



Centiglobe technical integration kit

This document should help a technical implementer of the Centiglobe REST API on the journey from starting the integration to certification in staging and finally using the implementation in production.

Get started	2
Credentials	2
Staging environment	2
Default test payment destination	2
Example code	3
Self-service tests	3
Migration to Production	3
Certification	3
Signing key registration	4
Appendix 1 - Acquire the access token	5
Appendix 2 - Payload signing and verification	6
Signed data as part of payload	6
Protected Header	6
Payload	6
Verify Signed data as part of payload	7



Get started

Centiglobe provides a staging environment that is similar to the production environment in setup but contains no live transactions or real money.

Credentials

To get access and be able to send payments, an authorization file and a signing key are required. Centiglobe will generate authorization credentials and the signing key and deliver them in a file.

Staging environment

Centiglobe will set up a payin member account on the DLT in your name. A set of default payment corridors will also be added for this member.

In staging, there is a possibility to test regular payments. These test payments are accessed using <https://staging.system.centiglobe.com:8000>.

To access the SCENARIO-BASED service use <https://staging.system.centiglobe.com:8001>.

Default test payment destination

The default configured corridors that are always configured and can be used by all members are:

Payin Currency	Payout MemberName	Service	Payout Currency	amlSchema
EUR	payout-fake	FAKE	SEK	<i>Schema 1*</i>
USD	payout-fake	FAKE	USD	<i>Schema 1*</i>
EUR	payout-fake	SCENARIO-BASED	SEK	<i>Schema 1*</i>
USD	payout-fake	SCENARIO-BASED	USD	<i>Schema 1*</i>

Schema 1: <https://schema.centiglobe.com/aml/id-test-3-20220307.json>

Other configurations (e.g. other currencies) can be created in agreement with Centiglobe.



Example code

Centiglobe provides an example repo to show a live example of how a payment works in code. [<https://bitbucket.org/centiglobe/centiglobe-integration-examples>]

Use the example code to verify received credentials and correct member setup in the staging environment.

The example code provided information and implementation on how to:

- Acquire the access token from the authentication system.
- Sign a Payload with the Member Signing Private Key.
- Verifying the integrity of a PaymentStatement with the Payout Signing Public Key

Self-service tests

Centiglobe Certification cases can also be run as self-service tests. The service is implemented as a proxy in front of the payment system to simulate recovery and timeout scenarios. Use <https://staging.system.centiglobe.com:8001> to access the self-service tests.

The certification document [Centiglobe system certification - payin.pdf] provides more information about the scenarios and the self-service tests.

Migration to Production

Certification

Centiglobe requires all members to complete a certification of integration. Contact Centiglobe to request a certification session.

There could be scenarios that aren't relevant for you as a member and can be exempted from certification. This will be noted, including the reason, in the certification protocol.

Before requesting a certification, please complete the relevant self-service scenarios, which will be included in the certification. Then, the certification will be carried out in a joint session together with designated personnel from Centiglobe.



Signing key registration

In production, the member signing key is generated by the member, and only the public part is sent to Centiglobe for registration on the DLT.

Centiglobe provides an example repo [<https://bitbucket.org/centiglobe/cg-key-generator>] to assist in key generation.



Appendix 1 - Acquire the access token

Centiglobe uses the AWS Authentication System (Cognito) for handling users. We recommend using the Direct Authentication mode. When using this mode, there is a requirement to use a third-party library provided by AWS. We have provided links to the libraries that we use in the Example code below. Similar libraries do exist in other languages.

Java	https://github.com/aws/aws-sdk-java-v2
Javascript	https://github.com/aws-amplify/amplify-js/tree/master/packages/amazon-cognito-identity-js
.NET	https://github.com/aws/aws-sdk-net-extensions-cognito/

There are reference implementations in the Example code that shows how to use the libraries.

When requesting the access token, the service responds with two other tokens, the refresh token and the ID token. The refresh token is used to refresh the access token without initiating an authentication process by the library. The ID token is not used within our application.



Appendix 2 - Payload signing and verification

Signed data as part of payload

In this section, we describe the content of the JWS that is expected to be sent to our endpoints. The full format of the JWS is defined by rfc7515

[\[https://www.rfc-editor.org/rfc/rfc7515\]](https://www.rfc-editor.org/rfc/rfc7515).

Signing the data should be done with the MemberSigningPrivateKey.

The JWS should look like this:

```
BASE64URL(UTF8(JWS Protected Header)) || '.' ||  
  BASE64URL(JWS Payload) || '.' ||  
  BASE64URL(JWS Signature)
```

For example code see the .NET code example.

Protected Header

Signing the data should be done with the MemberSigningPrivateKey. The signing algorithm is provided in the signing key.

The Protected header should look like this:

```
{  
  "alg": MemberSigningPrivateKey.alg,  
  "typ": "JOSE+JSON"  
}
```

Payload

The payload depends on which endpoint is being used. Check the Centiglobe REST API for more information about what data is required for each endpoint.

```
{  
  // content in JSON format  
}
```



Verify Signed data as part of payload

When verifying the received Payment statement data, it is critical to know that we can trust that the counterparty that signed the data is who it claims to be. Payout JWKS is available in the /well-known/ endpoint and should be used to verify the JWS claim.

Refer to the example repo [<https://bitbucket.org/centiglobe/centiglobe-integration-examples>] for what should be validated in the JWS payload.