gdi:1.0:id:RSIN",
    "iss": "urn:uuid:28e0686f-20ff-41bd-8520-57b9c68cc9a3",
    "alt_sub": {
      "sub": "27381312",
      "sub_id_type": "urn:nl-eid-gdi:1.0:id:KvKnr",
      "iss": "urn:uuid:ebc29845-d35f-4c6a-bbb2-a59fdcb1cc6b"
    }
    "represents": {
      /* service consumer - represented by the End-User */
      "sub": "4Yg8u72NxR",
      "sub_id_type": "urn:nl-eid-gdi:1.0:id:pseudonym",
      "iss": "urn:uuid:55291cc0-fd2a-4eb6-b444-5b2783e62673"
    }
  }
}
```

### § 5.2.5 Authentication Context

Whereas the iGov Assurance Profile for OpenID Connect [OpenID.iGov] recommends the use of Vectors of Trust (`vot`) to determine the amount of trust to be placed in digital transactions, using Authentication Context Class References (`acr`) instead is *RECOMMENDED* by this profile, due to their better alignment to the Levels of Assurance (LoA) defined by the `eIDAS` standards that are used in the European Union.

OpenID Providers *SHOULD* use eIDAS Level of Assurance (LoA) values for the `acr` Claim, but *MAY* use different values if eIDAS is not applicable. The eIDAS Level of Assurance values are defined as URIs in [eIDAS.SAML], Section 3.2.

OpenID Providers *MUST* provide a Level of Assurance as `acr` value that is at least the requested Level of Assurance value requested by the Client (either via the `acr_values` or `claims` parameters) or - if none was requested - a Level of Assurance established through prior agreement.

OpenID Providers *MUST NOT* provide Authentication Methods References (`amr`), but *MUST* use Authentication Context Class References (`acr`) instead.

Clients *MAY* send an `vtr` (Vectors of Trust Request) parameter. If both the `vtr` and `acr_values` are in the request, the `acr_values` *MUST* take precedence and the `vtr` *MUST* be ignored.

> **Note:** Risk Based Authentication (RBA) should be an integral part of the LoA framework that is used by an OpenID Provider (the Identity Provider), such that the risk criteria for the resulting authentication are at least sufficient to meet the applicable LoA. That is, an OpenID Provider *MAY* apply RBA to require authentication methods with enhanced security or ease towards more user friendly methods when allowed by evaluated risk for an authentication, as long as the trust framework requirements are met. Selection of and criteria for any LoA framework are, however, situation specific and beyond the scope of this profile.

### § 5.2.6 Vectors of Trust

OpenID Providers *MAY* provide `vot` (Vectors of Trust) and `vtm` (Vector Trust Mark) values in ID Tokens only if the `acr` Claim is not requested by the Client (either via the `acr_values` or `claims` parameters). More information on Vectors of Trust is provided in [RFC8485].

### § 5.2.7 Access Tokens

This profile requires an Access Token to be in JWT form. This is in line with the underlying NL GOV Assurance profile for OAuth 2.0 [OAuth2.NLGov].

Using a JWT formatted Access Token allows any OpenID Client to consume and verify a token without the need for introspection, thus reducing the dependency on an interaction with an external endpoint. As a result this may reduce load and availability requirements on the OpenID Provider. Furthermore, it provides a more uniform format over Access Token, ID Token, UserInfo response and Introspection response.

> Note that ID Tokens and UserInfo responses are primarily intended for the Client. The Access Token is primarily intended for consumption by a Resource Server. The Introspection response is intended for the requestor of an Introspection, which can be either a Client or Resource Server. The Resource Server is typically not considered as an actor in OpenID Connect, but OpenID Providers will often act as Authorization Servers. In the case of Service Intermediation this is applicable by definition. This profile does not directly place any constraints on the placement of Claims in various tokens or response messages. Claims may be placed in any of the four tokens/response messages, unless explicitly specified otherwise. This allows for maximum flexibility and interoperability.

### 5.2.8 Refresh Tokens

OpenID Providers *MAY* issue Refresh Tokens to Clients; when used, Refresh Tokens *MUST* be one-time-use or sender-constrained.

OpenID Providers *MAY* cryptographically bind Refresh Tokens to the specific Client instance (see also [OAuth2.1], Section 6.1); other methods to create sender-constrained Refresh Tokens *MAY* be applied as well.

For security reasons, Refresh Tokens that are not sender-constrained *MUST* be one-time-use, i.e. with every Access Token refresh response the OpenID Provider can issue a new Refresh Token and *MUST* invalidate the previous Refresh Token (see also [RFC6819], Section 5.2.2.3 and [OAuth2.1], Section 6.1).

Refresh Tokens *MUST* expire if the Client has been inactive for some time, i.e., the Refresh Token has not been used to obtain fresh Access Tokens for some time. The expiration time is at the discretion of the OpenID Provider, but *MUST NOT* exceed a maximum of 6 hours, preferably shorter.

For public Clients, no cryptographic key or Client Authentication method for binding Refresh Tokens to a specific Client is available. Public Clients therefore *MUST* use one-time-use Refresh Tokens with a limited validity, if applied.

## 5.3 UserInfo Endpoint

OpenID Providers *MUST* support the UserInfo Endpoint and, at a minimum, the `sub` (subject) Claim. It is expected that the `sub` Claim will remain pseudonymous in Use Cases where obtaining personal information is not needed.

Support for a UserInfo Endpoint is important for maximum Client implementation interoperability even if no additional End-User information is returned. Clients are not required to call the UserInfo Endpoint, but *SHOULD NOT* receive an error if they do.

> EXAMPLE 4
> In an example transaction, the Client sends a request to the UserInfo Endpoint like the following:
>
> ```
> GET /userinfo HTTP/1.1
> Authorization: Bearer eyJhbGciOiJSUzI1NiJ9.eyJleHAiOjE0MTg3MDI0MTIsIm
>   F1ZCI6WyJjMWJjODRlNC00N2VlLTRiNjQtYmI1Mi01Y2RhNmM4MWY3ODgiXSwiaXNzIj
>   iaHR0cHM6XC9cL2lkcC1wLmV4YW1wbGUuY29tXC8iLCJqdGkiOiJkM2Y3YjQ4Zi1iYzg
>   LTQwZWMtYTE0MC05NzRhZjc0YzRkZTMiLCJpYXQiOjE0MTg2OTg4MTJ9i.HMz_tzZ90_
>   0QZS-AXtQtvclZ7M4uDAs1WxCFxpgBfBanolW37X8h1ECrUJexbXMD6rrj_uuWEqPD73
>   oWRo0rOnoKJAgbF1GhXPAYnN5pZRygWSD1a6RcmN85SxUig0H0e7drmdmRkPQgbl2wMh
>   -6h2Oqw-ize4dKmykN9UX_2drXrooSxpRZqFVYX8PkCvCCBuFy2O-HPRov_SwtJMk5qj
>   WMyn2I4Nu2s-R20aCA-7T5dunr0iWCkLQnVnaXMfA22RlRiU87nl21zappYb1_EHF9eP
>   q3Q353cDUY7vje8m2kKXYTgc_bUAYuW-W3SMSw5UlKaHtSZ6PQICoA
> Accept: application/json
> Host: idp-p.example.com
> Connection: Keep-Alive
> ```
>
> And receives a document in response like the following:
>
> ```
> HTTP/1.1 200 OK
> Date: Mon, 16 Dec 2019 03:00:12 GMT
> Access-Control-Allow-Origin: *
> Content-Type: application/json
> Content-Language: en-US
> Content-Length: 333
> Connection: close
> ```
>
> ```
> {
>    "sub": "6WZQPpnQxV",
>    "iss": "https://idp-p.example.com",
>    "given_name": "Stephen",
>    "family_name": "Emeritus",
> }
> ```

OpenID Providers *MUST* support the generation of JWT encoded responses from the UserInfo Endpoint. Responding with unsigned JSON objects when neither signing nor encryption are requested by the Client as part of the `userinfo_signed_response_alg` and `userinfo_encrypted_response_alg` Client metadata parameters registered as part of Client

Registration is *OPTIONAL*. Signed responses *MUST* be signed by the OpenID Provider's signing key, and encrypted responses *MUST* be encrypted with the authorized Client's public key. Please refer to Algorithms for more information on cryptographic algorithms and keys.

