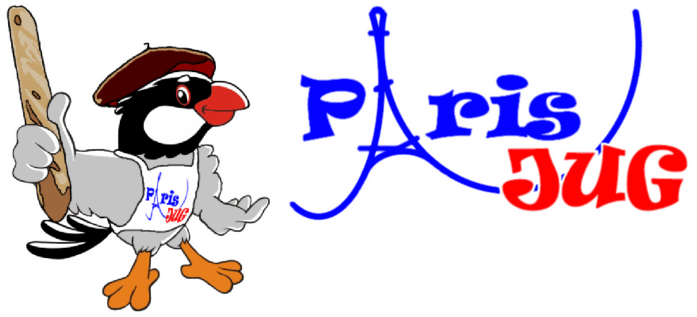
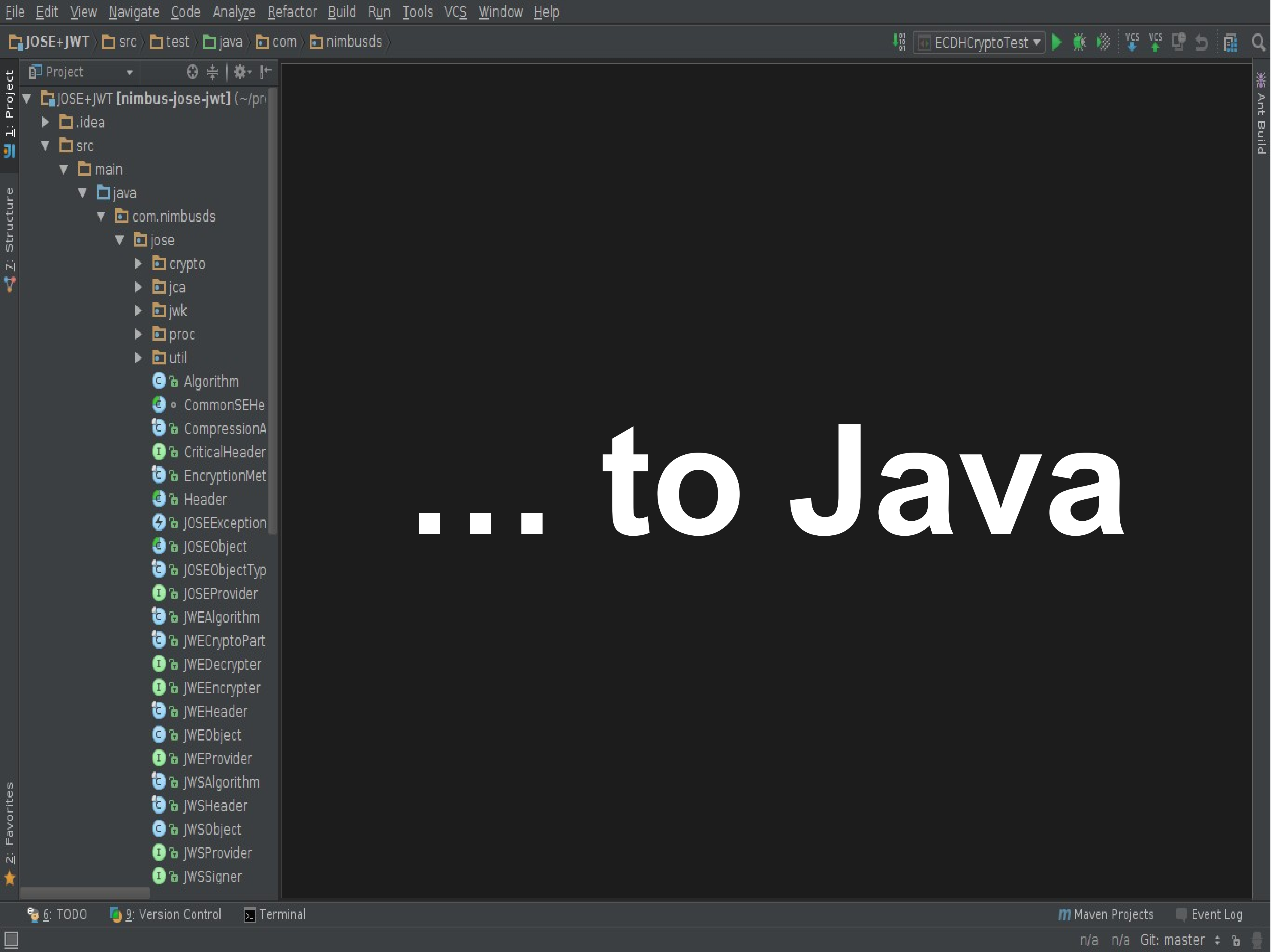
OpenID Connect explained



