ČESKÁCZECH
BANKOVNÍBANKING
ASOCIACEASSOCIATION

Banking Activities Standards

Standard CBA

Czech Standard for Banking IDentity Assurance

Public from: 1. 9. 2020
Version: 1.0

Changes

Date	Note	Guarantor of the Change
31.8.2020	**Basic document v.1.0**	ČS Petr Michalík

Table of Contents

1 Introduction 5
2 Standard governance 5
3 High Level Overview 6
3.1 Glossary 6
3.2 How to read this document 6
3.3 High level technical decisions 6
3.4 What does the Bank need to implement to be compliant? 8
4 Security Guidelines 9
4.1 Glossary 9
4.2 Requirements 9
4.3 Security Strategy 9
4.4 Ciphers & Hash algorithms 10
4.5 Protocol layer 10
4.5.1 TLS 10
4.6 Authentication (OpenID Connect) 11
4.6.1 Provider 12
4.6.2 Relying party 13
4.6.3 Possible attack vectors and their mitigation 13
4.7 The API layer 15
4.7.1 Input Validation 15
4.7.2 Escape Content: 15
4.7.3 Protect against Cross-Site Request Forgery: 16
4.7.4 HTTP Status codes: 16
4.7.5 Headers 16
4.7.6 Compression 18
4.7.7 Audit on the side of RP 18
4.7.8 API-Key Authentication 19
5 Technical Explanation of the Overall Flow 20
5.1 Glossary 20
5.2 Overview 20
5.3 1. Discovery 21
5.4 2. Dynamic registration 22
5.5 3. Authorization 22
5.6 4. Authorized API calls 23
5.7 5. Logout and token revocation 24
6 Bank API EndPoint overview 25

6.1 OpenID Configuration Discovery API 25
6.2 JWK Keys Discovery API 26
6.3 HealthCheck API 27
6.4 Dynamic Registration API 27
6.5 Authorization API 30
6.6 TokenInfo API 32
6.7 Token revocation API 33
6.8 UserInfo API 33
6.9 Profile API 36
7 Attachments 40

1 Introduction

This document is intended to give a bird eye overview of the BANK IDENTITY systems, it's technical goals, decisions, and related motivations. This document should also provide a starting point for bank implementors.

Within the provided banking identity, there may be communication between banks and various entities, within which it is necessary to share a certain data structure while maintaining an appropriate level of security.

Representatives of 16 banks participating in the further development of ČOBS (Czech Open Banking Standard) agreed that it would be appropriate to extend that standard by a brand new standard for the description of technical data transmitted within the Banking Identity services which are expected to be used in the Czech Republic in the near future.

Main features of the standard:

- The standard **is based on general standards**, **it is easy to implement** (the basis is already established in procedures and principles of the Czech Open Banking), which ensures an **easy combination of identity and API of bank** services (eg API for PSD2)
- The standard **is voluntary for banks,** it could be taken as a recommendation on how to technically provide a banking identity within digital services
- The standard **meets legislative requirements** (e.g. AML, Banking Act) and places high demands on security
- Technically **uses the international OpenID Connect (OIDC) standard** and **OAuth2**, even for data scope definitions it also includes the so-called Identity Assurance (eKYC) as a new extension of OIDC

2 Standard governance

The major version of the standard will be changed maximum once six months. Suggestions for change may be given by any new mandatory regulation, by a third party through the ČBA or by the Czech Open Banking Standards working group itself.

A minority release may occur more frequently. E.g. if it is necessary to correct any published error or clarification. The minor version of the document is represented by a number after the dot and is available in the same way as the main version.

Any suggestion for a change must be approved within a proper amendment procedure six months before the due date of the changes in force. This implies that the suggestion for a major change should be submitted no later than one year before the planned implementation date in case of legislative changes.

3 High Level Overview

3.1 Glossary

- **PII** - Personally Identifiable Information - End-User information like name, email address or state issued IDsPII - Personally Identifiable Information - End-User information like name, email address or state-issued IDs
- **AML** - Anti Money Laundering - A set of laws and regulations intended to prevent money laundering KYC - Know Your Customer - A Bank API which allows SePs to gather information about End-Users
- **KYC** - Know Your Customer - Bank API which allows RP to gather information about end users
- **IDP** - Identity Provider - Component on the side of the bank which handles authorization, consent and issues tokens
- **RP** - Relying Party - A third party which is registered to consume Bank APIs

3.2 How to read this document

The CAPITALIZED words throughout these guidelines have a special meaning as defined in [RFC2119]:

> The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119.

3.3 High level technical decisions

Why use standards instead of making a custom-tailored solution?

- Use of open standards such as OpenID Connect and OAuth2 allows all parties to decrease integration costs by reusing existing code and solutions.
- Use of well-defined standards removes most ambiguity and simplifies technical communication.
- Technical communication is easier as all parties can reference the open standard and use language and technical terms defined in the standard.

Why is it important to aim for maximal standard compliance?

- Full standard compliance allows for existing solutions and code to be used with minimal modification.

- It reduces quantity of bugs from misunderstood specification and wrong assumptions.
- Low standard compliance exposes for nasty surprising bugs that stem from code and solutions assuming standard compliance when that is not the case. These can be costly.

Why were OpenID Connect and OAuth2 selected as the standards to follow?

- OAuth2 is a very well established authorization framework that is used by most of high profile players in the IT vertical. These companies include Google, Microsoft, Facebook, Twitter, LinkedIn and many others. This implies a vast quantity of available tooling, code and solutions - Open Source or otherwise.
- OpenID Connect is a layer above OAuth2 which improves on security and fills gaps in OAuth2 regarding a, discovery and session management. OpenID Connect is a relatively newer protocol, however it is well though out and already in use by most of the companies above.

Why is dynamic registration required?

One of the APIs that Banks need to implement is dynamic registration. Purpose of this API is dynamically register RP into the Bank's systems. This is required because:

- This gives the Bank intelligence into which Third Parties it does business with
- It allows the Bank to properly display the consent screen to the End-User. Including RP specific branding, logo and TOS link.
- It gives the control over the consent screen into the hands of the Bank, rather than RP.
- It allows the Bank to be standards compliant and possibly use already made OpenID Connect or OAuth2 middleware in front of it's IDP just lightly modify it, rather than forcing the Bank to implement a custom solution.

What hypothetical alternatives would there be to dynamic registration?

All of the following solutions are standard non-compliant and thus incur costs mentioned above.

- RP MAY expose additional back-channel APIs which would expose branding information for the consent screen.
    - This would force the Bank to make additional calls and thus slow down the authentication process
    - Bank would need to fully implement this new non-standard protocol
    - It would only increase development costs for the Bank, since the Bank would still need to keep all the branding information, but now would have an additional step to reacquire it during each authorization
    - Bank would not be directly notified of branding changes

- RP MAY stream branding updates over some other non-standard back-channel like a Kafka queue
    - This would be exactly the same as the standard-compliant solution in a sense that the Bank would keep all the branding information
    - It would introduce another moving part, the queue server, which would need to be managed and maintained
    - Thus it would break the standard for no additional benefit
- Bank MAY only handle the authentication and let RP handle the consent screen
    - This gives control over the consent screen to the RP
    - It is less secure since now RP decides for the Bank what the RP can do with End-User's data
    - RP would have to propagate this authorization information into the Bank, which would need to validate it

3.4 What does the Bank need to implement to be compliant?

Please see Bank API EndPoint overview and Technical explanation of the overall flow for detailed technical description.

