Association pour un système d'information sur les marchés publics en Suisse
Association for an information system on public procurement in Switzerland
Associazione per un sistema informativo sulle commesse pubbliche in Svizzera

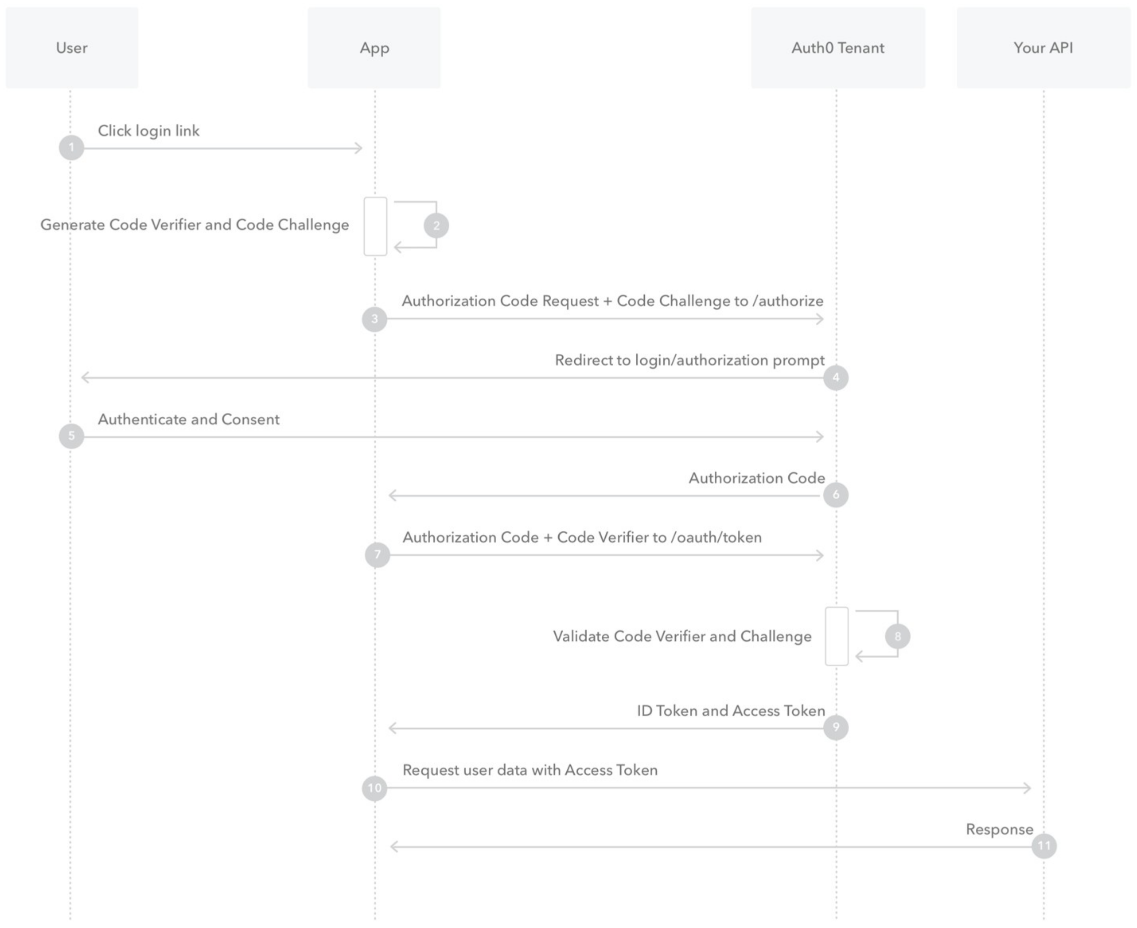
Holzikofenweg 36· CH-3003 Bern· +41 58 480 01 62· office@simap.ch

Quick guide Authorisation Code Flow simap.ch

Summary with diagram

The diagram illustrates the flow:

1. User interaction → Authorisation request with PKCE.
2. User logs in (with 2FA).
3. Authorisation code is forwarded to the app.
4. The app exchanges the code for an access token.
5. The access token is used to access protected resources.


User
App
Auth0 Tenant
Your API
1 Click login link
2 Generate Code Verifier and Code Challenge
3 Authorization Code Request + Code Challenge to /authorize
4 Redirect to login/authorization prompt
5 Authenticate and Consent
6 Authorization Code
7 Authorization Code + Code Verifier to /oauth/token
8 Validate Code Verifier and Challenge
9 ID Token and Access Token
10 Request user data with Access Token
11 Response

Source: Call Your API Using the Authorisation Code Flow with PKCE

Prerequisites

You will need your own registered Auth Client on Simap. This can be requested in the registration form.