Vladimir Dzhuvinov

Email: vladimir@dzhuvinov.com : Twitter: @dzhivinov

Married for 15 years


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
JOSE+JWT › src › test › java › com › nimbusds
ECDHCryptoTest
Project
JOSE+JWT [nimbus-jose-jwt] (~/pr
.idea
src
main
java
com.nimbusds
jose
crypto
jca
jwk
proc
util
Algorithm
CommonSEHe
CompressionA
CriticalHeader
EncryptionMet
Header
JOSEException
JOSEObject
JOSEObjectTyp
JOSEProvider
JWEAlgorithm
JWECryptoPart
JWEDecrypter
JWEEncrypter
JWEHeader
JWEObject
JWEProvider
JWSAlgorithm
JWSHeader
JWSObject
JWSProvider
JWSSigner
1: Project
Z: Structure
2: Favorites
Ant Build
6: TODO
9: Version Control
Terminal
Maven Projects
Event Log
n/a n/a Git: master

… to Java

C

Python

JavaScript

JavaScript on a bad day

So what is OpenID Connect?

OpenID Connect is a new internet standard for

Single Sign-On (SSO)

Identity Provision (IdP)

OpenID Connect supports

web clients

mobile / native clients

OpenID Connect is good for

consumer apps

social apps

enterprise apps

mobile apps

OpenID Connect is backed by

Google

Microsoft

eBay
PayPal

Aol

Salesforce

… us and
many others

OpenID Connect distilled

1. Need to authenticate user?

2. Send user to OpenID provider
   (via browser / HTTP 302 redirect)

3. Get Identity (ID) token back

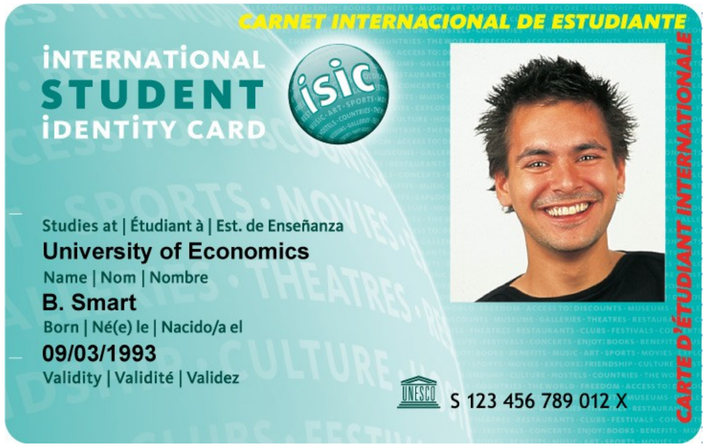
The key OpenID Connect artefact

ID Token

asserts the user's identity (user ID)

Client apps receive an ID token from the OpenID Provider

ID token



CARNET INTERNACIONAL DE ESTUDIANTE

INTERNATIONAL STUDENT IDENTITY CARD

isic

Studies at | Étudiant à | Est. de Enseñanza

University of Economics

Name | Nom | Nombre

B. Smart

Born | Né(e) le | Nacido/a el

09/03/1993

Validity | Validité | Validez

UNESCO

S 123 456 789 012 X

CARTE D'ÉTUDIANT INTERNATIONALE

Resembles the concept of an identity card, in a standard digital format that can be verified by clients.