In big strokes, as a Bank you will need to implement an OpenID Connect wrapper around your Identity Provider (IDP). Thus it will provide:

- Dynamic registration API - this API will allow RPs to register into your your identity system
- Discovery API - this is a set of well-known EndPoints which are there for auto configuration and URL discovery. It allows you flexibility regarding EndPoint locations, encryption algorithms and more. It also allows you to easily rotate encryption keys.
- Authorization and token exchange APIs - These are well specified OAuth2/OpenID Connect APIs which serve the core authentication
- Token revocation API - For RP originated logouts

Your solution will also need to have the following capabilities:

- Download RP encryption keys, URL of these is delivered during dynamic registration
- Encrypt and validate JSON Web Tokens (JWTs) - These enhance security and are prescribed in the OpenID Connect specification
- Setup Mutual TLS channel to the RP
- Properly use and verify Public Key Infrastructure (PKI) certificates

4 Security Guidelines

Due to the nature of the information provided by BankID, a high level of security is required.

The goal of this part is to describe the security of this solution. It provides both conceptual and implementation guidance on how to build the API in a secure way.

4.1 Glossary

- Personal Identifiable Information (PII) - In EU also known under wider term [Personal Data](https://en.wikipedia.org/wiki/Personal_data)
- Provider - Bank or other entity that provides KYC APIs to the consumers
- Relying Party (RP) - Consumer of KYC APIs
- Third Party (TP) - Party to which RP can share PII and other end user data (with their consent)

4.2 Requirements

KYC and KYC Connect are very sensitive services from the security standpoint. The following requirements are posed on the implementation:

- **Confidentiality** - Bank exposes Personally Identifiable Information (PII) via the APIs.
- **Non-repudiation** - Critical business decisions are made. There are legal implications if the provided data turn out to be incorrect. API consumers must be sure the data came from the bank and are genuine.
- **Integrity** - The PII exposed are valid as a set of properties and the claim made about the user must remain unchanged by third parties when transmitted between API Consumer and the bank.
- **Availability** - The system must be highly secure, but also highly available as critical infrastructure would rely on the KYC and KYC Connect implementations.

4.3 Security Strategy

The following principles were employed while designing the security of this system:

- **Use standards where possible** - The proposed security solution relies heavily on PKI and OpenID Connect security that is standardized.
- **Err on the side of security** - If there are multiple standardized ways how to implement things, the more secure one is chosen.

4.4 Ciphers & Hash algorithms

- MD5 and SHA1 MUST NOT be used for known collision attacks
- Weak ciphers such as DES and RC4 MUST NOT be used
- ECDSA key size MUST be at least 256 bits (preferred)
- RSA key size MUST be at least 3072 bits

For the purposes of this guideline **STRONG JWT CIPHER** is defined as one of the following signing algorithms:

- `ES512` (recommended)
- `PS512`

Or `A256GCM` for encryption.

4.5 Protocol layer

4.5.1 TLS

All communication MUST use a secure version of TLS. At the time of this writing, **TLS 1.2** is the most wide-spread version with no known serious vulnerabilities. Server MUST NOT allow protocol renegotiation to older/weaker TLS versions. Communication over the HTTP protocol MUST use **HTTPS**.

Any non-TLS requests SHOULD be dropped.

Server of the Provider SHOULD be protected from CRIME attack, TLS compression MUST be disabled (See: CVE-2012-4929)

4.5.1.1 Certificates

Provider MUST use **QWAC** certificates for server authentication.

Provider MUST use **QSEAL** certificates for JSON Web Signatures in payloads.

Server certificates MUST be rotated regularly; at least once a year.

Additional resources:

- QWAC

Certificates MUST have the following properties (see RFC5280, RFC7469 for additional requirements and recommendations):

- X.509 certificates key length MUST be strong

- ECDSA at least 256 bits (preferred)
- at least 2048 bits for RSA
- X.509 certificates MUST be signed only with secure hashing algorithms
- Certificate with FQDN MUST be used for server authentication
- SSL Wildcard certificate MUST NOT be used for server authentication

When **verifying certificates**, the following checks MUST be passed:

- Checking if the Certificate Authority (CA) is a known one (meaning one considered trusted).
- Checking that the certificate is currently valid
- Checking that the name of the site and the FQDN reported in the certificate match.
- If certificate pinning is used, the certificate or intermediate certificate MUST match.

4.5.1.2 mTLS (two-way authentication)

mTLS MUST be employed for increased transport layer security. Client certificate used for authentication MUST be a **VALID** certificate verifiable against a trusted root certificate store.

RP Client certificates MUST be rotated regularly; at least once a year.

RP MUST use **QWAC** certificates for client authentication.

The process of client certificate handover to the Provider is beyond the scope of this guideline

4.5.1.3 Certificate pinning

RP MAY use certificate pinning for added security when establishing TLS connection. RP should consider whether to pin leaf certificate or intermediate certificate. The latter provides less guarantee about the authenticity of the Provider in exchange for less operational burden. The decision is left to the implementing party and SHALL be made on a case-by-case basis.

Additional resources:

- OWASP: Certificate and Public Key Pinning
- TLS Certificate pinning 101

4.6 Authentication (OpenID Connect)

The client certificate used for authentication MUST be a VALID certificate verifiable against a trusted root certificate store.

Most security features of OAuth2 and OpenID Connect are used.
Encryption MUST be done after the signature as stated in the OpenID specification to ensure integrity and non-repudiation

4.6.1 Provider

Most notably, Provider MUST use JWS and JWE where OpenID connect protocol allows it:

- ID Token in token endpoint response *(signed and encrypted)*
- UserInfo endpoint response *(signed and encrypted)*

Provider MUST support encryption and signature of requests objects and responses and it needs to indicate it in the OpenID discovery endpoint.

Provider signatures MUST be done using the `X.509` certificate.

Provider MUST encrypt response to RP using the RPs key.

The OpenID connect server MUST have the following settings:

- `jwks_uri` - Must be set and point to X.509 based JSON Web Key Set.
- `id_token_signing_alg_values_supported` - MUST be set to STRONG JWT CIPHER
- `id_token_encryption_alg_values_supported` - MUST be set to STRONG JWT CIPHER
- `id_token_encryption_enc_values_supported` - MUST be set to STRONG JWT CIPHER
- `userinfo_signing_alg_values_supported` - MUST be set to STRONG JWT CIPHER
- `userinfo_encryption_alg_values_supported` - MUST be set to STRONG JWT CIPHER
- `userinfo_encryption_enc_values_supported` - MUST be set to STRONG JWT CIPHER
- `request_object_signing_alg_values_supported`- MUST be set to STRONG JWT CIPHER
- `request_object_encryption_alg_values_supported` - MUST be set to STRONG JWT CIPHER
- `request_object_encryption_enc_values_supported` - MUST be set to STRONG JWT CIPHER
- `token_endpoint_auth_methods_supported` - MUST include `private_key_jwt`
- `token_endpoint_auth_signing_alg_values_supported` - MUST include `private_key_jwt`

4.6.1.1 Registration

Dynamic registration endpoint MUST be protected using mTLS.

Dynamic registration endpoint SHOULD be protected by shared secret. Consumer provides this secret in `Authorization` HTTP Header using `Basic` [authentication scheme].