## § 5.4 Discovery

The OpenID Connect Discovery [OpenID.Discovery] standard provides a standard, programmatic way for Clients to obtain configuration details for communicating with OpenID Providers. Discovery is an important part of building scalable federation ecosystems.

OpenID Providers under this profile *MUST* publish their server metadata to help minimize configuration errors and support automation for scalable deployments.

- Exposing a Discovery endpoint does NOT inherently put the OpenID Provider at risk to attack. Endpoints and parameters specified in the Discovery document *SHOULD* be considered public information regardless of the existence of the Discovery document.
- Access to the Discovery document *MAY* be protected with existing web authentication methods if required by the OpenID Provider. Credentials for the Discovery document are then managed by the OpenID Provider. Support for these authentication methods is outside the scope of this profile.
- Endpoints described in the Discovery document *MUST* be secured in accordance with this profile and *MAY* have additional controls the Provider wishes to support.

### § 5.4.1 Discovery endpoint

All OpenID Providers are uniquely identified by a URL known as the `issuer` and *MUST* make a Discovery document in JSON format available at the path formed by concatenating `/.well-known/openid-configuration` to the `issuer` and *SHOULD* also make this Discovery document available at the path formed by concatenating `/.well-known/oauth-authorization-server` to the `issuer`. OpenID Providers *MAY* also publish their Discovery documents on other locations. All paths on which the Discovery document is published *MUST* use the `https` scheme.

> Note that for privacy considerations, only direct requests to the server metadata document *SHOULD* be used. The WebFinger method to locate the relevant OpenID Provider and its metadata, as described in [OpenID.Discovery] section 2, *MUST NOT* be supported.

### § 5.4.2 Discovery document

This profile imposes the following requirements upon the Discovery document:

`issuer`

- *REQUIRED*. The fully qualified Issuer URL of the OpenID Provider as defined by [RFC8414].

`authorization_endpoint`

- *REQUIRED*. The fully qualified URL of the OpenID Provider's Authorization Endpoint as defined by [RFC6749].

`token_endpoint`

- *REQUIRED*. The fully qualified URL of the OpenID Provider's Token Endpoint as defined by [RFC6749].

`userinfo_endpoint`

- *RECOMMENDED*. The fully qualified URL of the OpenID Provider's Userinfo Endpoint as defined by [OpenID.Core].

`registration_endpoint`

- *RECOMMENDED*. The fully qualified URL of the OpenID Provider's Dynamic Registration endpoint [RFC7591].

`introspection_endpoint`

- *OPTIONAL*. The fully qualified URL of the OpenID Provider's Introspection Endpoint as defined by 'OAuth 2.0 Token Introspection' [RFC7662].

`revocation_endpoint`

- *OPTIONAL*. The fully qualified URL of the OpenID Provider's Revocation Endpoint as defined by 'OAuth 2.0 Token Revocation' [RFC7009].

`jwks_uri`

- *REQUIRED*. The fully qualified URL of the OpenID Provider's public keys in JWK Set format. These keys can be used by Clients to verify signatures on tokens and responses from the OpenID Provider and for encrypting requests to the OpenID Provider.

`scopes_supported`

- *REQUIRED*. The list of scopes the OpenID Provider supports as defined by [RFC8414].

`response_types_supported`

- *REQUIRED*. JSON array containing the list of OAuth 2.0 `response_type` values that the OpenID Provider supports. In the context of this profile, the value *MUST* Be ['code'].

`grant_types_supported`

- *REQUIRED*. JSON array containing the list of OAuth 2.0 `grant_type` values that the OpenID Provider supports. In the context of this profile, the value *MUST* be ['authorization_code'].

`claims_parameter_supported`

- *OPTIONAL*. Boolean value specifying whether the OpenID Provider supports the use of the `claims` parameter, as defined by [OpenID.Discovery].

`claims_supported`

- *REQUIRED*. JSON array containing the list of Claims available in the supported scopes as defined by [OpenID.Discovery]. See Claims Supported.

`claim_types_supported`

- *OPTIONAL*. JSON array containing the list of Claim types that the OpenID Provider supports. *REQUIRED* when `aggregated` or `distributed` Claims are used. If omitted, the OpenID Provider only supports `normal` Claims. Identical to [OpenID.Discovery].

`sub_id_types_supported`

- *OPTIONAL*. JSON array containing the list of supported types of Subject Identifiers in the `sub` Claim of ID Tokens. The values *MUST* be URIs, the exact URIs to be used are situation specific; as an example encrypted BSNs and Pseudonyms could be specified with `urn:nl-eid-gdi:1.0:id:BSN` or `urn:nl-eid-gdi:1.0:id:Pseudonym` respectively.

`acr_values_supported`

- *OPTIONAL*. JSON array containing the list of supported Levels of Assurances, as defined by [OpenID.Discovery]. See Authentication Context.

`subject_types_supported`

- *REQUIRED*. JSON array containing the list of Subject Identifier types that this OpenID Provider supports. Valid types include `pairwise` and `public`.

`token_endpoint_auth_methods_supported`

- *REQUIRED*. JSON array containing the list of Client Authentication methods that this OpenID Provider supports. With respect to this profile, the allowed values are `private_key_jwt`, `tls_client_auth`, or both.

`id_token_signing_alg_values_supported`

- *REQUIRED*. JSON array containing the list of JWS signing algorithms (`alg` values) supported by the OpenID Provider for the ID Token to encode the Claims in a JWT. For more information, refer to Algorithms.

`id_token_encryption_alg_values_supported`

- *OPTIONAL*. JSON array containing the list of JWE encryption algorithms (`alg` values) supported by the OpenID Provider for the ID Token to encrypt the Content Encryption Key (CEK). *REQUIRED* when the OpenID Provider supports encryption of ID Tokens. For more information, refer to Algorithms.

`id_token_encryption_enc_values_supported`

- *OPTIONAL*. JSON array containing the list of JWE encryption algorithms (`enc` values) supported by the OpenID Provider for the ID Token to encrypt the Claims in a JWT using the CEK. *REQUIRED* when the OpenID Provider supports encryption of ID Tokens. For more information, refer to Algorithms.