- Asserts the user's identity.
- Specifies the issuing authority (the IdP).
- May specify how (strength) and when the user was authenticated.
- Is generated for a particular audience (client).
- Has an issue and an expiration date.
- May contain additional subject details such as the user's name, email address and other profile information.
- Is digitally signed, so it can be verified by the intended recipients.
- May optionally be encrypted for confidentiality.

ID token internals

```
{
   "iss"        : "https://c2id.com",
   "sub"        : "alice",
   "aud"        : "s6BhdRkqt3",
   "nonce"      : "n-0S6_WzA2Mj",
   "exp"        : 1311281970,
   "iat"        : 1311280970,
   "auth_time"  : 1311280969,
   "acr"        : "c2id.acr.hisec",
   "amr"        : [ "pwd", "otp" ]
 }
```

- Encoded as a JSON Web Token (JWT).
- The claims about the subject are packaged in a simple JSON object.
- Digitally signed typically with the provider's private RSA key or a shared secret (HMAC) issued to the client during registration.
- Is URL-safe.

Encoded ID token

```
eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ.ewogImlzc
yI6ICJodHRwOi8vc2VydmVyLmV4YW1wbGUuY29tIiwKICJzdWIiOiAiMjQ4Mjg5
NzYxMDAxIiwKICJhdWQiOiAiczZCaGRSa3F0MyIsCiAibm9uY2UiOiAibi0wUzZ
fV3pBMk1qIiwKICJleHAiOiAxMzExMjgxOTcwLAogImlhdCI6IDEzMTEyODA5Nz
AKfQ.ggW8hZ1EuVLuxNuuIJKX_V8a_OMXzR0EHR9R6jgdqrOOF4daGU96Sr_P6q
Jp6IcmD3HP99Obi1PRs-cwh3LO-p146waJ8IhehcwL7F09JdijmBqkvPeB2T9CJ
NqeGpe-gccMg4vfKjkM8FcGvnzZUN4_KSP0aAp1tOJ1zZwgjxqGByKHiOtX7Tpd
QyHE5lcMiKPXfEIQILVq0pc_E2DzL7emopWoaoZTF_m0_N0YzFC6g6EJbOEoRoS
K5hoDalrcvRYLSrQAZZKflyuVCyixEoV9GfNQC3_osjzw2PAithfubEEBLuVVk4
XUVrWOLrLl0nx7RkKU8NXNHq-rvKMzqg
```

