Tech Spec - M-Pin v3.3.0 Relying Party Service

Contents

General

This document describes the technical details of the Relying Party Service (RPS). It is targeted at Technical Staff - Developers, DevOps and QA personnel.

Acronyms

Acronym	Description
RPS	Relying Party Service
RPA	Relying Party Application
D-TA	Distributed Trusted Authority
ACL	Access Control List
OTP	One-Time Password
OTT	One-Time Token

System Overview

The M-Pin System consists of two groups of Services - Customer-hosted Services and CertiVox-hosted Services.

Customer-hosted Services

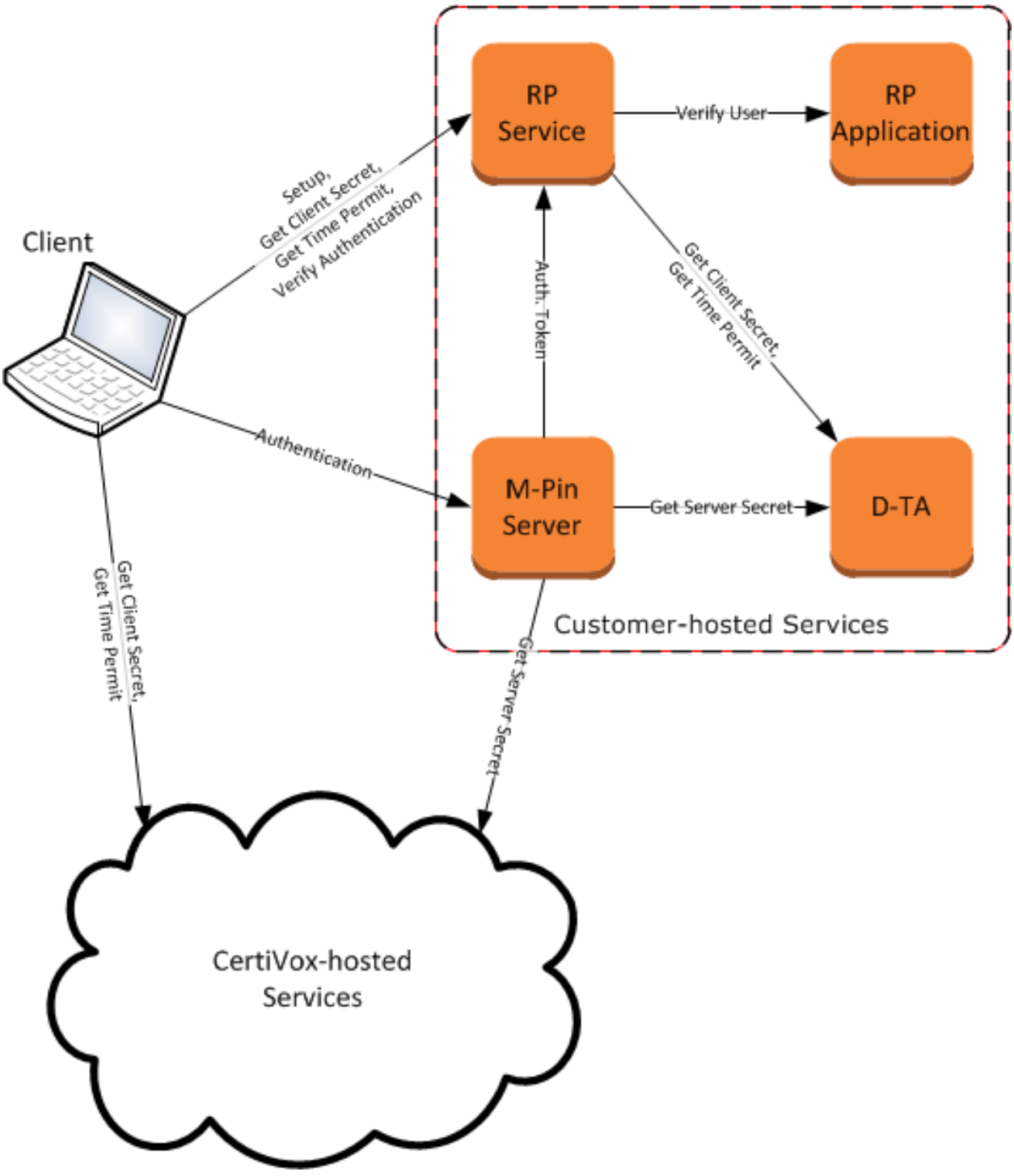
- **M-Pin Server** - The Authentication Server against which End-Users authenticate. The Client (PIN Pad) performs the "M-Pin Dance" against the M-Pin server in order to authenticate an End-User.
- **D-TA** - Distributed Trusted Authority Service, managed by the Customer. Responsible to generate Client and Server Secret Shares as well as Time Permit Shares
- **RPS** - Relying Party Service. Implements the M-Pin Protocol and Workflows on behalf of the Customer. The RPS serves as an abstraction layer between the specific implementation of the M-Pin Protocol and the RPA.
- **RPA** - Relying Party Application. This is the Application to which end-users are authenticated through the M-Pin System. This application is implemented/managed by the Customer and is strictly specific for each different Customer.