`userinfo_signing_alg_values_supported`

- *REQUIRED*. JSON array containing the list of JWS signing algorithms (`alg` values) supported by the UserInfo Endpoint to encode the Claims in a JWT. For more information, refer to Algorithms.

`userinfo_encryption_alg_values_supported`

- *OPTIONAL*. JSON array containing the list of JWE encryption algorithms (`alg` values) supported by the OpenID Provider for the UserInfo Endpoint to encrypt the Content Encryption Key (CEK). *REQUIRED* when the OpenID Provider supports encryption of UserInfo responses. For more information, refer to Algorithms.

`userinfo_encryption_enc_values_supported`

- *OPTIONAL*. JSON array containing the list of JWE encryption algorithms (`enc` values) supported by the OpenID Provider for the UserInfo Endpoint to encrypt the Claims in a JWT using the CEK. *REQUIRED* when the OpenID Provider supports encryption of UserInfo responses. For more information, refer to Algorithms.

`request_object_signing_alg_values_supported`

- *REQUIRED*. JSON array containing the list of JWS signing algorithms (`alg` values) supported by the OpenID Provider for Request Objects. These algorithms are applicable for Request Objects passed by value and passed by reference. For more information, refer to Algorithms.

`request_object_encryption_alg_values_supported`

- *OPTIONAL*. JSON array containing the list of JWE encryption algorithms (`alg` values) supported by the OpenID Provider for Request Objects to encrypt the Content Encryption Key (CEK). *REQUIRED* when the OpenID Provider supports encryption of UserInfo responses. For more information, refer to Algorithms.

`request_object_encryption_enc_values_supported`

- *OPTIONAL*. JSON array containing the list of JWE encryption algorithms (`enc` values) supported by the OpenID Provider for Request Objects to encrypt the Claims in a JWT using the CEK. *REQUIRED* when the OpenID Provider supports encryption of UserInfo responses. For more information, refer to Algorithms.

`request_uri_parameter_supported`

- *OPTIONAL*. Boolean value which specifies whether the OpenID Provider accepts Request Objects passed by reference using the `request_uri` parameter. As per [OpenID.Core], the default value is `true`.

`require_request_uri_registration`

- *REQUIRED* and *MUST* have Boolean value `true` if the OpenID Provider accepts Request Objects passed by reference using the `request_uri` parameter. *OPTIONAL* otherwise. This parameter indicates that `request_uri` values used by the Client to send Request Objects by reference must always be pre-registered.

`signed_metadata`

- *RECOMMENDED*. A JWT, signed using JWS, containing metadata values about the OpenID Provider as claims, as specified in [RFC8414], Section 2.1.

EXAMPLE 5

The following example shows the JSON document found at a discovery endpoint for an OpenID Provider:


```
{
  "request_parameter_supported": true,
  "id_token_encryption_alg_values_supported": [
    "RSA-OAEP", "RSA-OAEP-256"
  ],
  "registration_endpoint": "https://idp-p.example.com/register",
  "userinfo_signing_alg_values_supported": [
    "RS256", "RS384", "RS512"
  ],
  "token_endpoint": "https://idp-p.example.com/token",
  "request_uri_parameter_supported": false,
  "request_object_encryption_enc_values_supported": [
    "A192CBC-HS384", "A192GCM", "A256CBC+HS512",
    "A128CBC+HS256", "A256CBC-HS512",
    "A128CBC-HS256", "A128GCM", "A256GCM"
  ],
  "token_endpoint_auth_methods_supported": [
    "private_key_jwt",
  ],
  "userinfo_encryption_alg_values_supported": [
    "RSA-OAEP", "RSA-OAEP-256"
  ],
  "subject_types_supported": [
    "public", "pairwise"
  ],
  "id_token_encryption_enc_values_supported": [
    "A192CBC-HS384", "A192GCM", "A256CBC+HS512",
    "A128CBC+HS256", "A256CBC-HS512", "A128CBC-HS256",
    "A128GCM", "A256GCM"
  ],
  "claims_parameter_supported": false,
  "jwks_uri": "https://idp-p.example.com/jwk",
  "id_token_signing_alg_values_supported": [
    "RS256", "RS384", "RS512"
  ],
  "authorization_endpoint": "https://idp-p.example.com/authorize",
  "require_request_uri_registration": false,
  "introspection_endpoint": "https://idp-p.example.com/introspect",
  "request_object_encryption_alg_values_supported": [
    "RSA-OAEP", "RSA-OAEP-256"
  ],
  "service_documentation": "https://idp-p.example.com/about",
```

```
  "response_types_supported": [
    "code", "token"
  ],
  "token_endpoint_auth_signing_alg_values_supported": [
    "RS256", "RS384", "RS512"
  ],
  "revocation_endpoint": "https://idp-p.example.com/revoke",
  "request_object_signing_alg_values_supported": [
    "HS256", "HS384", "HS512", "RS256", "RS384", "RS512"
  ],
  "claim_types_supported": [
    "normal"
  ],
  "grant_types_supported": [
    "authorization_code",
  ],
  "scopes_supported": [
    "profile", "openid", "doc"
  ],
  "userinfo_endpoint": "https://idp-p.example.com/userinfo",
  "userinfo_encryption_enc_values_supported": [
    "A192CBC-HS384", "A192GCM", "A256CBC+HS512","A128CBC+HS256",
    "A256CBC-HS512", "A128CBC-HS256", "A128GCM", "A256GCM"
  ],
  "op_tos_uri": "https://idp-p.example.com/about",
  "issuer": "https://idp-p.example.com/",
  "op_policy_uri": "https://idp-p.example.com/about",
  "claims_supported": [
    "sub", "name", "vot", "acr"
  ],
  "acr_values_supported" [
    "http://eidas.europa.eu/LoA/substantial",
    "http://eidas.europa.eu/LoA/high"
  ]
}
```

### § 5.4.3 Caching

It is *RECOMMENDED* that OpenID Providers provide caching directives through HTTP headers for the Discovery endpoint and the `jwks_uri` endpoint and make the cache valid for at least one week. OpenID Providers *SHOULD* document their change procedure. In order to support automated transitions to configuration updates, OpenID Providers *SHOULD* only make non-breaking changes and retain backward compatibility when possible. It is *RECOMMENDED* that

OpenID Providers monitor usage of outdated configuration options used by any OpenID Client and actively work with their administrators to update configurations. The above on caching and changes *MUST* be applied to the `jwks_uri` containing the OpenID Provider's key set as well.

