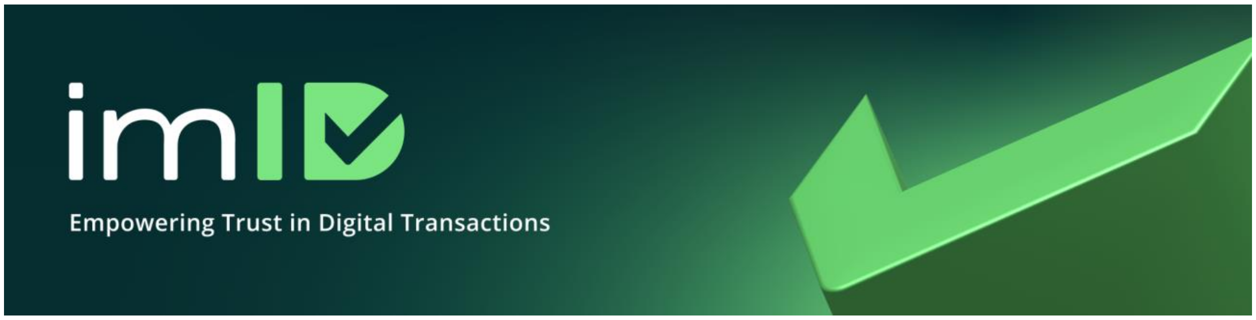
Tel: +374 55 249760
E-mail: enterprise@imid.am
Azatutyun Ave 24/1



Identification as a Service (IDaaS)

Accessing the imID API via OIDC Authentication

API Integration & Specifications Document

March 2025

V 2.2

Tel: +374 55 249760
E-mail: enterprise@imid.am
Azatutyun Ave 24/1

Tel: +374 55 249760
E-mail: enterprise@imid.am
Azatutyun Ave 24/1

Overview

This document provides a comprehensive guide for Relying Parties to integrate with the imID API using an OpenID Connect (OIDC) protocol. It outlines the identification process, integration steps, and technical requirements.

Additionally, the document provides OpenAPI documentation for seamless implementation, detailing available endpoints, authentication methods, and response structures.

Relying Parties integrating with the imID API must adhere to product branding guidelines, including the use of approved button styles, colors, and user-facing text requirements.

What is Identification as a Service (IDaaS)?

imID offers Identification as a Service to facilitate seamless identification for RA citizens using SIM and eSIM Mobile IDs (imID). Users can easily be identified through their imID by simply using their PIN. This service simplifies digital identity verification across various platforms, enhancing security and user experience.

Definitions

IDaaS: Identification as a Service

imID: Mobile ID

Relying Party: any organization or business that integrates imID to its entity (such as a website or application) for user authentication.

IDP: Identity Provider, which is imID CJSC

Users: end users, such as citizens, who will be identified through imID.

IDaaS User Journey

Example: Onboarding a user to a bank application using imID.

In the example below, we walk you through how a citizen uses imID for seamless identification.

Scenario:

You are a bank that wants to onboard users and make them clients without requiring them to visit a physical branch.

Precondition:

The user must have an **activated imID**.

User Journey:

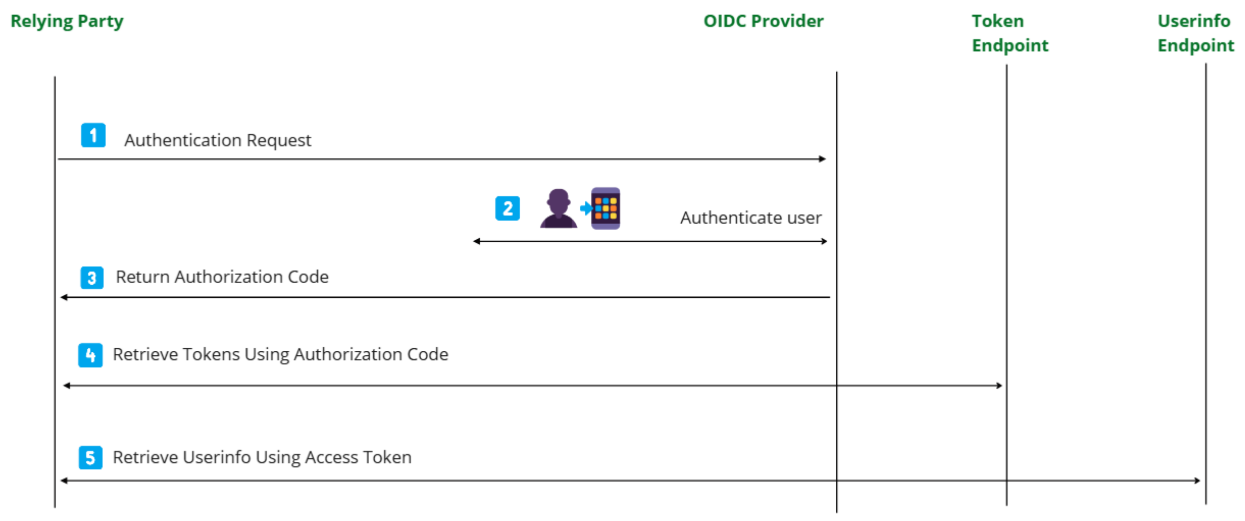
1. **User Initial Interaction:**
    a. A user opens the bank's application, where a **"Register with imID"** button is visible.
2. **Entering Phone Number:**
    a. Upon clicking the button, the user is prompted to enter their **phone number** associated with an activated imID.
3. **Identity Verification:**
    a. The bank receives a verified user identity through imID and retrieves the necessary information for onboarding.
4. **End of Interaction:**
    a. The user’s identity is confirmed, allowing the bank to proceed with the registration process without requiring a physical visit.

The cases when users must be identified should be defined by a Relying Party.

Authentication Request Flow

1. Relying Party sends request to Identity Provider.
2. IDP will send PIN approval request to user. By entering PIN Code (6 digits) and pressing OK on mobile phone user will give consent to request. Response from mobile phone is sent back to IDP.
3. IDP will send Access Token to Relying Party (based on integration flow type).
4. Relying Party will request user info by providing the received Access Token.

OIDC Authorization Code Flow


Relying Party
OIDC Provider
Token Endpoint
Userinfo Endpoint
1 Authentication Request
2 Authenticate user
3 Return Authorization Code
4 Retrieve Tokens Using Authorization Code
5 Retrieve Userinfo Using Access Token

User Attributes Shared with Relying Parties

A Relying Party integrating with Identification as a Service will receive the following user data upon successful identification.

- Family_name
- Family_name_am
- Given_name

- Given_name_am
- SSN

Integration Process and Requirements

Once a Relying Party (RP) decides to integrate with imID API and defines its use cases, the following process takes place:

1. RP Receives the Integration Document

The RP is provided with this integration document to familiarize themselves with the implementation details.

2. Technical Deep Dive (if needed)

A technical call or meeting may be arranged between imID and RP technical teams for a deep dive into the integration process and to address any questions.

3. Establishing a Communication Channel

A communication channel is set up for ongoing collaboration between imID and RP technical teams. The preferred platform is Telegram, but alternative methods can be chosen based on mutual agreement.

4. RP Submits Required Information

The RP sends the required information of the *Test Environment* to **enterprise@imid.am.**

Please note that the same information must be generated and provided for the *Production/Live Environment* before launching the feature.

- **Official Name:** The official name of the Relying Party.
- **Service/Website URL:** The URL of the service requesting authentication.

- **Callback URL:** The callback URI where users will be redirected after authentication. Important! For security and compatibility reasons, the callback URL must use HTTPS and be served over port 443. Any callback endpoint that does not meet these requirements will be rejected by the system.
- **Public Certificate:** The public certificate for server-initiated call authentication.
- **KID:** The Key ID (KID) of the service principal's public key.

5. RP Receives Credentials

Once the required information is submitted, imID provides the RP with the following credentials:

- **Audience**
- **OIDC URL**
- **Client ID**
- **Client Secret**
- **Assigned Scopes**

How to Generate a JSON Web Key (JWK)

To generate a JWK for authentication, follow these steps:

Step 1: Generate a Private Key and Certificate

Run the following OpenSSL command to create a key pair:

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days
3650 –nodes
```

This example generates a certificate valid for 10 years (3650 days). You can modify the '-days' parameter to set a different validity period based on your requirements.

Step 2: Convert PEM to JWK

Use the following Docker command to convert the generated private key to JWK format:

```
docker run -i danedmunds/pem-to-jwk:latest --pretty <<EOF

-----BEGIN PRIVATE KEY-----
MIIJQwIBADANBgkqhkiG9w0BAQEFAASCCS0wggkpAgEAAoICAQDAnim8sfRzPTHC

-----END PRIVATE KEY-----

EOF
```

Alternatively, use an online converter: PEM to JWK.

Step 3: Provide the Public Certificate

Once converted, send the following details to imID team:

- Public PEM Certificate
- KID (Key ID) of the JWK

This completes the authentication setup for accessing the imID API.

How to Integrate?

Supported protocols:

- OAuth 2.0 (Authorization Code Flow, Client Credentials)
- OpenID Connect (OIDC)

The Identification as a Service adopts the OIDC (OpenID Connect) protocol in its design. OIDC is a modern authentication protocol built on top of the OAuth 2.0 framework. It provides an identity layer, allowing applications to verify users' identity and obtain user profile information in a standardized manner.

In this section, we will outline the process of integrating with the imID API to securely authenticate citizens using the platform.

Integration Example

As an example to showcase the integration, we are using a **Node.js (Nest.js)** application (openid-client library).

However, the concepts covered here are applicable to any setup, as the OpenID Connect protocol is widely supported across different programming languages and frameworks.

Note that the code below is only a snippet.

Brief Overview

After gaining access to the imID API as a relying party, you should have the following essential information:

- **Client ID**: A unique identifier for your application, used in API requests to identify your source.
- **Client Secret**: A confidential key for secure authentication when making API requests.
- **Client Scope**: Defines your application's specific access permissions within the API.
- **KID**: The Key ID associated with your JWK.
- **Full JWK Private Key**: Required for secure API interactions.
- **Audience**: The intended recipient of the issued tokens.
- **Response Type**: Specifies the expected response format from the authorization server.
- **Callback URL**: The endpoint to which users will be redirected after authentication.
- **OIDC URL**: The OpenID Connect endpoint for authentication and token exchange.

With this information, we can set up the imID OpenlD Connect client.

In this example, we are using the ”openid-client” Node package, a widely known and well-supported package for this purpose. Please note, that RP can use any other package based on any technology stack.

Tel: +374 55 249760
E-mail: enterprise@imid.am
Azatutyun Ave 24/1

Step 1: Setup the imID Connect Client

To begin the integration, we created the instance of an OpenID Connect client using the ”Issuer” class provided by the openid-client library. First, we'll discover the OpenID Connect service provided using the base URL.

https://yesem.am:8643/idp/yesem/oidc/mc

Next, we will create the client and provide all the necessary information, which includes the Client ID, Client Secret, Client Redirect URl, Response Types, Request Object Signing Algorithm, and the previously generated full JWK.

We will use this.client in next steps.

Code integration part with using “openid-client“ node package

```
import { BaseClient, Issuer } from 'openid-client';


async initClient(): Promise<void> {
  const issuer = await Issuer.discover(this.url);
  this.client = new issuer.Client(
    {
      client_id: this.clientId,
      client_secret: this.clientSecret,
      redirect_uris: [this.redirectUrl],
      response_types: [this.responseType],
      request_object_signing_alg: 'RS512',
    },
    {
      keys: [this.privateKey],
    },
  );
}
```

Step 2: Authorization request

You should call authorization API when user want to authenticate

```
async authenticate(phoneNumber: string) {
 const transactionId = uuid();
 const request = await generateClientRequest(phoneNumber, transactionId);
 this.httpService
  .post<ThalesAuthenticateResponse>(
   `${this.url}/authorize`,
   stringify({
    client_id: this.clientId,
    response_type: this.responseType,
    scope: this.scope,
    request,
   }),
   {
    headers: {
     'Content-Type': 'application/x-www-form-urlencoded',
     'Accept-Encoding': 'identity',
    },
   },
  )
}


async generateClientRequest(
 phoneNumber: string,
 transactionId: string,
) {
 const [notificationId, nonce] = [uuid(), uuid()];
 return await this.client.requestObject({
  scope: this.scope,
  aud: this.audience,
  response_type: this.responseType,
  version: 'mc_si_r2_v1.0',
  nonce,
  login_hint: phoneNumber,
  acr_values: '4',
  binding_message: 'Authoize imID Access',
  context: 'Do you want to authenticate using imID?',
  correlation_id: transactionId,
  client_notification_token: notificationId,
  notification_uri: this.redirectUrl,
 });
}
```

Step 3: Handle Callback

You will receive the result to your callback URL which we pre-registered