4.6.2 Relying party

4.6.2.1 Registration

During client registration RP MUST use the following parameters:

- `token_endpoint_auth_method` - MUST be set to `private_key_jwt`
- `grant_types` - MUST be set to `authorization_code refresh_token`
- `jwks_uri` - MUST be set and point to X.509 based JSON Web Key Set.
- `id_token_signed_response_alg` - MUST be set to STRONG JWT CIPHER
- `id_token_encrypted_response_alg` - MUST be set to STRONG JWT CIPHER
- `id_token_encrypted_response_enc` - MUST be set to STRONG JWT CIPHER
- `userinfo_signed_response_alg` - MUST be set to STRONG JWT CIPHER
- `userinfo_encrypted_response_alg` - MUST be set to STRONG JWT CIPHER
- `userinfo_encrypted_response_enc` - MUST be set to STRONG JWT CIPHER
- `request_object_signing_alg` - MUST be set to STRONG JWT CIPHER
- `request_object_encryption_alg` - MUST be set to STRONG JWT CIPHER
- `request_object_encryption_enc` - MUST be set to STRONG JWT CIPHER
- `request_uris` - MUST be set to HTTPS based URL fully controlled by RP.

Returned `registration_access_token` MUST be treated as an security asset and be securely stored.

RP MUST pass request parameters as JWTs signed (by its key) and encrypted (by the key of Provider) as per the [OpenID specification].

4.6.3 Possible attack vectors and their mitigation

Please be sure to read through common OAuth2 threats and their mitigations in [RFC6819].

- Dynamic registration by an unauthorized third party
- The adversary would need access to the mTLS keys to be able to call the Dynamic Registration API
- Dynamic registration response eavesdropping
- If the adversary had access to the mTLS signing keys and was able to eavesdrop the Dynamic Registration response, they would gain access to the client_id; however they would still not be able to craft signed and encrypted JWTs used in the following steps since asymmetric keys are used.
- Eavesdropped communication between the RP and the Bank
- RP <==> Bank communication is protected against eavesdropping using Mutual TLS
- Eavesdropped communication between mTLS terminator and OIDC handler
- Even if an adversary manages to gain access to inner network (after mTLS is terminated), they will not be able to read specific message payloads thanks to JWT encryption and signing
- Compromised encryption keys
- There are mechanisms in place to mitigate this issue, namely PKI based certificate revocation and an easy way to propagate rotated JWK keys through the use of the JWKS endpoint (from Discovery or Dynamic Registration)
- redirect_uri change during the `/auth` call
- A redirect_uris whitelist is a mandatory requirement during dynamic registration
- CSRF attacks on the `/auth` endpoint
- These are out of scope, since protection will be handled on the SONIA side
- Leak of Confidential Data in HTTP Proxies
- Since mTLS 1.2 is used, all proxies along the way would need to have access to the mTLS keys, which means that only trusted entities can access Credential Data through HTTP proxies
- Replay of Authorized Resource Server Requests
- To prevent replay attacks, we use the nonce query parameter in the authorization call and in JWTs
- Authorization code in the browser history
- Since code is passed as a query parameter during a redirect from the Bank consent screen, it MAY be visible in the browser history. However, this does not pose a security threat because code is short lived and is exchanged (and blacklisted for reuse) in the Bank immediately upon its reception by SONIA
- CRIME attack
- To prevent the CRIME attack, mTLS compression SHOULD be avoided

4.7 The API layer

Securing the API layer between Consumer and Provider is done mainly by using the OpenID Connect keys to sign and encrypt API communication.

- OpenID Provider key from `jwks_uri` is used to sign KYC and other future api responses (MUST)
- OpenID Consumer key is used to encrypt KYC and other future api responses (MUST)

All responses from KYC and other future APIs MUST be signed by key from `jwks_uri` on OpenID Provider side. `jwks_uri` MUST contain X.509 v3 extensions to validate certificate and all its chain. `jwks_uri` MUST contain all certificates from certificate chain, from leaf to root, in one .pem file.

- All responses from KYC and other future APIs MUST be encrypted by key from `jwks_uri` on OpenID Consumer side.
- JWT from KYC and other future APIs MUST be first signed by OpenID Producer key and then encrypted by OpenID Consumer key.

4.7.1 Input Validation

Secure parsing:

> It is recommended to use parsing packages for validating inputs. These packages should NOT be VULNERABLE to XXE or similar attacks.
>
> - Strong typing: It is recommended to use strongly typed inputs.
> - Validating incoming content-types: Server SHOULD NEVER assume the Content-Type. The Content-Type Header and content SHOULD ALWAYS be checked to make sure they are of the same type. An unexpected Content-Type or a missing Header SHOULD result in a 406 Not Acceptable response.
> - Validating response types: Server SHOULD NEVER copy the Accept Header to the response Content-Type. Server SHOULD ALWAYS check the Accept Header for allowed types.

4.7.2 Escape Content:

- Server and browser MUST ALWAYS sanitize all input data from HTML tags and attributes. NEVER TRY TO DO IT BY YOURSELF. This SHOULD be done by known library or the auto-escaping features of some template library.

4.7.3 Protect against Cross-Site Request Forgery:

- Every RESTful resources MUST be protected from Cross Site Request Forgery. More info at [How to prevent XSS].

4.7.4 HTTP Status codes:

> While designing a REST API, DON'T just use 200 for success or 404 for error. Every error message needs to be customized as NOT to reveal any unnecessary information. Here are some guidelines to consider for each REST API status return code. Proper error handle may help to validate the incoming requests and better identify the potential security risks.
>
> - 200 OK - Response to a successful REST API action.
> - 400 Bad Request - The request is malformed, such as message body format error.
> - 401 Unauthorized - Wrong or no authentication ID/password provided.
> - 403 Forbidden - It's used when the authentication succeeded but authenticated user doesn't have permission to the requested resource
> - 404 Not Found - When a non-existent resource is requested
> - 405 Method Not Allowed - The error checking for unexpected HTTP method. For example, the Rest API is expecting HTTP GET, but HTTP PUT is used.
> - 429 Too Many Requests - The error is used when there may be DOS attack detected or the request is rejected due to rate limiting

4.7.5 Headers

- Server versioning information or any other sensitive information from the HTTP headers SHOULD BE removed/masked according to industry best practices. This prevents any form of targeted attacks since the vulnerabilities are mostly specific to the vendors.
- Server

  > This header contains information about the backend server (type and version). For instance, the screenshot below shows that the webserver that runs Nike's web page is Jetty, version 9.4.8.v20171121.

- X-Powered-By

  > It contains the details of the web framework or programming language used in the web application.

- X-AspNet-Version

> As the name suggests, it shows the version details of the ASP .NET framework. This information may help an adversary to fine-tune their attack based on the framework and its version.

- If the communication between the OpenID Provider and the Consumer uses mTLS, the following headers MUST be set:
- Strict-Transport-Security: max-age=; IncludeSubDomains

> Without HSTS enabled, an adversary can perform a man-in-the-middle attack and steal sensitive information from the web session of a user. Imagine a scenario where a victim connects to an open Wi-Fi which is actually controlled by an attacker. Accessing a website over HTTP would allow the attacker to intercept the request and read the sensitive information. (The site is on HTTPS but the user accesses it with HTTP which later gets redirected to HTTPS). If the same user had accessed the website earlier, the HSTS details recorded in the browser would have caused the connection to be made over HTTPS automatically

- Content-Security-Policy

> Content Security Policy is used to instruct the browser to load only allowed content defined in the policy. This uses the whitelisting approach which tells the browser where to load the images, scripts, CSS, applets, etc. from. If implemented properly, this policy prevents exploitation of Cross-Site Scripting (XSS), ClickJacking and HTML injection attacks.