### § 5.4.4 Public keys

The OpenID Provider *MUST* provide its public keys in JWK Set format, such as the following example JWK Set containing a PKIoverheid certificate chain and its 2048-bit RSA key (example certificates abbreviated):

EXAMPLE 6


```
{
  "keys": [
    {
      "alg": "RS256",
      "e": "AQAB",
      "n": "o80vbR0ZfMhjZWfqwPUGNkcIeUcweFyzB2S2T-hje83IOVct8gVg9Fx
            vHPK1ReEW3-p7-A8GNcLAuFP_8jPhiL6LyJC3F10aV9KPQFF-w6Eq6V
            tpEgYSfzvFegNiPtpMWd7C43EDwjQ-GrXMVCLrBYxZC-P1ShyxVBOze
            R_5MTC0JGiDTecr_2YT6o_3aE2SIJu4iNPgGh9MnyxdBo0Uf0TmrqEI
            abquXA1-V8iUihwfI8qjf3EujkYi7gXXelIo4_gipQYNjr4DBNl
            E0__RI0kDU-27mb6esswnP2WgHZQPsk779fTcNDBIcYgyLujlcUATEq
            fCaPDNp00J6AbY6w",
      "kty": "RSA",
      "kid": "rsa-PKIo",
      "x5c": [
        "MIIE3jCCA8agAwIBAgICAwEwDQYJKoZIhvcNAQEFBQAwYzELMAkGA
        1UEBhMCVVMxITAfBgNVBAoTGFRoZSBHbyBEYWRkeSBHcm91cCwgSW5
        jLjExMC8GA1UECxMoR2[...]TVSzGh6O1mawGhId/dQb8vxRMDsxux
        N89txJx9OjxUUAiKEngHUuHqDTMBqLdElrRhjZkAzVvb3du6/KFUJh
        eqwNTrZEjYx8WnM25sgVjOuH0aBsXBTWVU+4=",
        "MIIE+zCCBGSgAwIBAgICAQ0wDQYJKoZIhvcNAQEFBQAwgbsxJDAiB
        gNVBAcTG1ZhbGlDZXJ0IFZhbGlkYXRpb24gTmV0d29yazEXMBUGA1U
        EChMOVmFsaUNlcnQsIE[...]luYAzBgNVBAsTLFZhbGlDZXJ0IENsY
        XNzIDIgUG9saWN5IFZhbGlkYXRpb24gQXV0aG9yaXR5MSEwHwYDVQQ
        DExhodHRwOjZXRn453HWkrugp++85j09VZw==",
        "MIIC5zCCAlACAQEwDQYJKoZIhvcNAQEFBQAwgbsxJDAiBgNVBAcTG
        1ZhbGlDZXJ0IFZhbGlkYXRpb24gTmV0d29yazEXMBUGA1UEChMOVmF
        saUNlcnQsIEluYy4xNT[...]AzBgNVBAsTLFZhbGlDZXJ0IENsYXNz
        IDIgUG9saWN5IFZhbGlkYXRpb24gQXV0aMtsq2azSiGM5bUMMj4Qss
        xsodyamEwCW/POuZ6lcg5Ktz885hZo+L7tdEy8W9ViH0Pd"
      ],
      "use": "sig",
    }
  ]
}
```


In case PKIoverheid certificates are used, the certificate and entire certificate chain up until the root certificate *MUST* be included as either an `x5c` or as `x5u` parameter, according to [RFC7517] Sections 4.6 and 4.7. Parties *SHOULD* support the inclusion of the certificate chain as `x5c` parameter, for maximum interoperability. Parties *MAY* agree to use `x5u`, for instance for communication within specific environments.

The OpenID Provider *SHOULD* utilize the approaches described in [OpenID.Core], Sections 10.1.1 (signing keys) and 10.2.1 (encryption keys), to facilitate rotation of public keys.

Please refer to Algorithms for more information on eligible cryptographic methods and keys that can be used by OpenID Providers.

## § 5.5 Dynamic Registration

If the OpenID Provider is acting as an NL-Gov OAuth Authorization Server [OAuth2.NLGov], then Dynamic Registration *MUST* be supported in accordance with Section 3.1.3 of that specification.

Dynamic Registration *MUST* also be supported in combination with per-instance provisioning of secrets when registering Native Applications as confidential Clients.

In other cases, particularly when dealing with Browser-based applications or Native Apps, Dynamic Registration *SHOULD* be supported in accordance with the NL GOV Assurance profile for OAuth 2.0 [OAuth2.NLGov].

This profile imposes the following requirements upon the Client Registration request:

`Initial access tokens`

- In cases where the OpenID Provider limits the parties that are allowed to register Clients using Dynamic Registration (i.e. when open registration is not applicable), the use of an initial access token in the form of an OAuth2 Bearer token using the `Authorization` HTTP header [RFC6750] is *REQUIRED* for making Client Registration requests. In cases where open registration is applicable, the use of an initial access token is *OPTIONAL*.

`redirect_uris`

- *REQUIRED*. Array of Redirection URI values used by the Client. *MUST* be absolute HTTPS URLs. One of these registered Redirection URI values *MUST* exactly match the `redirect_uri` parameter value used in each Authorization Request.

- The only exception is when the Client is a Native Application operating on a desktop device and is exclusively registered as such. In such cases:

- the `redirect_uri` *MAY* contain absolute HTTP URLs with the literal loopback IP addresses and port numbers the Client is listening on as hostnames. *MUST NOT* use `localhost` as hostname for the loopback address, see [RFC8252] Sections 7.3 and 8.3; and

- even though the port number is part of the registered `redirect_uri`, the OpenID Provider *MUST* allow any port to be specified in the Authorization Request for loopback IP redirect

URIs.

`jwks_uri` *or* `jwks`

- Clients *SHOULD* reference their JSON Web Key (JWK) Set via the `jwks_uri` parameter rather than passing their JWK Set document by value using the `jwks` parameter, as it allows for easier key rotation. Also, the `jwks` and `jwks_uri` parameters *MUST NOT* both be present in the same request.

`subject_type`

- For cases where correlation of End-User's activities across Clients is not appropriate, the `subject_type` parameter *MUST* be set to `pairwise`. In other cases, the use of `pairwise` is *RECOMMENDED* unless the use of public identifiers is required.

`request_uris`

- Array of `request_uri` values that are pre-registered by the Client for use at the OpenID Provider. Clients that make Authentication Requests using the `request_uri` parameter, *MUST* only do so via pre-registered `request_uri` values.

Section 2 of [OpenID.Dynamic-Registration] lists all Client Metadata values that are used by OpenID Connect. Note that additional parameters are defined in OAuth 2.0 Dynamic Client Registration Protocol ([RFC7591]) can be relevant as well and *MAY* be used.

EXAMPLE 7

An example of a Client registration request:

```
POST /connect/register HTTP/1.1
Content-Type: application/json
Accept: application/json
Host: server.example.com
Authorization: Bearer eyJhbGciOiJSUzI1NiJ9.eyJ ...
```

```
{
  "application_type": "web",
  "redirect_uris":
    ["https://client.example.org/callback",
    "https://client.example.org/callback2"],
  "client_name": "My Example",
  "subject_type": "pairwise",
  "sector_identifier_uri":
    "https://other.example.net/file_of_redirect_uris.json",
  "token_endpoint_auth_method": "client_secret_basic",
  "jwks_uri": "https://client.example.org/my_public_keys.jwks",
  "userinfo_encrypted_response_alg": "RSA1_5",
  "userinfo_encrypted_response_enc": "A128CBC-HS256",
  "contacts": ["mary@example.org"],
}
```

Please refer to Algorithms for more information on eligible cryptographic methods and keys that can be used when registering a Client.

# 6. User Info

The availability, quality and reliability of an individual's identity attributes will vary greatly across jurisdictions and Provider systems. The following recommendations ensure maximum cross-jurisdictional interoperability, while setting Client expectations on the type of data they may acquire.

## 6.1 Claim Interoperability

As per Section 5.1.2 of [OpenID.Core], Claim names *SHOULD* be collision-resistant. It is *RECOMMENDED* to use domain name based URIs as attribute names.