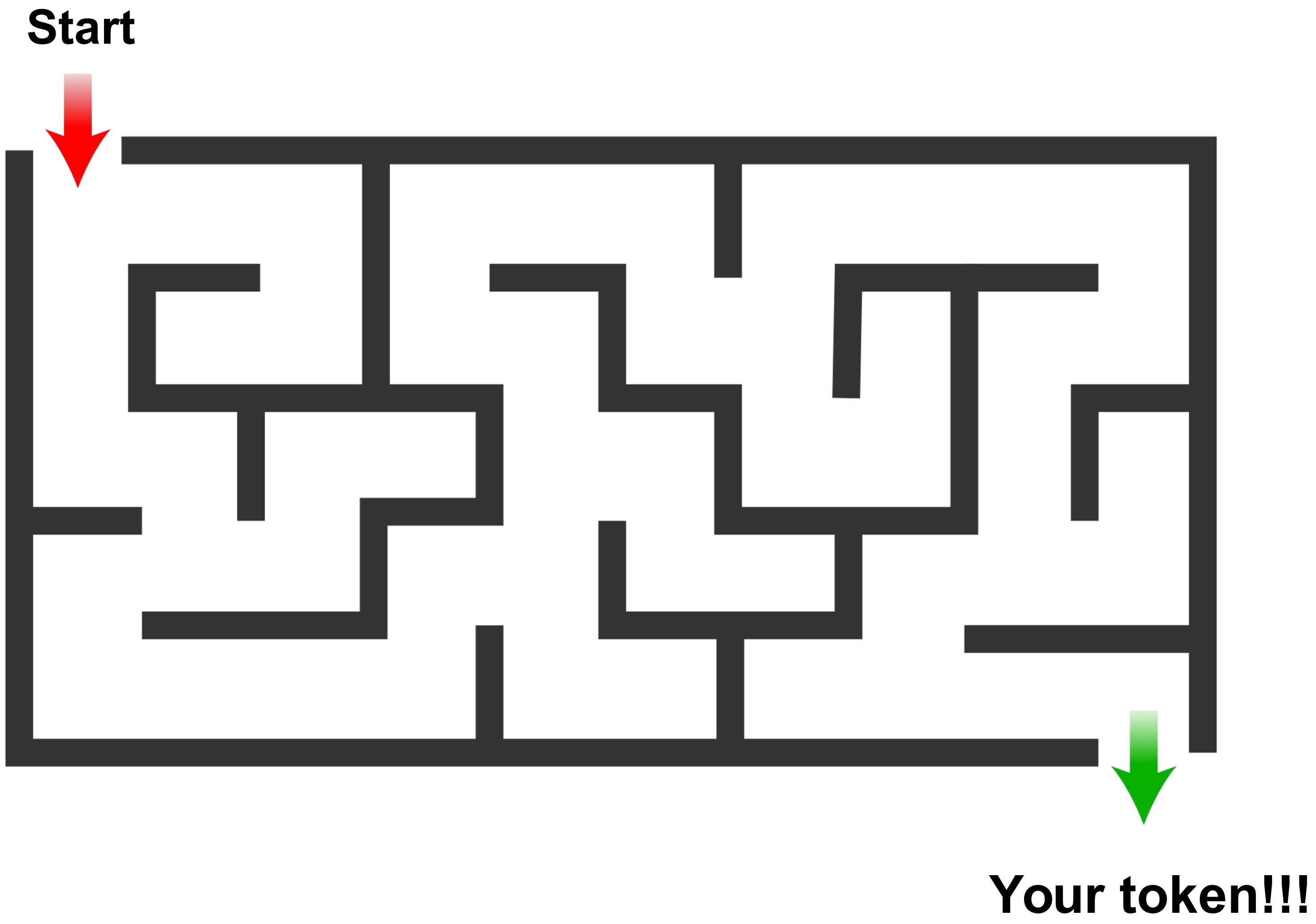
Cool ID token uses

- Simple stateless session management – no need to store sessions in memory / on disk
- May be passed to 3rd parties to assert the user's identity
- May be exchanged for an access token at the token endpoint of an OAuth 2.0 authorisation server. This feature has uses in distributed and enterprise applications. See RFC 7523.

How to obtain an ID token?

Using the OAuth 2.0 protocol flows

The OAuth 2.0 flows



Choose your flow

- **Authorisation code flow**
  - for typical web and mobile apps
  - the client is authenticated
  - tokens retrieved via backchannel
- **Implicit flow**
  - for JavaScript applications that run in the browser
  - the client is **not** authenticated
  - tokens returned via front-channel, revealed to browser
- **Hybrid flow** -
  - allows app front-end and back-end to receive tokens independently
  - rarely used

http://openid.net/specs/openid-connect-core-1_0.html#Authentication

The OpenID auth request (code flow)

Send user to OpenID provider with auth request:

```
https://openid.provider.com/authorize?
    response_type=code
    &scope=openid
    &client_id=s6BhdRkqt3
    &state=af0ifjsldkj
    &redirect_uri=https%3A%2 %2Fclient.example.org%2Fcb
```

The OpenID auth response (code flow)