- Access-Control-Allow-Origin

> Access-Control-Allow-Origin is a CORS (Cross-Origin Resource Sharing) header. This header allows the defined third party to access a given resource. This header is a workaround for restrictions posed by the Same Origin Policy which does not allow two different origins to read each other's data.

- X-XSS-Protection: 1; mode=block

> This header is designed to protect against Cross-Site Scripting attacks. It works with the XSS filters used by modern browsers

- X-Content-Type-Options: nosniff

> This response header is used to protect against MIME sniffing vulnerabilities. MIME sniffing is a feature of the web browser that serves for examining the content of the file being served.

4.7.6 Compression

- Compression MUST be DISABLED in mTLS communication to prevent [CRIME] attack.

> The CRIME attack is used to extract session tokens protected by the SSL/TLS protocol. CRIME exploits the data compression feature of SSL and TLS. As the compression happens at the SSL/TLS level, both the header and the body are subjected to compression. SSL/TLS and SPDY compression use an algorithm called DEFLATE which compresses HTTP requests by eliminating duplicate strings. CRIME takes advantage of the way duplicate strings are eliminated to guess session tokens by systematically brute forcing them.

- HTTP compression MUST be DISABLED to prevent the [BREACH] attack.

> The BREACH attack is quite similar to the CRIME attack but there are subtle differences. This attack also leverages compression to extract data from an SSL/TLS channel. However, its focus is not on SSL/TLS compression; instead, it exploits HTTP compression. Here, the attack tries to exploit the compressed and encrypted HTTP responses instead of requests as it was the case with the CRIME attack.

4.7.7 Audit on the side of RP

In certain legal cases, RP MUST be able to provide the content received from the Bank. This content MUST be auditable. On the other hand, the content contains PII and thus is very sensitive in nature.

The following guideline MUST be adhered to for auditing:

- If the data from Bank is relayed to the RP, the digest of original data MUST be included and signed in the relayed data.
- Digests of messages from Bank MUST be stored together with digests of any potential derived messages to RP.
- Dates of every messages sent and received MUST be logged.
- Data MUST be auditable for the duration of 15 years.
- Special key pair for auditing MUST exist. The parts of the key are defined as **APub** for public part and **APriv** for the private part.
- **APub** MUST be used by the RP systems to encrypt payloads of Bank data messages. Payloads of the messages MUST NOT be stored in plaintext or encrypted using symmetric cipher.
- **APriv** MUST be in cold storage and MUST be stored in tamper resistant device (HSM, for example).
- Access to **APriv** MUST be limited and controlled by strict security process. _(This security process is beyond scope of this guideline)_
- RP systems MUST NOT have access to **APriv**. Decryption by **APriv** MUST be manual process and the act itself MUST be audited.

4.7.8 API-Key Authentication

Certain APIs (for example planned document signing API) MAY require authentication that is not bound to specific End-User but still need to enforce rate-limits and associate, log or bill requests per application. API-Key authentication SHOULD be used to satisfy this use-case.

During dynamic registration, Bank generates a `client_api_key` API-Key token and RP stores it. RP MUST include the API-Key token in the `X-API-Key` HTTP header when requesting said Bank APIs.

Please note that **API-Key token is NOT required when requesting OAuth2/OIDC resources**.

5 Technical Explanation of the Overall Flow

This part describes in technical terms the overall RP-Bank flow and it's intricacies, it's intended purpose is to present a big picture of the APIs involved.

5.1 Glossary

- **EP** - EndPoint - An URL or pathname on which service exposes it's REST API
- **IDP** - Identity Provider - Component on the side of the bank which handles authorization, consent and issues tokens
- **OIDC** - OpenID Connect - Authentication framework, superset of OAuth2 authorization framework
- **RP** - Relying Party - An OIDC middleman and orchestrator which sits between Service Providers and Banks
- **SeP** - Service provider - A third party which is registered in the SONIA and intends to consume Bank APIs
- **KYC** - Know Your Customer - Bank API which allows SePs to gather information about end users
- **JWT** - JSON Web Token - Base64 encoded, signed and possibly encrypted JSON document
- **JWK** - JSON Web Key - JSON describing keys for JWT encryption, description, signing and verification
- **PII** - Personally Identifiable Information - End-user information like name, email address or state issued IDs
- **CRUD** - Create Read Update Delete - A full set operations for working with entities over the REST interface

5.2 Overview

1. Discovery – RP calls the Bank's well-known discovery EP to acquire endpoint locations, crypto certificates and other settings
2. Dynamic registration - RP registers a new Client application in Bank's IDP by calling the dynamic registration EP - it's URL has been acquired using Discovery. During this call RP also registers other settings, like certificates and whitelist of `request_uris`. In turn IDP generates a `client_id` for RP, which will get used in following calls
3. Authorization - RP redirects the end-user's browser to Bank IDP's authorize endpoint with `client_id`, `scopes`, `redirect_uri` and a few other things. IDP shows the user a login screen followed by the consent screen, which asks user to approve or deny the requested `scopes`. When the user approves, IDP generates Authorization Code `code` and redirects the user to the `redirect_uri` registered by the RP.

4. Authorized API calls - RP is now authenticated can safely consume Bank's APIs (like KYC) with the `access_token`
5. Logout and revocation - Logout can either be initiated by the RP through calling the Revoke endpoint. This invalidates the consent for a specific SeP/Bank/End-User combination

EndPoints for all of these are documented in the [bank.yaml see Attachments] OpenApi document.

5.3 1. Discovery

RP calls the Bank's well-known discovery EP to acquire endpoint locations, crypto certificates and other settings.

- Bank MUST expose `/.well-known/openid-configuration` OIDC Configuration endpoint. This serves as an entrypoint into several processes and allows banks to have a certain flexibility in regards to specific locations and paths - each Bank can structure it's endpoint locations as it requires as long as they are documented in the OIDC Configuration.
- Bank also exposes a JWK key collection endpoint (URL of which is referenced in the OIDC Configuration). These keys are exposed on a separate endpoint to allow easy key rotation. These keys are later used for:
  1. Verifying Bank signed JWTs
  2. Encrypting JWTs to be sent to a Bank
- Discovery endpoints are documented in the [bank.yaml see Attachments] OpenApi document

Source material and relevant modifications:

- [OpenID.Discovery] OpenID Connect Discovery 1.0
- `jwks_uri` is mandated instead of embedding `jwks`
- Only asymmetric crypto is allowed, other options are omitted
- X.509 certificate is specified using the `x5u` URI reference
- JWK signing as well as encryption is assumed
- `x5u` is removed in favor of `x5c`
- `x5u` is removed in favor of `x5c`
- `claims_supported` is REQUIRED

5.4 2. Dynamic registration

RP registers a new Client in Bank's IDP by calling the dynamic registration EP - it's URL has been acquired using Discovery. During this call RP also registers other settings, like certificates and whitelist of `request_uris`. In turn IDP generates a `client_id` for RP, which will get used in following calls.