[OpenID.Core] Section 5.1 specifies a list of standard Claims. In a Dutch governmental context, attribute Claims are commonly registered in the BRP (*Basis Registratie Personen*, the Dutch citizen registry), as defined in [LO.GBA]. Note that some of the standard Claims of OpenID Connect do not map directly or correctly with BRP attributes. BRP attributes *SHOULD* be prefered over OpenID Connect claims for attributes. Additionally, usage of, or interoperability with, the ISA$^2$ core vocabularies is *RECOMMENDED*.

## § 6.2 Claims Supported

Discovery requires including the `claims_supported` field, which defines the Claims a Client *MAY* expect to receive for the supported scopes. OpenID Providers *MUST* return Claims on a best effort basis. However, an OpenID Provider asserting it can provide an End-User Claim does not imply that this data is available for all its End-Users: Clients *MUST* be prepared to receive partial data. OpenID Providers *MAY* return Claims outside of the `claims_supported` list, but they *MUST* still ensure that the extra Claims to not violate the privacy policies set out by the trust framework the Provider supports. The OpenID Provider *MUST* ensure to comply with applicable privacy legislation (e.g. informed consent as per GDPR) at all times.

> Note that when Representation is supported, the OpenID Provider *MUST* include `represents` in the list of supported Claims and *MAY* include nested Claims inside the `represents` Claim.

## § 6.3 Scope Profiles

In the interests of data minimization balanced with the requirement to successfully identify the individual signing in to a service, the default OpenID Connect scope profiles to request Claims ([OpenID.Core] Section 5.4) may not be appropriate.

Matching of the identity assertion based on Claims to a local identifier or `account` related to the individual identity at a Level of Assurance is a requirement where the government in question is not able to provide a single identifier for all citizens based on an authoritative register of citizens.

The requirement for matching is also of importance where a cross-border or cross-jurisdiction authentication is required and therefore the availability of a single identifier (e.g. social security number) cannot be guaranteed for the individual wishing to authenticate.

However, in the Netherlands the BSN is, as a common identifier for citizens, available to BSN-eligible organizations. Nationwide interoperable pseudonyms per OpenID Client for non-BSN-eligible organizations exist as well.

The default `profile` scope of OpenID Connect is very wide, which is undesired from a privacy perspective. As such, the `profile` scope *SHOULD NOT* be used.

> Note that the `doc` profile described in the iGov profile for OpenID Connect [OpenID.iGov] is not in common use in the Netherlands and therefore not included in this profile.

## 6.4 Claims Request

OpenID Core Section 5.5 [OpenID.Core] defines a method for a Client to request specific Claims in the UserInfo object or ID Token. OpenID Providers *MUST* support this `claims` parameter in the interest of data minimization - that is, the Provider only returns information on the subject the Client specifically asks for, and does not volunteer additional information about the subject.

Clients requesting the `profile` scope *MAY* provide a `claims` request parameter. If the Claims request is omitted, the OpenID Provider *SHOULD* provide a default Claims set that it has available for the subject, in accordance with any policies set out by the trust framework the Provider supports.

> **Note:** Clients *SHOULD NOT* request the `profile` scope, as described in the previous section.

## 6.5 Claims Response

Response to a UserInfo request *MUST* match the scope and Claims requested to avoid having a OpenID Provider over-expose an End-User's identity information. OpenID Providers *MUST NOT* provide any personal identifiable information without applicable consent.

Claims responses *MAY* also make use of the aggregated and/or distributed Claims structure to refer to the original source of the subject's Claims.

## 6.6 Claims Metadata

Claims Metadata (such as locale or the confidence level the OpenID Provider has in the Claim for the End-User) can be expressed as attributes within the UserInfo object, but are outside the scope of this document. These types of Claims are best described by the trust framework the Clients and OpenID Providers operate within. It is up to the Client to assess the level of confidence provided

by the OpenID Provider or the trust framework, per Claim. Expressing or evaluating such confidence is beyond the scope of this profile.

In order to provide a source, including integrity and optionally confidentiality, an OpenID Provider *SHOULD* be able to provide aggregated or support distributed Claims. The signee of such aggregated or distributed Claims implies the source and can support in assessing the level confidence or quality of the Claim.

# § 7. Considerations

## § 7.1 Privacy considerations

Data minimization is an essential concept in trust frameworks and federations exchanging End-User identity information for government applications. The design of this profile takes into consideration mechanisms to protect the End-User's government identity information and activity from unintentional exposure.

Pairwise Subject identifiers *MUST* be supported by the OpenID Providers for frameworks where subjects should not be traceable or linkable across Clients by their Subject ID. This prevents situations where an End-User may inadvertently be assigned a universal government identifier.

Request Claims using the `claim` parameter *MUST* be supported by OpenID Providers to ensure that only the data the Client explicitly requests is provided in the UserInfo response or ID Token. This prevents situations where a Client may only require a partial set of Claims, but receives (and is therefore exposed to) a full set of Claims. For example, if a Client only needs an identifier and the persons legal age, the OpenID Provider *MUST NOT* send the Client the full user name and birth date. Similarly, broad attribute requests through the `scope` parameter, such as `profile` *SHOULD NOT* be used.

All Clients *MUST* apply the concept of data minimization. As a result, a Client *MUST NOT* request any more identifiers, attributes or other Claims than strictly necessary. Additionally, Clients *SHOULD* ensure they minimize the scope and audience they request, use and forward. This principle applies to both to usage at the Client as well as forwarded Access Tokens in a Service Intermediation scenario. Token Exchange [RFC8693] *SHOULD* be used to request Access Tokens with a minimal scope and audience.

> Note that per-instance registration of Native Clients can increase the risk of Client -- and thus End-User -- observability and traceability. This because the `client_id` is unique, can be linked to an individual and may be observed. The `client_id` *SHOULD* be considered and treated as sensitive data in case per-instance registration is applied. Although the `client_id` will be protected by TLS, it may be exposed at the Client itself or the OpenID Provider or elsewhere. As mitigating measure, implementations *MAY* use encrypted request objects and tokens. OpenID Providers *SHOULD* assign unpredictable Client Identifiers in case of per-instance registration for Native Clients, in order to mitigate guessing and (cross Client and cross audience) linkability of Client Identifiers.

In order to provide end-to-end security and privacy, identifiers and attributes *SHOULD* be encrypted from the providing source to the ultimate intended recipient. This can be accomplished by either encrypting entire response messages and tokens or by using aggregated or distributed Claims (see Section 5.6.2 of [OpenID.Core]). Applying end-to-end encryption is strongly *RECOMMENDED* for both the BSN (*Burgerservicenummer*, the Dutch citizen ID) and sensitive attributes.

Despite the mechanisms enforced by this profile, the operational circumstances may allow these controls to be relaxed in a specific context. For example, if a bilateral agreement between two agencies legally entitles usage of citizen identifiers, then the Pairwise Pseudonymous Identifier requirement may be relaxed. In cases where all Clients are entitled to process Claims associated to a subject at an OpenID Provider, the Claims request requirement may be relaxed.

The reasons for relaxing the controls that support data minimization are outside the scope of this profile.

