Getting started

Before you start

Before you can begin integrating with BankID OIDC platform, you need to make sure you have:

- Contacted one of our partners in order to acquire necessary client credentials (i.e. client_id and client_secret) to our test environment.
- A web application with server side components (as pure frontend applications are not supported).

We strongly recommend that you make use of an OpenID Connect client library applicable for your platform.

You can see some examples below, and you can also refer to the officially certified implementations.

Framework / Language	Examples
Node.js	openid-client: https://www.npmjs.com/package/openid-client
C# (ASP.NET / ASP.NET Core)	Identity server 4: https://identitymodel.readthedocs.io/en/latest/index.html BankID OIDC example with .NET Core 2.0: https://github.com/vippsas/bankid-oidc-example-aspnetcore2
Java	Nimbus OAuth 2 with OpenID Connect extensions: https://connect2id.com/products/nimbus-oauth-openid-connect-sdk BankID OIDC Example Java implementation: https://github.com/vippsas/bankid-oidc-example-java

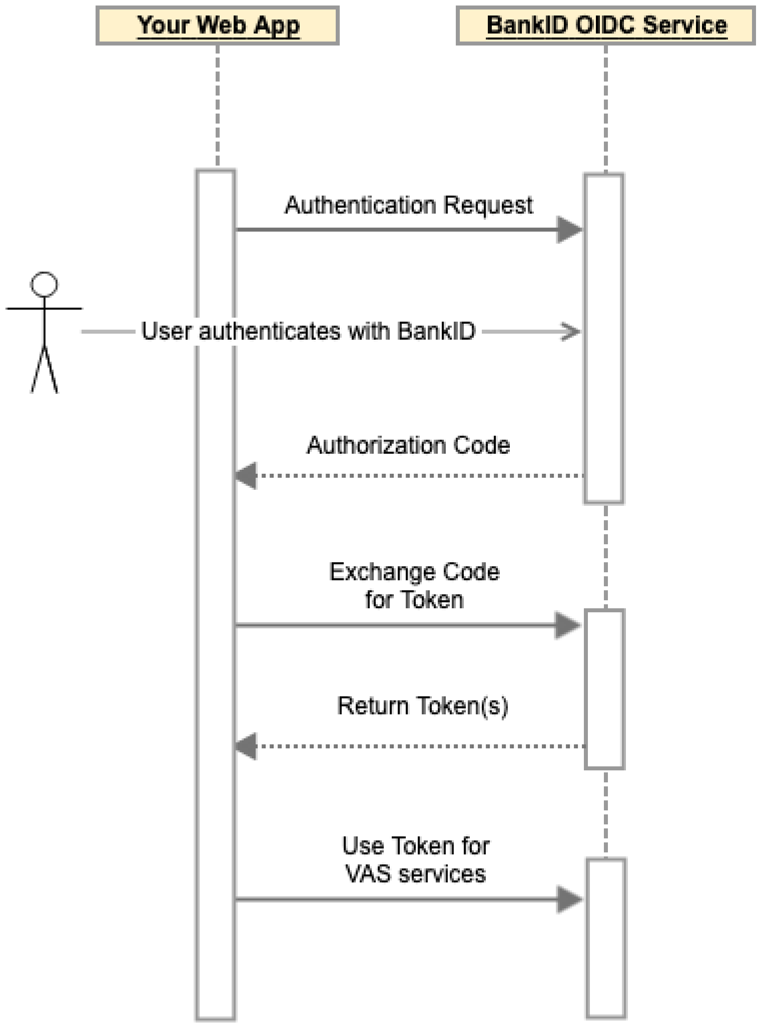
OpenID Connect at a glance

OpenID connect is a protocol for authentication that builds on the OAuth 2.0 authorization framework. This getting started guide describes how to implement BankID OIDC using the authorization code flow.

Note that implicit flow is **not** supported as recommended by IETF (https://tools.ietf.org/html/draft-ietf-oauth-security-topics-13).

The steps necessary to perform an OpenID Connect authentication request is:

1. Redirect the end-user to the authorization endpoint with applicable parameters.
2. The end user authenticates using BankID and is redirected back to your web application.
3. Handle the redirect request by checking parameters and handle potential errors.
4. Exchange the code parameter for tokens by making a request to the token endpoint.
    a. Use the ID token to identify the user.
    b. Use the Access token to access additional Value-Added services provided by the BankID OIDC Service.
    c. Use the Refresh token to retrieve new access tokens.