- Bank exposes a Dynamic Registration and Client management endpoints (CRUD over Clients). This allows SeP to register into the Bank systems. Which in turn allows the Bank to:
    1. Associate API calls, like the KYC calls, with a specific Service Provider
    2. Present custom consent screen to the end user
    3. Have full knowledge of Service Providers which are accessing it's systems
    4. Have different set of keys for each SeP, thus increasing security
    5. It is possible for the End-User or Bank to selectively logout a single SeP
- These endpoints are also required for proper and secure working of JWT encryption for messages sent by RP to the Bank

Source material and relevant modifications:

- [OpenID.DynamicRegistration] OpenID Connect Dynamic Client Registration
- Notably, this API assumes that JWTs with asymmetric signatures and encryption will be used for the OIDC as well as service APIs when communicating between Bank and RP. For this reason, several fields and options are omitted.
- `jwks_uri` will be used rather than directly embedding `jwks`.
- `default_max_age` is used to prevent tokens with too long of a expiration from causing security issues.
- `client_name`, `logo_uri`, `tos_uri` are required for a bank to have intelligence into what sort of SeP has registered and to have these available for a consent screen.
- [draft-ietf-oauth-dyn-reg-management-11] OAuth 2.0 Dynamic Client Registration Management Protocol

5.5 3. Authorization

RP redirects the end-user's browser to Bank IDP's authorize endpoint with `client_id`, `scopes`, `redirect_uri` and a few other things. IDP shows the user a login screen followed by the consent screen, which asks user to approve or deny the requested `scopes`. When the user approves, IDP generates Authorization Code `code` and redirects the user to the `redirect_uri` registered by the RP.

- `redirect_uri` is configured by the RP during dynamic registration and `client_id` is returned from IDP to SONIA from dynamic registration
- SONIA will utilize the Request Object mechanism documented in the [OpenID.CORE] through the use of `request_uri` query parameter. This is to:

  1. Avoid issues with URL length - different web servers have different URL length limits. URLs that are too long is a real problem in OAuth2 and OIDC.

  2. Provide confidentiality through use of encrypted JWT Request Objects - this prevents all third parties, including the end-user, from seeing credentials used in RP to Bank communication.
- `openid` and `offline_access` scopes will always be sent, other scopes are translated from what the SeP has requested of RP
- `response_type` will ALWAYS be `code`. There is no use for implicit grant between RP and Bank
- Bank IDP SHOULD very carefully validate parameters sent during this request, namely:
- Verify that `nonce` has not been used before
- Ensure `request_uri` and `redirect_uri` have been whitelisted during Dynamic Client Registration
- Verify signatures of `request_uri` Request Object
- Additional validation requirements and recommendations specified in [OpenID.Core]
- Additional validation requirements and recommendations specified in [[RFC6749] The OAuth 2.0 Authorization Framework]
- Additional OAuth2 best practices specified in [draft-ietf-oauth-security-topics-09]

Source material and relevant modifications:

- [OpenID.Core] OpenID Connect Core 1.0
- Signed and encrypted JWT request object is referenced by the `request_uri` for the /auth endpoint

5.6 4. Authorized API calls

RP is now authenticated can securely consume Bank's APIs (like KYC) with the access_token.

Source material:

- [RFC6750] The OAuth 2.0 Authorization Framework: Bearer Token Usage

5.7 5. Logout and token revocation

Logout can either be initiated by the RP through calling the Revoke endpoint. This invalidates the consent for a specific SeP/Bank/End-User combination.

- End-User MAY logout though the SeP, in which case - SeP logouts the session with the RP. RP in turn calls the Revoke endpoint (specified in **bank.yaml** see Attachments) in the Bank with a valid Refresh Token. This SHOULD revoke the consent given by the End-User for the specific SeP (even if there are multiple active Refresh Tokens).
- End-User SHOULD also have an option to revoke consent in the Internet Banking of the Bank, on a screen where all his consents are listed. This is very similar to [Google's consent management screen]. In this case the Bank generates, signs and encrypts the Logout Token and sends it to the Back-Channel Logout EP of the RP.

Source material and relevant modifications:

- [RFC7009] OAuth 2.0 Token Revocation
- [OpenID.BackChannelLogout] OpenID Connect Back-Channel Logout
- Logout is based on combination of sub and aud rather than iss
- Logout token itself is signed and encrypted JWT

6 Bank API EndPoint overview

The following EndPoints are to be implemented by the Bank. Each of these EndPoints only contains illustratory examples. Please see the referenced OpenAPI documents for full authoritative descriptions of these resources.

6.1 OpenID Configuration Discovery API

Exposed by the **Bank**. Authoritatively described in [**bank.yaml** see Attachments]. **Not authenticated**

OpenID Connect configuration discovery.

GET `/.well-known/openid-configuration`

Response 200 OK:


```
{
  "issuer": "https://idp.example.com",
  "authorization_endpoint": "https://idp.example.com/auth",
  "token_endpoint": "https://idp.example.com/token",
  "userinfo_endpoint": "https://idp.example.com/userinfo",
  "profile_endpoint": "https://idp.example.com/profile",
  "jwks_uri": "https://idp.example.com/.well-known/jwks",
  "registration_endpoint": "https://idp.example.com/register",
  "scopes_supported": ["openid", "kyc.email"],
  "response_types_supported": ["openid", "kyc.email"],
  "response_modes_supported": ["query"],
  "grant_types_supported": ["authorization_code", "refresh_token"],
  "acr_values_supported": ["string"],
  "subject_types_supported": ["pairwise"],
  "id_token_signing_alg_values_supported": ["PS512"],
  "id_token_encryption_alg_values_supported": ["A256GCM"],
  "id_token_encryption_enc_values_supported": ["A256GCM"],
  "userinfo_signing_alg_values_supported": ["PS512"],
  "userinfo_encryption_alg_values_supported": ["A256GCM"],
  "userinfo_encryption_enc_values_supported": ["A256GCM"],
  "profile_signing_alg_values_supported": ["PS512"],
  "profile_encryption_alg_values_supported": ["A256GCM"],
  "profile_encryption_enc_values_supported": ["A256GCM"],
  "request_object_signing_alg_values_supported": ["PS512"],
  "request_object_encryption_alg_values_supported": ["A256GCM"],
  "request_object_encryption_enc_values_supported": ["A256GCM"],
  "token_endpoint_auth_methods_supported": ["private_key_jwt"],
  "token_endpoint_auth_signing_alg_values_supported": ["PS512"],
  "display_values_supported": ["page"],
  "service_documentation": "https://idp.example.com/docs",
  "claims_locales_supported": ["en-CA", "en"],
  "ui_locales_supported": ["cs", "en-US", "en"],
```

```
  "claims_parameter_supported": false,
  "request_parameter_supported": false,
  "request_uri_parameter_supported": true,
  "require_request_uri_registration": false,
  "op_policy_uri": "https://idp.example.com/policy",
  "op_tos_uri": "https://idp.example.com/tos",
  "backchannel_logout_supported": true,
  "claims_supported": ["string"]
}
```


6.2 JWK Keys Discovery API

Exposed by the **Bank** and **RP**. Authoritatively described in [**bank.yaml** see Attachments]. **Not authenticated**

This endpoint returns JSON Web Keys to be used as public keys for verifying OIDC ID Tokens, responses and encrypting requests.