## § 7.2 Security considerations

Implementations of this profile or any form of access to a service, *MUST* make a risk assessment or security classification for that service and the information disclosed. It is strongly *RECOMMENDED* to follow the guide 'Assurance level for digital service provision' [SG.LoA]. Particularly when implementing for higher levels of assurance (e.g. eIDAS "high" or "substantial"), requirements specified as *SHOULD* (NOT) or (NOT) *RECOMMENDED* in this profile are more pertinent to implement accordingly. In line with the scope of the "Assurance level for digital service provision" guide, information and services classified as "state secret" (Dutch: "*staatsgeheim*") are out of scope for implementations under this profile.

An OpenID Provider *MUST* use a distinct Client Identifier (`client_id`) and registration for each unique Client. This in particular applies to public Clients, these registrations *MUST NOT* be shared with confidential Clients, even if they are operated by the same organisation. Distinct registrations *MAY* be applied to different versions of (native and browser-based public) Clients as well. This will

allow a form of support for version management, noting that this can not be considered a very reliable method from a security point of view.

Refresh Tokens *SHOULD* only be applied and enabled when a functional need exists. Support for Refresh Tokens *SHOULD* therefore be disabled by default. Refresh Tokens for confidential Clients *MUST* be sender-constrained by the issuing OpenID Provider. How the OP accomplishes this is implementation specific, suggestions can be found in [OAuth2.1], Section 6.1. Using Refresh Tokens in combination with public Clients *SHOULD* be avoided when possible. If a specific scenario does call for usage of Refresh Tokens with public Clients, Refresh Tokens *MUST* rotate on each use with a limited valid lifetime.

All transactions *MUST* be protected in transit by TLS as described in BCP195 [RFC7525]. In addition, all compliant implementations *MUST* apply the IT Security Guidelines for TLS by the Dutch NCSC [SG.TLS]. Implementations *SHOULD* only implement settings and options indicated as "good", *SHOULD NOT* use any settings with a status "phase out" and *MUST NOT* use any setting with a status "insufficient" in these security guidelines or future updates thereof.

Implementations *MUST* implement 'HTTP Strict Transport Security', as specified in [RFC6797].

All Clients *MUST* conform to applicable recommendations found in the 'Security Considerations' sections of [RFC6749] and those found in 'OAuth 2.0 Threat Model and Security Considerations' [RFC6819]. For all Tokens, the 'JSON Web Token Best Current Practices' [RFC8725] *SHOULD* be applied.

All Clients *MUST* apply cross-site request forgery (CSRF) counter measures. Clients can leverage the OpenID Connect `nonce` and OAuth2 `state` parameters to do so. A Client *MUST* utilize one or more of these parameters to verify an Authentication Response matches with the Authentication Request sent. After first use, the Client *SHOULD* invalidate the parameter so it can be used only once (see [OAuth2.Security], Section 4.2.4).

In case Clients are relying on and communicating with multiple OpenID Providers (and/or OAuth2 Authorization Servers), Clients *MUST* implement countermeasures to prevent mix-up attacks. Clients *SHOULD* at least use distinct redirect URIs for each OpenID Provider / Authorization Server, or alternatively validate the issuer (`iss`) in the response (ID Token) matches the initiating Authentication Request (see [RFC8252], Section 8.10 and [OAuth2.Security], Section 2.1 and 4.4.2).

### § 7.2.1 Algorithms

Security of OpenID Connect and OAuth 2.0 is significantly based on the application of cryptography. Herein the choice of algorithms is important for both security as well as interoperability. This section lists relevant choices of algorithms for all messages and tokens.

For signing of messages and tokens, implementations:

- *MUST* support `RS256`.
- *SHOULD* support `PS256`; usage of `PS256` is *RECOMMENDED* over `RS256`.
- *MAY* support other algorithms, provided they are at least equally secure as `RS256`.
- *MUST NOT* support algorithms that are less secure than `RS256`.

For asymmetric encryption, in particular encryption of content encryption keys, implementations:

- *MUST* support `RSA-OAEP`.
- *SHOULD* support `RSA-OAEP-256`.
- *MAY* support other algorithms, provided they are at least equally secure as `RSA-OAEP`.
- *MUST NOT* support algorithms that are less secure than `RSA-OAEP`.

For symmetric encryption, implementations:

- *MUST* support `A256GCM`.
- *MAY* support other algorithms, provided they are at least equally secure as `A256GCM`.
- *MUST NOT* support algorithms that are less secure than `A256GCM`.

In addition to proper selection and configuration of algorithms, implementations *MUST* ensure to use a cryptographically secure (pseudo)random generator. Administrators and implementations *MUST* apply industry best practices for key management of cryptographic keys. This includes best practices for selection of applicable key length as applicable for the relevant algorithm(s) selected.

## § 7.3 Future updates

This profile was created using published, finalized specifications and standards as basis. Some relevant new documents are under development at the time of writing. As this profile does not use any draft documents as basis, these cannot be included. However, we want to attend readers to these developments and for them to take into account that future updates to this profile may incorporate the resulting standards and specifications. Furthermore we would like encourage readers to follow relevant developments.

### § 7.3.1 Service Intermediation

One functionality that is widely used in the (semi-)governmental sector but is not included in the initial version of this profile specification is *Service Intermediation*. This scenario is sometimes

also refered to as identity propagation. Examples of Service Intermediation scenario's include portals, API aggregators and Clients with enhanched or automated assistence for consuming services.

Service Intermediation is applicable when the Service Provider does not directly interact with the End-User, but delegates this responsibility to a Service Intermediary. The Service Intermediary therefore interacts with the OpenID Provider for End-User authentication, with the service offered by the Serivce Provider in scope of the Authentication Request. The Service Provider can now rely on a token from the OpenID Provider received via the Service Intermediary. Note that there is interaction with OAuth2, the Service Provider acts as Resource Server.

Such a Service Intermediary can intermediate a single service offered by a single Service Provider (e.g. an accounting app (service) that has an option to submit a tax declaration) or it can aggregate multiple Services offered by multiple Service Providers using intermediation (e.g. an app that aggregates your health information stored at several health organisations).

It is anticipated that support for Service Intermediation will be added in a later version of this profile; when it will, the following should be considered:

- Service Intermediaries should be able to obtain Claims and subject identifiers for different intermediated Services via different interactions with the OpenID Provider, with End-User consent but without the need of complete re-authentication.
- Service Intermediaries are generally not allowed to access Claims and subject identifiers. Hence, the use of pairwise and encrypted subject identifiers and Claims is usually required.
- Service Providers control which Service Intermediaries they support, specifically when confidential information is involved. Hence, Client Registration with the OpenID Provider must be established such that Service Intermediaries can only intermediate (and request Claims and subject identifiers for) Services that they are authorized for. A potential solution direction could be the use of Proof-of-Possession Key Semantics, as described in [RFC7800].

### § 7.3.2 Federations

This profile acknowledges that federations are widely in use, in particular among (semi-)governmental and public domain orgranisations. However, no specific support or requirements for federations are included in this version of this profile. The OpenID Foundation is currently drafting a specification for explicit support of federations using OpenID Connect. Future updates to this profile may adopt such federation specificaitons once finalized. See Federation at the OpenID Foundation.

### 7.3.3 Other features

§

The following overview lists RFC and BCP documents being drafted by the OAuth 2.0 working group of the Internet Engineering Task Force (IETF) and work-in-progress by the OpenID Foundation. Future updates to this profile are likely to seek usage of and interoperability with these specifications once finalized.