CertiVox-hosted Services

- **D-TA** - Distributed Trusted Authority Service, managed by CertiVox. Responsible for generating Client and Server Secret Shares as well as Time Permit Shares.
- **D-TA Proxy** - Proxies request to the D-TA, validating RPS signatures. The D-TA Proxy is public-facing, while the D-TA should not be publicly accessible.
- **Time Permits Service** - A service responsible to publish Time Permits to an online storage (CDN), such as AWS S3.
- **Registration Service** - A service that handles new Customer registration.

RPS

The RPS resides on the Customer premises - it might be a physical machine hosted by the Customer or an M-Pin dedicated virtual machine on AWS, purchased via the AWS Marketplace. The RPS serves as an abstraction layer between the M-Pin protocol, workflow and crypto, and between the RPA. The RPS provides an API to the RPA, for the operations that the RPA is responsible for, and which cannot be performed by the RPS itself.



System Architecture

The RPS is a Python written service, based on the Tornado framework, i.e. it serves requests in a single thread utilizing non-blocking IO operations.

The RPS should not be exposed to the public Internet, but it should be accessible by the Client either through the RPA or through a dedicated public facing proxy (e.g. Nginx). For this reason, all the "public" RPS API requests start with a predefined prefix, which is set by default to */rps*. Any request that starts with that prefix should be redirected to the RPS by the public facing service - the RPA or a proxy. RPS API requests that do not start with the prefix should be accessible only from the Customer's private network, and more specifically, by the RPA.

Additionally, the RPS will implement an ACL, so only authorized machines will be able to make requests to its API

RPS Scalability and High Availability

By default, the RPS will store all of its work data in its memory storage. This approach, although simpler and more secure, introduces a problem when the RPS needs to be scaled or made highly available. For this reason the RPS supports an option to use Redis as a work storage and have several RPS instances work together, behind a load balancer. Since the actual state is stored on Redis, the RPS instances become stateless and mutually replaceable. For more information regarding the configuration for Redis storage, see Configuration.

RPS API

Client Initialization

GET /rps/clientSettings

Called by the Client (PIN Pad), through a proxy, to obtain the setting it should use. Most of the settings are Service Endpoints and Customer details.

Parameters: <none>

Data: <none>

Response: 200 OK on success, 4xx otherwise

Response Data:

```
{
        "mpinAuthServerURL": <mpin-auth-endpoint>,
        "timePermitsURL": <customer-time-permit-endpoint>,
        "timePermitsStorageURL": <certivox-time-permit-cache-url>,
        "authenticateURL": <auth-validation-endpoint>,
        "timePermitsStorageURL": <tp-storage-url>,
        "certivoxURL": <certivox-services-url>,
        "mobileAuthenticateURL": <mobile-auth-validation-endpoint>,
        "signatureURL": <signature-endpoint>,
```

```
        "requestOTP": <request-otp>,
        "setupDoneURL": <setup-done-endpoint>,
        "successLoginURL": <successful-login-url>,
        "accessNumberURL": <access-number-endpoint>,
        "accessNumberDigits": <access-number-digits>,
        "seedValue": <seed-value>,
        "registerURL": <user-register-endpoint>,
        "identityCheckRegex": <identity-check-regex>,
        "useWebSocket": <use-web-sockets>,
        "setDeviceName": <set-device-name>,
        "appID": <app-id>
}
```

<*mpin-auth-endpoint*> - M-Pin Server WebSocket or REST endpoint for end-user authentication. Example: "ws://192.168.10.138:8002"

<*customer-time-permit-endpoint*> - Endpoint on which requests for Customer Time Permit Share should be done. Example: "http://192.168.10.138:8005/rps/timePermit"

<*certivox-time-permit-cache-url*> - Base URL of cache storage for CertiVox Time Permit Share.