On successful auth the OpenID provider will redirect the browser back to the client with an authorisation code:

```
https://client.example.org/cb?
    code=SplxlOBeZQQYbYS6WxSbIA
    &state=af0ifjsldkj
```

The OpenID auth response (code flow)

If authentication failed the OpenID provider may return an error code:

```
https://client.example.org/cb?
    error=access_denied
    &state=af0ifjsldkj
```

Exchange code for ID token (code flow)

Client makes back channel request to exchange code for ID token. Note that the client authenticates itself to the server here!

```
POST /token HTTP/1.1
Host: openid.provider.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

grant_type=authorization_code
 &code=SplxlOBeZQQYbYS6WxSbIA
 &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
```

Exchange code for ID token (code flow)

Finally, we get our ID token! But what's this access token?

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{
   "access_token": "SlAV32hkKG",
   "token_type": "Bearer",
   "refresh_token": "8xLOxBtZp8",
   "expires_in": 3600,
   "id_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazc..."
}
```

UserInfo

```
{
   "sub"            : "alice",
   "name"           : "Alice Adams",
   "given_name"     : "Alice",
   "family_name"    : "Adams",
   "email"          : "alice@wonderland.net",
   "email_verified" : true,
   "phone_number"   : "+359 (99) 100200305",
   "profile"        : "https://c2id.com/users/alice",
   "ldap_groups"    : [ "audit", "admin" ]
 }
```

OpenID Connect defines an extensible JSON schema for releasing consented user details to client applications

Requesting UserInfo with the OpenID auth request

Send user to OpenID provider with auth request:

```
https://openid.provider.com/authorize?
    response_type=code
    &scope=openid%20profile%20email
    &client_id=s6BhdRkqt3
    &state=af0ifjsldkj
    &redirect_uri=https%3A%2 %2Fclient.example.org%2Fcb
```

Access token

Resembles the concept of a physical token or ticket. Permits the bearer access to a specific resource or service. Has typically an expiration associated with it.

- OAuth 2.0 access tokens are employed in OpenID Connect to allow the client application to retrieve consented user details from a UserInfo endpoint.
- The server may extend the access token scope to allow the client access to other protected resources and web APIs.
- The client treats the access token as simple opaque string to be passed with the HTTP request to the protected resource.

UserInfo request with access token

Simply include the token in the authorisation header using the Bearer schema (RFC 6750).

```
GET /userinfo HTTP/1.1
Host: server.example.com
Authorization: Bearer SlAV32hkKG
```

UserInfo response

The response from the UserInfo endpoint, containing the consented details (claims / assertions) about the end-user:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "sub": "248289761001",
  "name": "Jane Doe",
  "given_name": "Jane",
  "family_name": "Doe",
  "preferred_username": "j.doe",
  "email": "janedoe@example.com",
  "picture": "http://example.com/janedoe/me.jpg"
}
```