[OAuth2.JWT]

- An RFC for Access Tokens in JWT format is being drafted in the OAuth 2.0 working group at IETF.

[OAuth2.JAR]

- An RFC for Secured (signed and/or encrypted) Authorization Requests is being drafted in the OAuth 2.0 working group at IETF. Most of the practices described in this RFC are already part of the OpenID Connect Core specification.

[OAuth2.RAR]

- An RFC that introduces a request parameter `authorization_details`, which allows for more expressive Authentication Requests than those possible with the `scope` parameter, is being drafted in the OAuth 2.0 working group at IETF.

[OAuth2.PAR]

- An RFC that introduces an endpoint to which Clients can push Authorization Requests via a direct POST request to an Authorizaton Server, prior to forwarding the End-User with a `request_uri` referencing the request to the Authorization Server, is being drafted in the OAuth 2.0 working group at IETF. The practices described in this RFC are already part of the OpenID Connect Core specification.

[OAuth2.Security]

- A Best Current Practice document that extends the OAuth 2.0 Security Threat Model and provides security recommendations to address security challenges in OAuth 2.0 is being drafted in the OAuth 2.0 working group at IETF.

[OAuth2.Browser-Based-Apps]

- A Best Current Practice document that details security considerations and best practices to be taken into account when implementing browser-based applications that use OAuth 2.0 is being drafted in the OAuth 2.0 working group at IETF.

[OAuth2.1]

- An effort to consolidate and simplify OAuth 2.0 by adding and removing functionality of the core OAuth 2.0 specification and by incorporating several RFCs and BCPs that were built upon OAuth 2.0.

[OpenID.Federation]

- Work by the OpenID Foundation to support federations of OpenID Providers and relying Service Providers, by publishing aggregated metadata in a specified format.

# 8. Glossary

*This section is non-normative.*

The following terms that are specific to this profile or its functional context are used throughout this specification:

Representation

> The action of one party acting on behalf of another party through delegated authority, which was given voluntary or based on legal grounds. Both parties can either be natural or juridical persons.

Representation Relationship

> When one party represents another party, both parties have a Representation Relationship. Typically a Representation Relationship needs to be formally documented in order to be useable in automated processes, resulting in a statement or registration of the Representation Relationship.

eIDAS

> eIDAS (Electronic Identification, Authentication and Trust Services) is an EU regulation on electronic identification and trust services for electronic transactions in the European Union.

## 8.1 Notices

The technology described in this specification was made available from contributions from various sources, including members of the OpenID Foundation (OIDF), the Internet Engineering Task Force (IETF) and others.

## 8.2 Acknowledgements

Special thanks go to the following people for their involvement in the working group and for their contributions to the specification:

*Anouschka Biekman (Logius), Arjen Monster (Gemeente Den Haag), Arnout van Velzen (Logius/RV), Bart Geesink (Surfnet), Bart Kerver (VWS), Bob te Riele (RVIG), Dennis Reumer (RVO), Dolf Smits (Belastingdienst), Elsbeth Bodde (iShare), Erwin Reinout (Kennisnet), Frank van Es (Logius), Frans de Kok (Logius/ETD), Jan Geert Koops (Dictu), Jan Jaap Zoutendijk (CIV), Jeroen de Ruig (Lost Lemon), Joost van Dijk (Surfnet), Joris Joosten (VZVZ/MedMij), Lakeshia Tjin Liep Shie (Logius), Mark Nijmeijer (Justid), Martin Borgman (Kadaster), Peter Haasnoot (Logius/CvS), Peter Kooi (Belastingdienst), Pieter Heering (Logius), Redouan Ahaloui (Forum Standaardisatie), Remco Schaar (Logius), Rob Post (RVIG), Robin Gelhard (Forum Standaardisatie), Timen Olthof (VNG), and Victor den Bak (iShare).*

# 9. Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words *MAY*, *MUST*, *MUST NOT*, *NOT RECOMMENDED*, *OPTIONAL*, *RECOMMENDED*, *REQUIRED*, *SHALL*, *SHALL NOT*, *SHOULD*, and *SHOULD NOT* in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

# 10. List of Figures

# A. Index

## A.1 Terms defined by this specification

## A.2 Terms defined by reference

# B. References

## B.1 Normative references


**[CORS]**
*Cross-Origin Resource Sharing*. Anne van Kesteren. W3C. 2 June 2020. W3C Recommendation. URL: https://www.w3.org/TR/cors/

**[CSP]**
*Content Security Policy Level 3*. Mike West; Antonio Sartori. W3C. 4 September 2023. W3C Working Draft. URL: https://www.w3.org/TR/CSP3/

**[OAuth2.NLGov]**
*NL GOV Assurance profile for OAuth 2.0*. F. Terpstra; J. van Gelder. Logius. july 2020. URL: https://gitdocumentatie.logius.nl/publicatie/api/oauth/

**[OpenID.Core]**
*OpenID Connect Core 1.0*. N. Sakimura; J. Bradley; M. B. Jones; B. de Medeiros; C. Mortimore. The OpenID Foundation. 2014. URL: https://openid.net/specs/openid-connect-core-1_0.html

**[OpenID.Discovery]**
*OpenID Connect Discovery 1.0*. N. Sakimura; J. Bradley; M. Jones; E. Jay. The OpenID Foundation. 2014. URL: https://openid.net/specs/openid-connect-discovery-1_0.html

**[OpenID.Dynamic-Registration]**
*OpenID Connect Dynamic Client Registration 1.0*. N. Sakimura; J. Bradley; M. Jones. The OpenID Foundation. 2014. URL: https://openid.net/specs/openid-connect-registration-

1_0.html

**[OpenID.iGov]**

*International Government Assurance Profile (iGov) for OpenID Connect 1.0*. M. Varley; P. Grassi. The OpenID Foundation. 2018. URL: https://openid.net/specs/openid-igov-openid-connect-1_0.html

**[RFC2119]**

*Key words for use in RFCs to Indicate Requirement Levels*. S. Bradner. IETF. March 1997. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc2119

**[RFC2616]**

*Hypertext Transfer Protocol -- HTTP/1.1*. R. Fielding; J. Gettys; J. Mogul; H. Frystyk; L. Masinter; P. Leach; T. Berners-Lee. IETF. June 1999. Draft Standard. URL: https://www.rfc-editor.org/rfc/rfc2616

**[RFC3986]**

*Uniform Resource Identifier (URI): Generic Syntax*. T. Berners-Lee; R. Fielding; L. Masinter. IETF. January 2005. Internet Standard. URL: https://www.rfc-editor.org/rfc/rfc3986

**[RFC6749]**

*The OAuth 2.0 Authorization Framework*. D. Hardt, Ed.. IETF. October 2012. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc6749

**[RFC6750]**

*The OAuth 2.0 Authorization Framework: Bearer Token Usage*. M. Jones; D. Hardt. IETF. October 2012. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc6750