GET `/.well-known/jwks.json`

Response 200 OK:


```
{
  "keys": [
    {
      "alg": "RS256",
      "kid": "1603dfe0af8f4596",
      "key_ops": ["sign", "verify"],
      "use": "sig",
      "kty": "RSA",
      "x5c":
"MIICUDCCAbmgAwIBAgIBADANBgkqhkiG9w0BAQsFADBFMQswCQYDVQQGEwJjejEOMAwGA1
UECAwFUHJhaGExEDAOBgNVBAoMB0V4YW1wbGUxFDASBgNVBAMMC2V4YW1wbGUuY29tMB4XD
TIwMDExNjE2NDExOFoXDTIxMDExNTE2NDExOFowRTELMAkGA1UEBhMCY3oxDjAMBgNVBAgM
BVByYWhhMRAwDgYDVQQKDAdFeGFtcGxlMRQwEgYDVQQDDAtleGFtcGxlLmNvbTCBnzANBgk
qhkiG9w0BAQEFAAOBjQAwgYkCgYEArOEYBRyBhcd6u3phrbU2xvTaBoy6W14CpqqfsBrfsU
suSB+JELBCj3a+zRIvy4EY9cnQbF7cPNxbXdCbGEokAUjIIuVBk/I6XhKRe01vlax82o+eF
fIhUfl7Xb2Bx9U3m98Qbt3WNrv+VYJjjFP8HWSsWCHKCazj+yvozjuFXUsCAwEAAaNQME4w
HQYDVR0OBBYEFN5SUrsStd4aLhBs+MWGRDxLeUP4MB8GA1UdIwQYMBaAFN5SUrsStd4aLhB
s+MWGRDxLeUP4MAwGA1UdEwQFMAMBAf8wDQYJKoZIhvcNAQELBQADgYEAL59fE6itiRrck6
Z7RCjwnOnebQJxpoB/L7TUC/aUIXss40mCviBVKD+Hl4+3sGyp4J2LlzzqhFcPgR9NyxQt0
bkahJGH0UXvZETJe719UA0kGFrPMdt6ujwB6/rafT6TinzXN0lEEGikersTrh3BR9Hjw+v7
nCQ0D5RfuDn6s5s="
    }
  ]
}
```

6.3 HealthCheck API

Exposed by the **Bank**. Authoritatively described in [**bank.yaml** see Attachments]. **Not authenticated**.

This endpoint returns the current Bank liveness and outage information.

GET `/healthcheck`

Response 200 OK:


```
{
  "status": "OK",
  "description": "SONIA API",
  "version": "1.1.0",
  "time": "2020-05-20T11:06:27.767Z",
  "outage_planned_until": null,
  "outage_description": null,
  "poll_interval": 200,
  "details": {
    "dynamic-registration": {
      "status": "OK",
      "description": "Dynamic Registration API",
      "version": "1.1.2",
      "time": "2020-05-20T11:06:27.767Z"
    },
    "oidc-core": {
      "status": "OK",
      "description": "Core OpenID Connect APIs",
      "version": "1.1.7",
      "time": "2020-05-20T11:06:27.767Z"
    },
    "revoke": {
      "status": "OK",
      "description": "OAuth2 Token Revocation API",
      "version": "1.1.2",
      "time": "2020-05-20T11:06:27.768Z"
    },
    "kyc": {
      "status": "OK",
      "description": "KYC API",
      "version": "1.1.0",
      "time": "2020-05-20T11:06:27.768Z"
    }
  }
}
```


6.4 Dynamic Registration API

CRUD over registered clients. Protected by mTLS and Basic HTTP Authorization with pre-shared secret.

Exposed by the **Bank**. Authoritatively described in [**bank.yaml** see Attachments].

POST (create) `/register`

POST endpoint uses **bearer authentication** where bearer token is a JWT signed with a key from `/.well-known/jwks.json` of the RP.

GET (read), **PUT** (update), **DELETE** (remove) `/register/{client_id}`

This endpoints uses **bearer authentication** where bearer token is `registration_access_token` value acquired from the dynamic client registration POST /register endpoint'

Request body (POST and PUT):

```
{
  "redirect_uris": [
    "https://client.example.com/callback",
    "https://rp.example.com/callback"
  ],
  "response_types": ["code"],
  "grant_types": ["authorization_code"],
  "application_type": "web",
  "contacts": ["ve7jtb@example.com", "mary@example.com"],
  "client_name": "My Example",
  "client_type": "bank",
  "client_provider_name": "My Company",
  "tax_number": "45244782",
  "logo_uri": "https://client.example.com/logo.png",
  "client_uri": "https://client.example.com",
  "policy_uri": "https://client.example.com/policy",
  "tos_uri": "https://client.example.com/tos",
  "jwks_uri": "https://client.example.com/my_public_keys.jwks",
  "sector_identifier_uri":
"https://other.example.net/file_of_redirect_uris.json",
  "subject_type": "pairwise",
  "id_token_signed_response_alg": ["PS512"],
  "id_token_encrypted_response_alg": ["A256GCM"],
  "id_token_encrypted_response_enc": ["A256GCM"],
  "userinfo_signed_response_alg": ["PS512"],
  "userinfo_encrypted_response_alg": ["A256GCM"],
  "userinfo_encrypted_response_enc": ["A256GCM"],
  "request_object_signing_alg": ["PS512"],
  "request_object_encryption_alg": ["A256GCM"],
  "request_object_encryption_enc": ["A256GCM"],
  "token_endpoint_auth_method": "private_key_jwt",
  "token_endpoint_auth_signing_alg": ["PS512"],
  "default_max_age": 3600,
  "require_auth_time": false,
  "default_acr_values": ["string"],
  "initiate_login_uri": "https://client.example.com/login",
```

```
  "request_uris": [
    "https://client.example.com/rf.txt",
    "#qpXaRLh_n93TTR9F252ValdatUQvQiJi5BDub2BeznA"
  ],
  "backchannel_logout_uri":
"https://client.example.com/backchannel_logout",
  "scope": "openid profile.name profile.addresses",
  "required_scope": "profile.addresses"
}
```

Response 200 OK:

```
{
  "client_id": "D40D25DB-C330-4331-A191-0A4F6CCD17D8",
  "client_secret": "689A79B8-9EE4-45A9-83B1-4F0130649394",
  "registration_access_token": "44010DFF-F77F-461A-8F3D-1466B24C5CC1",
  "registration_client_uri":
"https://idp.example.com/connect/register/D40D25DB-C330-4331-A191-
0A4F6CCD17D8",
  "client_id_issued_at": 1579263956,
  "redirect_uris": [
    "https://client.example.com/callback",
    "https://rp.example.com/callback"
  ],
  "response_types": ["code"],
  "grant_types": ["authorization_code"],
  "application_type": "web",
  "contacts": ["ve7jtb@example.com", "mary@example.com"],
  "client_name": "My Example",
  "client_type": "bank",
  "client_provider_name": "My Company",
  "tax_number": "45244782",
  "logo_uri": "https://client.example.com/logo.png",
  "client_uri": "https://client.example.com",
  "policy_uri": "https://client.example.com/policy",
  "tos_uri": "https://client.example.com/tos",
  "jwks_uri": "https://client.example.com/my_public_keys.jwks",
  "sector_identifier_uri":
"https://other.example.net/file_of_redirect_uris.json",
  "subject_type": "pairwise",
  "id_token_signed_response_alg": ["PS512"],
  "id_token_encrypted_response_alg": ["A256GCM"],
  "id_token_encrypted_response_enc": ["A256GCM"],
  "userinfo_signed_response_alg": ["PS512"],
  "userinfo_encrypted_response_alg": ["A256GCM"],
  "userinfo_encrypted_response_enc": ["A256GCM"],
  "request_object_signing_alg": ["PS512"],
  "request_object_encryption_alg": ["A256GCM"],
  "request_object_encryption_enc": ["A256GCM"],
  "token_endpoint_auth_method": "private_key_jwt",
  "token_endpoint_auth_signing_alg": ["PS512"],
  "default_max_age": 3600,
```

```
  "require_auth_time": false,
  "default_acr_values": ["string"],
  "initiate_login_uri": "https://client.example.com/login",
  "request_uris": [
    "https://client.example.com/rf.txt",
    "#qpXaRLh_n93TTR9F252ValdatUQvQiJi5BDub2BeznA"
  ],
  "backchannel_logout_uri":
"https://client.example.com/backchannel_logout",
  "scope": "openid profile.name profile.addresses",
  "required_scope": "profile.addresses"
}
```

Response 400 Request invalid:

```
{
  "error": "invalid_redirect_uri",
  "error_description": "Redirect uri must be using https scheme"
}
```

6.5 Authorization API

This GET endpoint is a starting point for OAuth2 and OpenID Connect authorization code flows. This request authenticates the user and returns tokens to the client application as a part of the callback response.