OpenID Connect (OIDC) is used for authorisation.

The endpoints can be found via OpenID Connect Discovery:

https://www.simap.ch/auth/realms/simap/.well-known/openid-configuration

- Authorisation Endpoint: https://www.simap.ch/auth/realms/simap/protocol/openid-connect/auth
- Token Endpoint: https://www.simap.ch/auth/realms/simap/protocol/openid-connect/token

Step 1: Generate PKCE values

A PKCE code must be generated in the application for the authorisation request.

PKCE protects the Authorisation Code Flow. You need:

- code_verifier: A random, base64-url-encoded string between 43 and 128 characters.
- code_challenge: A SHA-256 hash of the code_verifier, also Base64-url encoded.

JavaScript example:

```
// Dependency: Node.js crypto module
// https://nodejs.org/api/crypto.html#crypto_crypto
function base64URLEncode(str) {
    return str.toString('base64')
        .replace(/\+/g, '-')
        .replace(/\//g, '_')
        .replace(/=/g, '');
}
var verifier = base64URLEncode(crypto.randomBytes(80));
```

Source: https://www.oauth.com/oauth2-servers/pkce/authorization-request/

Step 2: Authorisation Request

The user initiates the authentication process in the application,

The app directs the user to the **Authorisation Endpoint** to request the Authorisation Code.

Parameters:

- response_type=code: Requests an authorisation code.
- client_id={cient_id}: The ID of the issued client.
- redirect_uri={redirect_uri}: One of the configured redirect URIs of the client, URL encoded. The user is redirected there after login. This is mandatory for Simap.
- code_challenge={code_challenge}: The previously generated code challenge.
- code_challenge_method=S256: Specifies that SHA-256 is used for PKCE.
- scope=openid,profile: The requests scope

Example URL:

```
https://www.simap.ch/auth/realms/simap/protocol/openid-connect/auth
?response_type=code
&client_id={client_id}
&redirect_uri={redirect_uri}
&code_challenge={code_challenge}
&code_challenge_method=S256
&scope=openid%20profile
```

The user is redirected to the login page.

Step 4: Redirect to login

- The user is prompted to log in to the authentication service.
- A code is sent by e-mail as part of the 2-FA and the user enters it.

Step 5: User gives consent

- After successful authentication, the user is redirected to the specified *redirect_uri*, which contains the authorisation code.

Example redirect:

https://yourapp.com/callback?code=abc123&state=xyz123

Step 6: Use authorisation code

The app uses the authorisation code to obtain an access token. This is done via a **POST request** to the token endpoint.

Example request:

```
POST https://www.simap.ch/auth/realms/simap/protocol/openid-connect/token
Content-Type: application/x-www-form-urlencoded
grant_type=authorization_code
client_id= {client_id}
redirect_uri={redirect_uri}
code={auth_code}
code_verifier= {original_code_verifier}
```

Step 7: Validation

The authentication server validates the *code_verifier* against the originally transmitted *code_challenge*.

Step 8-9: Receive token

If the validation is successful:

- An access token and an ID token are returned.

Example answer:

```
{
  "access_token": "eyJhbGciOiJIUzI1...",
  "id_token": "eyJhbGciOiJIUzI1...",
  "expires_in": 3600,
  "token_type": "Bearer",
  "scope": "openid",
 "refresh_expires_in" : 1763,
 "refresh_token" : "eyJhbGciOiJIU…”
}
```

Step 10-11: API call

The app uses the received access_token to call protected API endpoints.

Example request:

```
GET https://api.simap.ch/protected-resource
Authorisation: Bearer eyJhbGciOiJIUzI1...
```

Using the Refresh Token

With the returned access_token there is also the refresh_token.

By design the access token has a short lifespan. With the issued refresh token a new access token can be requested.

Example request:

```
POST https://www.simap.ch/auth/realms/simap/protocol/openid-connect/token
Content-Type: application/x-www-form-urlencoded
grant_type=refresh_token
client_id={client_id}
scope=openid%20profile
refresh_token={refresh_token}
```

Further links:

- https://www.oauth.com/
- Call Your API Using the Authorisation Code Flow with PKCE
- Refreshing Access Tokens
- API for simap.ch (PROD) and INT