**[RFC6797]**

*HTTP Strict Transport Security (HSTS)*. J. Hodges; C. Jackson; A. Barth. IETF. November 2012. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc6797

**[RFC6819]**

*OAuth 2.0 Threat Model and Security Considerations*. T. Lodderstedt, Ed.; M. McGloin; P. Hunt. IETF. January 2013. Informational. URL: https://www.rfc-editor.org/rfc/rfc6819

**[RFC7009]**

*OAuth 2.0 Token Revocation*. T. Lodderstedt, Ed.; S. Dronia; M. Scurtescu. IETF. August 2013. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc7009

**[RFC7234]**

*Hypertext Transfer Protocol (HTTP/1.1): Caching*. R. Fielding, Ed.; M. Nottingham, Ed.; J. Reschke, Ed.. IETF. June 2014. Proposed Standard. URL: https://httpwg.org/specs/rfc7234.html

**[RFC7515]**

*JSON Web Signature (JWS)*. M. Jones; J. Bradley; N. Sakimura. IETF. May 2015. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc7515

**[RFC7516]**

*JSON Web Encryption (JWE)*. M. Jones; J. Hildebrand. IETF. May 2015. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc7516

**[RFC7517]**

*JSON Web Key (JWK)*. M. Jones. IETF. May 2015. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc7517

**[RFC7519]**

*JSON Web Token (JWT)*. M. Jones; J. Bradley; N. Sakimura. IETF. May 2015. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc7519

**[RFC7523]**

*JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants*. M. Jones; B. Campbell; C. Mortimore. IETF. May 2015. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc7523

**[RFC7525]**

*Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*. Y. Sheffer; R. Holz; P. Saint-Andre. IETF. May 2015. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc7525

**[RFC7591]**

*OAuth 2.0 Dynamic Client Registration Protocol*. J. Richer, Ed.; M. Jones; J. Bradley; M. Machulak; P. Hunt. IETF. July 2015. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc7591

**[RFC7592]**

*OAuth 2.0 Dynamic Client Registration Management Protocol*. J. Richer, Ed.; M. Jones; J. Bradley; M. Machulak. IETF. July 2015. Experimental. URL: https://www.rfc-editor.org/rfc/rfc7592

**[RFC7636]**

*Proof Key for Code Exchange by OAuth Public Clients*. N. Sakimura, Ed.; J. Bradley; N. Agarwal. IETF. September 2015. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc7636

**[RFC7662]**

*OAuth 2.0 Token Introspection*. J. Richer, Ed.. IETF. October 2015. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc7662

**[RFC7800]**

*Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)*. M. Jones; J. Bradley; H. Tschofenig. IETF. April 2016. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc7800

**[RFC8174]**

*Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words*. B. Leiba. IETF. May 2017. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc8174

**[RFC8252]**

*OAuth 2.0 for Native Apps*. W. Denniss; J. Bradley. IETF. October 2017. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc8252

**[RFC8414]**
*OAuth 2.0 Authorization Server Metadata*. M. Jones; N. Sakimura; J. Bradley. IETF. June 2018. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc8414

**[RFC8485]**
*Vectors of Trust*. J. Richer, Ed.; L. Johansson. IETF. October 2018. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc8485

**[RFC8693]**
*OAuth 2.0 Token Exchange*. M. Jones; A. Nadalin; B. Campbell, Ed.; J. Bradley; C. Mortimore. IETF. January 2020. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc8693

**[RFC8705]**
*OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens*. B. Campbell; J. Bradley; N. Sakimura; T. Lodderstedt. IETF. February 2020. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc8705

**[RFC8725]**
*JSON Web Token Best Current Practices*. Y. Sheffer; D. Hardt; M. Jones. IETF. February 2020. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc8725

**[SG.LoA]**
*Assurance level for digital service provision*. . The Standardisation Forum (NL). September 2017. URL: https://www.forumstandaardisatie.nl/sites/default/files/BFS/4-basisinformatie/publicaties/Assurance-levels-for-digital-service-provision.pdf

**[SG.TLS]**
*IT Security Guidelines for Transport Layer Security (TLS) v2.1*. . NCSC. 19-01-2021. URL: https://english.ncsc.nl/publications/publications/2021/january/19/it-security-guidelines-for-transport-layer-security-2.1

**[SRI]**
*Subresource Integrity*. Devdatta Akhawe; Frederik Braun; Francois Marier; Joel Weinberger. W3C. 23 June 2016. W3C Recommendation. URL: https://www.w3.org/TR/SRI/

## § B.2 Informative references

**[eIDAS.SAML]**
*eIDAS SAML Message Format*. eIDAS Coorperation Network. URL: https://ec.europa.eu/digital-building-blocks/wikis/download/attachments/467109280/eIDAS%20SAML%20Message%20Format%20v.1.2%20Final.pdf

**[LO.GBA]**
*Logisch ontwerp BRP*. . RvIG. July 2023. URL: https://www.rvig.nl/logisch-ontwerp-brp

**[OAuth2.1]**

*OAuth 2.1 Working draft*. D. Hardt; A. Parecki; T. Lodderstedt. IETF OAuth Working Group. April 2020. URL: https://tools.ietf.org/html/draft-parecki-oauth-v2-1

**[OAuth2.Browser-Based-Apps]**

*OAuth 2.0 for Browser-Based Apps*. A. Parecki; D. Waite. IETF OAuth Working Group. Internet-Draft. URL: https://tools.ietf.org/html/draft-ietf-oauth-browser-based-apps

**[OAuth2.JAR]**

*The OAuth 2.0 Authorization Framework: JWT Secured Authorization Request (JAR)*. N. Sakimura; J. Bradley. IETF OAuth Working Group. Internet-Draft. URL: https://tools.ietf.org/html/draft-ietf-oauth-jwsreq

**[OAuth2.JWT]**

*JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens*. V. Bertocci. IETF OAuth Working Group. Internet-Draft. URL: https://tools.ietf.org/html/draft-ietf-oauth-access-token-jwt

**[OAuth2.PAR]**

*OAuth 2.0 Pushed Authorization Requests*. T. Lodderstedt; B. Campbell; N. Sakimura; D. Tonge; F. Skokan. IETF OAuth Working Group. Internet-Draft. URL: https://tools.ietf.org/html/draft-lodderstedt-oauth-par

**[OAuth2.RAR]**

*OAuth 2.0 Rich Authorization Requests*. T. Lodderstedt; J. Richer; B. Campbell. IETF OAuth Working Group. Internet-Draft. URL: https://tools.ietf.org/html/draft-lodderstedt-oauth-rar

**[OAuth2.Security]**

*OAuth 2.0 Security Best Current Practice*. T. Lodderstedt; J. Bradley; A. Labunets; D. Fett. IETF OAuth Working Group. Internet-Draft. URL: https://tools.ietf.org/html/draft-ietf-oauth-security-topics

**[OpenID.Federation]**

*OpenID Connect Federation 1.0 - draft 12*. R. Hedberg; M. Jones; A. Solberg; S. Gulliksson; J. Bradley. June 30, 2020. URL: https://openid.net/specs/openid-connect-federation-1_0-12.html

**[OpenID.NLGov]**

*NL GOV Assurance profile for OpenID Connect 1.0*. R. Schaar; F. van Es; J. Joosten; J. G. Koops. Logius. 2021. URL: https://logius.gitlab.io/oidc/

↥