The 2 key OpenID Connect artefacts

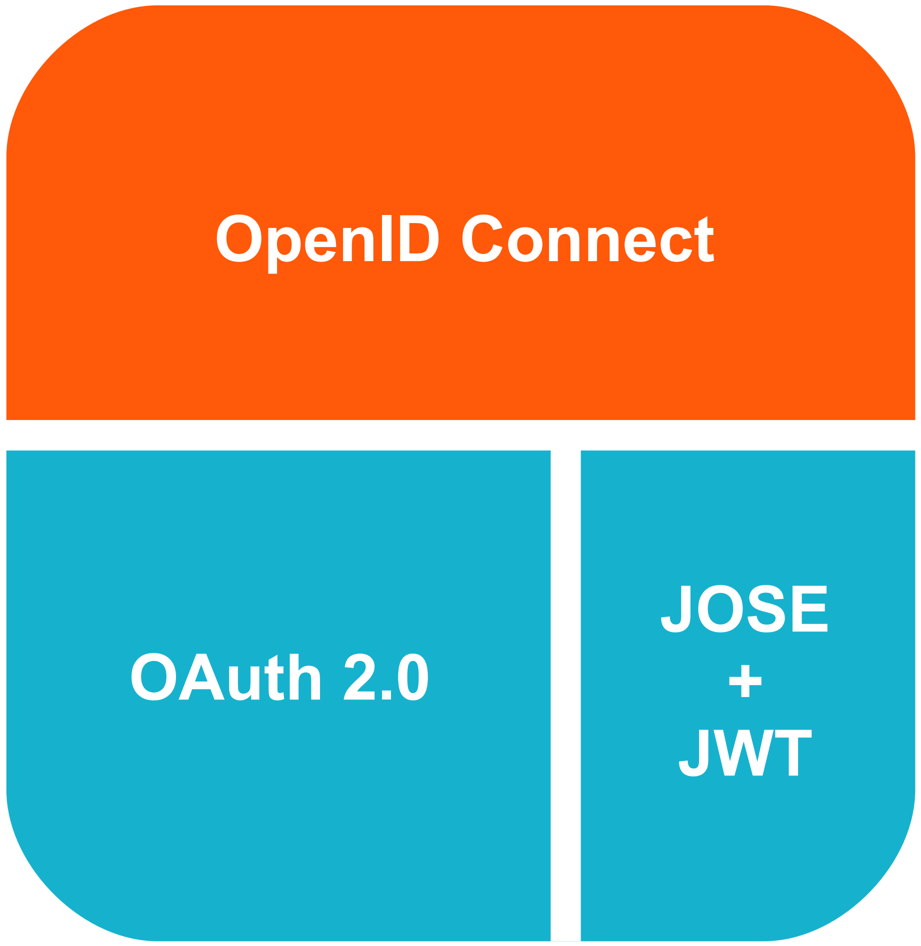
ID Token

asserts the user's identity (user ID)

Access Token

optional, to retrieve consented UserInfo

The OpenID Connect framework



- User identity is asserted by means of JSON Web Tokens (JWT)
- Clients use standard OAuth 2.0 flows to obtain ID tokens
- Mantra: Simple clients, complexity absorbed by the server
- Any method for authenticating users – LDAP, tokens, biometrics, etc.
- JSON schema for UserInfo
- Supports optional provider discovery, dynamic client registration and session management.
- Extensible to suit many use cases.
- Federation is possible.