Exposed by the **Bank**. Authoritatively described in [**bank.yaml** see Attachments]. **Not authenticated**

Request:

GET `/auth?redirect_uri=https://rp.example.com/callback&client_id=D40D25DB-C330&response_mode=code&state=1234`

Response redirection:

GET `https://rp.example.com/callback?code=92C74291-C5FE&state=1234`

Error redirection:

GET `https://rp.example.com/callback?error=unauthorized_client&state=1234`

Token exchange API

The token endpoint is used by the client to obtain an access token by presenting its authorization grant or refresh token.

Exposed by the **Bank**. Authoritatively described in [bank.yaml see Attachments]. **Uses client credentials for authentication.**

POST `/token`

Authorization code exchange request body:

Encrypted and signed JWT.

```
{
  "grant_type": "authorization_code",
  "code": "8BFAC1DA-3F94-4BBD-A743-473080FB6073",
  "redirect_uri": "https://rp.example.com/callback"
}
```

Authorization code exchange response 200 OK:

Encrypted and signed JWT.

```
{
  "access_token": "c03e997c-aa96-4b3f-ad0c-98626833145d",
  "token_type": "Bearer",
  "refresh_token": "1f703f5f-75da-4b58-a1b0-e315700e4227",
  "expires_in": 6000,
  "id_token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZ
SI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36PO
k6yJV_adQssw5c"
}
```

Refresh token exchange request body:

Encrypted and signed JWT.

```
{
  "grant_type": "refresh_token",
  "scope": "openid profile.name profile.addresses",
  "refresh_token": "A9B54609-FF9E-42F0-B089-89E1E73E224F",
  "redirect_uri": "https://rp.example.com/callback"
}
```

Refresh token exchange response 200 OK:

Encrypted and signed JWT.

```
{
```

```
  "access_token": "c03e997c-aa96-4b3f-ad0c-98626833145d",
  "token_type": "Bearer",
  "expires_in": 6000,
  "id_token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZ
SI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36PO
k6yJV_adQssw5c"
}
```


Response 400 Request invalid:


```
{
  "error": "invalid_request",
  "error_description": "The request is missing a required parameter"
}
```


6.6 TokenInfo API

The introspection endpoint is an OAuth 2.0 endpoint that takes a parameter representing an OAuth 2.0 token and returns a JSON representing the meta information surrounding the token, including whether this token is currently active.

Exposed by the **Bank**. Authoritatively described in [**bank.yaml** see Attachments]. **Uses client credentials or a client access token for authentication.**

POST `/token-info`

Request body:


```
{
  "token": "WwVEraxkI7KbtP31wD3XSpZKqGpsLiXg",
  "token_type_hint": "refresh_token"
}
```


Response:


```
{
  "active": true,
  "scope": "openid profile.addresses",
  "client_id": "d1bdc32e-1b06-4609-9f60-073685267f88",
  "token_type": "access_token",
  "exp": 1419356238,
  "iat": 1419350238,
  "sub": "25657805-66d4-4707-980a-f12429f17592",
  "aud": "https://rp.example.com/resource",
```

```
  "iss": "https://server.example.com/"
}
```

6.7 Token revocation API

Token revocation endpoint. Revokes access and refresh tokens. Revoking a refresh token effectively cancels the "session".

Exposed by the **Bank**. Authoritatively described in [**bank.yaml** see Attachments]. **Uses client credentials or a client access token for authentication**.

POST `/revoke`

Request body:

```
{
  "token": "WwVEraxkI7KbtP31wD3XSpZKqGpsLiXg",
  "token_type_hint": "refresh_token"
}
```

6.8 UserInfo API

The UserInfo Endpoint is an OAuth 2.0 Protected Resource that returns Claims about the authenticated End-User.

Exposed by the **Bank**. Authoritatively described in [**userInfo_bank.yaml** see Attachments]. **Uses client credentials or a client access token for authentication.**

To obtain the required claims, RP will use an Access Token obtained from the authentication flow defined as OpenID Connect Authentication. These Claims are normally represented by a JSON object that contains a collection of name and value pairs for the Claims.

Verified Data Representation extension for `/userinfo` resource:

This extension to OpenID Connect wants to ensure that IDPs cannot mix up verified and unverified Claims and incidentally process unverified Claims as verified Claims.

The representation proposed therefore provides the IDP with the verified Claims within a container element `verified_claims`. This container is composed of the verification evidence related to a certain verification process and the corresponding Claims about the End-User which were verified in this process.

Verification elements contains the information about the process conducted to verify a person's identity and bind the respective person data to a user account.

Preliminary arrangements for the use of described resources, list of source materials:

- [OpenID.Core] OpenID Connect Core 1.0
- Signed and encrypted JWTs are used for any `/userinfo` calls and responses
- In SONIA's IDP relationship the UserInfo Claims MUST be returned as the signed and encrypted JSON object as required by definition during SONIA Client Registration to IDP.
- In Service Provider's SONIA relationship the UserInfo Claims MUST be returned as the members of a JSON object unless a signed or encrypted response was requested during Client Registration.
- The sub (subject) Claim MUST always be returned in the UserInfo Response.
- If a Claim is not returned, that Claim Name SHOULD be omitted from the JSON object representing the Claims; it SHOULD NOT be present with a null or empty string value.
- The sub Claim in the UserInfo Response MUST be verified to exactly match the sub Claim in the ID Token; if they do not match, the UserInfo Response values MUST NOT be used.
- The UserInfo Endpoint MUST return a content-type header to indicate which format is being returned and if the response body is a text JSON object; the response body SHOULD be encoded using UTF-8.
- If the UserInfo Response is signed and/or encrypted, then the Claims are returned in a JWT and the content-type MUST be application/jwt. The response MAY be encrypted without also being signed. If both signing and encryption are requested, the response MUST be signed then encrypted, with the result being a Nested JWT.

- [OpenID.IdentityAssurance] OpenID Connect for Identity Assurance 1.0
    - The `txn` Claim as defined in [RFC8417](https://tools.ietf.org/html/rfc8417) is used in the context of SONIA data exchange to build audit trails across the parties involved in an OpenID Connect transaction. Claim txn is always REQUIRED in the userinfo response content.
    - This arrangement introduces the possibility for the bank to separate the verified Claims within a container element `verified_claims`. This container is composed of the verification evidence related to a certain verification process and the corresponding Claims about the End-User which were verified in this process.

- Implementations MUST ignore any sub-element not defined in this specification or extensions of this specification.
- In the case of this definition verification element MUST consists of the following elements:
- `trust_framework`: REQUIRED. String determing the trust framework governing the identity verification process. For example, the value of `cz_aml` for verification according to the Czech AML law.
- `time`: OPTIONAL. Time stamp in ISO 8601:2004 [ISO8601-2004] YYYY-MM-DDThh:mm:ss±hh format representing the date and time when identity verification took place.
- `verification_process`: REQUIRED. In the case of this specification, the verification process shall include the tax number of the relevant bank where the initial physical verification of the client took place.
- `claims`: The claims element contains the claims about the End-User that were verified during the defined verification process.