```
async handleCallback(
 request: Request,
): Promise<ThalesUserInfo> {
 try {
  const payload = request.body as CallbackPayload;
  const authToken = request.headers.authorization || '';
  const transactionId = payload.correlation_id;
  const redisData = getDataFromRedis(transactionId); // You should get your stored data
 if ('error' in payload) {
 handleCallbackError(payload.error); // handle error
}

if (redisData.notificationId !== authToken.split('Bearer ')[1]) { // check token
 throw new UnauthorizedException('Invalid authorization token');
}

if (redisData.authenticationRequestId !== payload.auth_req_id) {
 throw new BadRequestException('Parameter mismatch: auth_req_id');
}

const tokenSet = await getTokenSet(request, redisData.nonce);

if (!tokenSet.access_token) {
 throw new FailedDependencyException('The access token is missing');
}

return await this.client.userinfo<ThalesUserInfo>(tokenSet.access_token, {
 method: 'POST',
 via: 'header',
 params: { correlation_id: transactionId },
});
} catch (error) {

}
}

async getTokenSet(request: Request, nonce: string) {
 const params = this.client.callbackParams(request);

 return await this.client.callback(this.redirectUrl, params, {
  nonce,
 });
}
```

OpenAPI (Swagger)

Please see the OpenAPI documentation for seamless implementation, detailing available endpoints, authentication methods, and response structures: https://api-demo.imid.am/api

***AuthAPI KEY*- YTiQvZzT70**

Error Handling

Depending on the case, an RP may receive either error codes or error messages. Below are the key errors that an RP should handle.

Endpoints	Error Description		Message Examples
Authorization endpoint	HTTP500	Internal error. *This error may occur in several scenarios, but one key instance is when a user repeatedly tries to authenticate after a failure. In such cases, they need to wait approximately 200 seconds for the system to recover.*	Փորձեք մի փոքր ուշ Please try later
	ERR201	User not found	Ձեր imID-ն ակտիվացված չէ

			Your imID is not activated
Callback	user_cancel	User does not enter PIN	Հարցումը չի հաստատվել օգտատիրոջ կողմից The request was cancelled by the user

imID Integration Kit

Please refer to the imID Integration Kit for branded guidelines, including colors, logo variations, fonts, and other imID products.

https://imid.am/integration-kit
Password: 2.pd\!X1f5fN

imID User-Facing Text Requirements

- Make sure to write “imID” when referring to Mobile ID-related errors or any messages and user facing texts (including texts on buttons)
- For multi-language platforms do not translate “imID”, keep it as is.

imID Contact Details

Email: enterprise@imid.am

Phone: +374 98 811 190

Responsible Person: Gevorg Amirjanyan

Tel: +374 55 249760
E-mail: enterprise@imid.am
Azatutyun Ave 24/1

FAQ

1. **Q**: After we initiate the authentication request, what is the maximum amount of time we should wait for your callback response? Meaning, what is the latest possible time you will send a callback?
   A: The timeout currently configured is 300 seconds.
   In practice, the average timings are as follows:
   - Time from authentication request to applet opening for PIN entry: approximately 2–5 seconds.
   - Time from PIN submission to receiving the response: approximately 5–15 seconds.

2. **Q**: What is the maximum amount of time the applet remains open for the user to enter their PIN before an error is returned?
   A: On the applet side the timeout is 120 seconds.
   Note that it is not guaranteed that the transaction will expire exactly after 120 seconds because of how the scheduler works. Normally it will expire between 120 seconds and 300 sec +1min.

3. **Q**: Are there any scenarios where we might not receive a callback at all, even after 300 seconds +1min?
   A: Yes, in case of network cut between you and imID MobileID system, you won't receive a response.

4. **Q**: How are applet opening for PIN entry and receiving a phone call interconnected?
   A: <u>Case 1: User receives a call before the applet opens:</u>
   If the user receives a call first, it may delay the applet opening. The applet will appear either after the user answers the call or once the call ends.
   <u>Case 2: Applet opens before the user receives a call:</u>
   If the applet opens first, and then the user receives a call, the applet will remain accessible once the user unlocks the phone to answer the call.
   In any case, the applet opens normally even when the user is on a phone call.