OpenID Connect provider endpoints

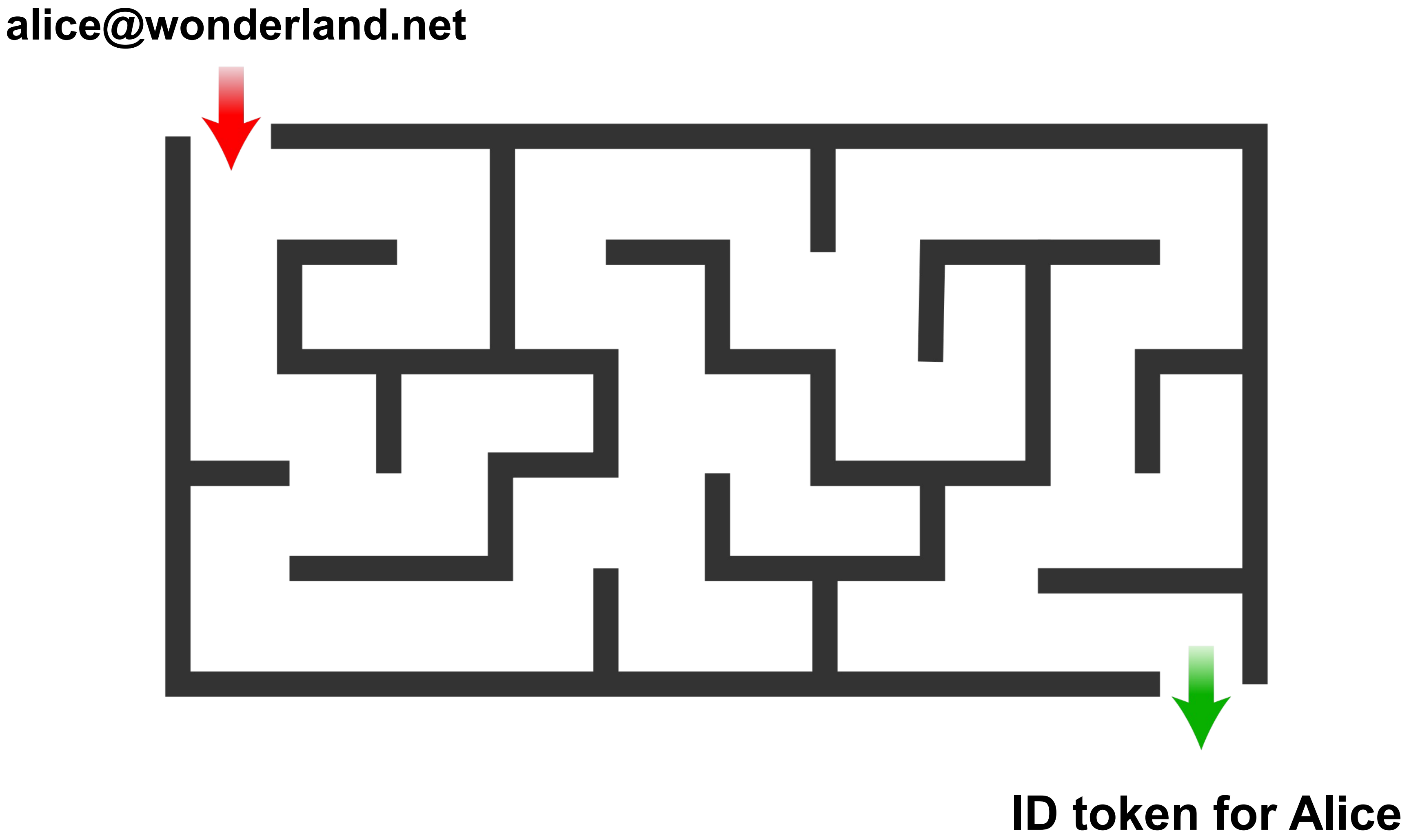
HTTP Endpoints

- Core provider endpoints:
  - Authorisation endpoint
  - Token endpoint
  - UserInfo endpoint
- Optional provider endpoints:
  - WebFinger endpoint
  - Provider metadata URI
  - Provider JWK set URI
  - Client registration endpoint
  - Session management endpoint

Optional endpoints

- WebFinger: enables dynamic discovery of the OpenID Connect provider for a user based on their email address.
- Provider configuration URI: well-known URI returning endpoint and other provider information such as optional capabilities; the client applications can use it to configure their OpenID Connect requests to the provider.
- Provider JWK set URI: JSON document containing the provider's public (typically RSA) keys in JSON Web Key (JWK) format; these keys are used to secure the issued ID tokens and other artefacts.
- Client registration: enables client applications to register dynamically, then update their details or unregister; registration may be open (public).
- Session management: enables client applications to check if a logged in user has still an active session with the OpenID Connect provider; also to signal logout.

The future: dynamic discovery + client registration



The specifications

- OpenID Connect: http://openid.net/connect
- OAuth 2.0 (RFC 6749): http://tools.ietf.org/html/rfc6749
- OAuth 2.0 Bearer token (RFC 6750): http://tools.ietf.org/html/rfc6750
- JSON Web Token: http://tools.ietf.org/html/rfc7519
- JSON Web Signature: http://tools.ietf.org/html/rfc7515
- JSON Web Encryption: http://tools.ietf.org/html/rfc7516
- JSON Web Key: http://tools.ietf.org/html/rfc7517

Thank You!

Q + A