- Verified Claims can be requested on the level of individual Claims about the End-User by utilizing the claims parameter as defined in Section 5.5. of the OpenID Connect specification [OpenID].

GET `/userinfo`

Response:

```
{
  "sub": "23f1ac00-5d54-4169-a288-794ae2ead0c4",
  "txn": "31470547-0f7f-4794-acb0-d959f5a711a5",
  "verified_claims": {
    "verification": {
      "trust_framework": "cz_aml",
      "time": {},
      "verification_process": "45244782"
    },
    "claims": {
      "name": "Jan Novák",
      "given_name": "Jan",
      "family_name": "Novák",
      "title_prefix": "Ing.",
      "gender": "male",
      "birthdate": "1970-08-01"
    }
  },
  "nickname": "Fantomas",
  "prefered_username": "JanN",
  "email": "j.novak@email.com",
  "email_verified": false,
```

```
  "zoneinfo": "Europe/Prague",
  "locale": "cs_CZ",
  "phone_number": "+420123456789",
  "phone_number_verified": true,
  "updated_at": 1568188433000
}
```

6.9 Profile API

The Know Your Customer Endpoint is an OAuth 2.0 Protected Resource that returns identification data of the currently authenticated End-User.

Exposed by the **Bank**. Authoritatively described in [**profile_bank.yaml** see Attachments]. **Uses client credentials or a client access token for authentication.**

To obtain the required claims, RP will use an Access Token obtained from the authentication flow defined as OpenID Connect Authentication. These Claims are normally represented by a JSON object that contains a collection of name and value pairs for the Claims.

Verified Data Representation extension for `/profile` resource:

This extension to OpenID Connect wants to ensure that IDPs cannot mix up verified and unverified Claims and incidentally process unverified Claims as verified Claims.

The representation proposed therefore provides the IDP with the verified Claims within a container element `verified_claims`. This container is composed of the verification evidence related to a certain verification process and the corresponding Claims about the End-User which were verified in this process.

Verification elements contains the information about the process conducted to verify a person's identity and bind the respective person data to a user account.

Preliminary arrangements for the use of described resources, list of source materials:

- [OpenID.Core] OpenID Connect Core 1.0
- Signed and encrypted JWTs are used for any `/profile` calls and responses
- In SONIA's IDP relationship the Profile Claims MUST be returned as the signed and encrypted JSON object as required by definition during SONIA Client Registration to IDP.

- In Service Provider's SONIA relationship the Profile Claims MUST be returned as the members of a JSON object unless a signed or encrypted response was requested during Client Registration.
- The sub (subject) Claim MUST always be returned in the Profile Response.
- If a Claim is not returned, that Claim Name SHOULD be omitted from the JSON object representing the Claims; it SHOULD NOT be present with a null or empty string value.
- The sub Claim in the Profile Response MUST be verified to exactly match the sub Claim in the ID Token; if they do not match, the Profile Response values MUST NOT be used.
- The Profile Endpoint MUST return a content-type header to indicate which format is being returned and if the response body is a text JSON object; the response body SHOULD be encoded using UTF-8.
- If the Profile Response is signed and/or encrypted, then the Claims are returned in a JWT and the content-type MUST be application/jwt. The response MAY be encrypted without also being signed. If both signing and encryption are requested, the response MUST be signed then encrypted, with the result being a Nested JWT.

- [OpenID.IdentityAssurance] OpenID Connect for Identity Assurance 1.0
    - The `txn` Claim as defined in [RFC8417] is used in the context of SONIA data exchange to build audit trails across the parties involved in an OpenID Connect transaction. Claim txn is always REQUIRED in the userinfo response content.
    - This arrangement introduces the possibility for the bank to separate the verified Claims within a container element `verified_claims`. This container is composed of the verification evidence related to a certain verification process and the corresponding Claims about the End-User which were verified in this process.
    - Implementations MUST ignore any sub-element not defined in this specification or extensions of this specification.
    - In the case of this definition verification element MUST consists of the following elements:
    - `trust_framework`: REQUIRED. String determing the trust framework governing the identity verification process. For example, the value of `cz_aml` for verification according to the Czech AML law.
    - `time`: OPTIONAL. Time stamp in ISO 8601:2004 [ISO8601-2004] YYYY-MM-DDThh:mm:ss±hh format representing the date and time when identity verification took place.
    - `verification_process`: REQUIRED. In the case of this specification, the verification process shall include the tax number of the relevant bank where the initial physical verification of the client took place.
    - `claims`: The claims element contains the claims about the End-User that were verified during the defined verification process.

- Verified Claims can be requested on the level of individual Claims about the End-User by utilizing the claims parameter as defined in Section 5.5. of the OpenID Connect specification [OpenID].

GET `/profile`

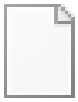
Response:


```
{
  "sub": "23f1ac00-5d54-4169-a288-794ae2ead0c4",
  "txn": "6941683f-c6ee-410c-add0-
d52d63091069:openid:profile.name:profile.address",
  "verified_claims": {
    "verification": {
      "trust_framework": "cz_aml",
      "time": {},
      "verification_process": "45244782"
    },
    "claims": {
      "given_name": "Jan",
      "family_name": "Novák",
      "gender": "male",
      "birthdate": "1970-08-01",
      "addresses": [
        {
          "type": "PERMANENT_RESIDENCE",
          "street": "Olbrachtova",
          "buildingapartment": "1929",
          "streetnumber": "62",
          "city": "Praha",
          "zipcode": "14000",
          "country": "CZ"
        }
      ],
      "idcards": [
        {
          "type": "ID",
          "description": "Občanský průkaz",
          "country": "CZ",
          "number": "123456789",
          "valid_to": "2023-10-11",
          "issuer": "Úřad městské části Praha 4",
          "issue_date": "2020-01-28"
        }
      ]
    }
  },
  "given_name": "Jan",
  "family_name": "Novák",
  "middle_name": "",
  "gender": "male",
```

```
  "birthdate": "1970-08-01",
  "birthnumber": "7008010147",
  "age": 50,
  "majority": true,
  "date_of_death": null,
  "birthplace": "Praha 4",
  "primary_nationality": "CZ",
  "nationalities": [
    "CZ",
    "AT",
    "SK"
  ],
  "maritalstatus": "MARRIED",
  "email": "J.novak@email.com",
  "phone_number": "+420123456789",
  "pep": false,
  "limited_legal_capacity": false,
  "addresses": [
    {
      "type": "PERMANENT_RESIDENCE",
      "street": "Olbrachtova",
      "buildingapartment": "1929",
      "streetnumber": "62",
      "city": "Praha",
      "zipcode": "14000",
      "country": "CZ",
      "ruian_reference": "186GF76"
    }
  ],
  "idcards": [
    {
      "type": "ID",
      "description": "Občanský průkaz",
      "country": "CZ",
      "number": "123456789",
      "valid_to": "2023-10-11",
      "issuer": "Úřad městské části Praha 4",
      "issue_date": "2020-01-28"
    }
  ],
  "paymentAccounts": [
    "CZ0708000000001019382023"
  ],
  "updated_at": 1568188433000
}
```

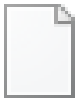
7 Attachments

Open API specification that describes APIs to be implemented by bank IDPs:



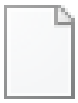
bank.yaml

Open API specification that describes API for Profile Endpoint (Know Your Customer) that returns profile Claims about the authenticated End-User:



profile_bank.yaml

Open API specification that describes API for the UserInfo Endpoint is an OAuth 2.0 Protected Resource that returns Claims about the authenticated End-User:



userInfo_bank.yaml