Implementation

Initiate authentication request

1. Connect to the appropriate OIDC discovery endpoint using your preferred framework or store the JSON response in your application.
   For the CURRENT environment: https://oidc-current.bankidapis.no/auth/realms/current/.well-known/openid-configuration.

2. Use the `authorization_endpoint` found in the OIDC configuration.
   Note: Do not hard code the `authorization_endpoint` value. Always retrieve it from the OpenID Connect configuration as it may change.

3. Generate a random string value and store it in a variable called `state`.
   Important: The value must be *non-guessable* (e.g. GUID) and *unique* for each request.
   This value will be used to mitigate cross-site request forgery, but can also be used to link the callback to the original request.

4. Generate a random string value and store it in a variable called `nonce`.
   Important: The value must be non-guessable, cryptographically random (your framework/language most likely have support for this) and unique for each request.
   This will be used to verify integrity of ID token and mitigate replay attacks.

5. Store the `state` and `nonce` value in your application so they can be retrieved later, i.e. using your preferred session mechanism. These values should be stored together.

6. Build the authorization URL by adding the following query parameters (see Authorize for more options and details):
   - `client_id`: Unique client identifier provided when provisioning the client.
   - `scope`: Specify which information (see ID token) and resources (see Value Added Services) you request access to. In the example, we use "openid" and "profile" to get a regular ID token.
   - `redirect_uri`: URI to your server-side callback endpoint where you want to receive response from the BankID OIDC service.

- `response_type`: Determines message flow. Only "**code**" is supported.
- `state`: Your generated value for state.
- `nonce`: Your generated value for nonce.

Example authorization request

```
GET authorization_endpoint
 ?client_id=myclient-bankid-current
 &scope=openid+profile
 &redirect_uri=https%3A%2F%2Fmywebapp.example.org%2Fcallback
 &response_type=code
 &state=01e3ac8e-4a26-4dfb-79ca-2631394c4144
 &nonce=1fb72f68-1bea-2ba2-12d7-24df1c999d1b
```

7. Redirect end-user to the completed authorization URL.

Handle callback from BankID OIDC

1. Create an endpoint in your web application's backend for receiving the callback from the application (i.e. a GET request to your redirect_uri).

2. Check that the `state` query parameter returned corresponds to the `state` parameter you created when initiating the request.
   a. If the `state` is missing or is not matching a `state` value stored in your application, reject the request.
   b. If the `state` match a `state` stored in your application, update the `state` from in your session mechanism and continue to the next step.

3. Check if the request contains the response parameter `error`.
   a. If the `error` parameter is not present, continue to the next step.
   b. If the `error` parameter is present, you need to handle the error.
      - The possible ASCII values for the `error` query parameter is defined in the specification for OIDC (§3.1.4.6) or OAuth 2.0 (RFC 6749, §4.1.2.1).
      - If the end-user cancels an authentication request in the BankID OIDC client, `error` will have the value `access_denied`.
      - In addition to `error`, the response parameter `error_description` might be available, but it is only intended for developers and not to be shared with the end-user.

4. Retrieve the ID token, Access token and Refresh token. by sending a POST request to the token endpoint.
   a. Verify that the `nonce` claim in the received ID token corresponds to the `nonce` value you belonging to the same session (i.e. the nonce belonging to the state).
      i. If the `nonce` is missing or does not match the nonce stored in your application, reject the request and do not continue.
      ii. If the `nonce` match the `nonce` stored in your application, delete the `state` and `nonce` in your session mechanism and continue to the next step.

5. The ID token contains claims that can be used to identify the end-user. See ID token for available claims.

6. The Access token can be used to access Value Added Services such as document signing or additional user information

Next steps

Depending on your needs you may want to read more about of our Value Added Services:

- Document and text signing
- Additional User Information (TINFO)
- Anti-money laundering (AML)
- Fraud data

If you are interested in a more detailed explanation of the authorization code flow for the OIDC provider from BankID OIDC, you should read Message flow details.

It is also recommended to take a look at Session handling to get a better understanding of the session lifetime.

Known issues contains a list of restrictions, caveats and known problems.

The list of error codes displayed to the end-user is useful for troubleshooting.