<*auth-validation-endpoint*> - Endpoint implemented by the RPA for authentication validation. Example: "/mpinAuthenticate".

<*tp-storage-url*> - Base URL for the Time Permit Storage. This is the storage where CertiVox API publishes its pre-generated shares of Time Permits. Example: "https://s3-eu-west-1.amazonaws.com/freetier-timeperbit-bucket-qa-v3"

<*certivox-services-url*> - Base URL for the CertiVox-hosted Services. Example: "https://m-pinapi-qa-v03.certivox.org/v0.3"

<*mobile-auth-validation-endpoint*> - Endpoint for Mobile Authentication. Example: "http://192.168.10.138:8005/rps/authenticate"

<*signature-endpoint*> - Endpoint for requesting Relying Party Signature, further used to get Client Secret Share. Example: "http://192.168.10.138:8005/rps/signature"

<*request-otp*> - Indicates (true or false) whether One-Time Password should be generated on successful authentication. Used for the Arion Server.

<*setup-done-endpoint*> - Endpoint for the setupDone request. Example: "http://192.168.10.138:8005/rps/setupDone"

<*successful-login-url*> - URL that the PIN Pad should load after a successful end-user authentication.

<*access-number-endpoint*> - Endpoint for Access Number Generation request. Example: "http://192.168.10.138:8005/rps/getAccessNumber"

<*access-number-digits*> - Number of digits for the requested Access Number. Example (and Default): 7

<*seed-value*> - Ephemeral hex-encoded value, used as a seed for further random number generation.

<*user-register-endpoint*> - Endpoint for initiating end-user registration flow. Example: "http://192.168.10.138:8005/rps/user"

<*identity-check-regex*> - A regular expression used to verify the format of a new end-user identity. Example: "[0-9a-zA-Z]+"

<*use-web-sockets*> - Indicates (true or false) whether the PIN Pad should make an attempt to use WebSockets when authenticating an end-user against the M-Pin Server. If not specified, the default is *true*.

<*set-device-name*> - Indicates (true or false) whether the client should display to and enable the user to set a friendly name for the device. If so, the client should send the device name with the PUT /rps/user request.

<*app-id*> - Customer specific Application ID, received upon Customer/Application registration for the M-Pin Service.

Setup

The purpose of the Setup process is as follows:

1. Verify the user identity
2. Get the two Shares of the Client Secret and combine them.
3. Extract the user PIN Code from the Client Secret to form the M-Pin Token.
4. Store the M-Pin Token on the Client machine.

PUT /rps/user[/<mpin-id>]

This request is made by the Client (through a proxy) to initiate the Setup flow for an end-user, or to re-start it. When the flow is initially started, the request is made without the optional */<mpin-id>*. In this case the RPS generates a new <mpin-id> which is returned in the response data. The <mpin-id> is a hex-encoded JSON structure of the following form:

```
{
    "issued": <date-time>,
    "userID": <user-identity>,
    "mobile": <1|0>,
    "salt": <64-bit-hex-encoded-random-number>
}
```

In certain cases, the Setup flow might need to be re-started for an already generated <mpin-id>. In this case the optional */<mpin-id>* in the request is appended, as well as the *regOTT* parameter in the request data.

During this request, the RPS makes a POST /verify request to the RPA, sending to it an *activateKey*. During the POST /verify request the RPA could validate whether the provided <user-id> is a valid system user, could initiate a user verification procedure (e.g. sending verification e-mail) or use the optional <user-data> for additional operations that are required to validate the end-user. As a result, the RPA might indicate to the RPS whether the user should be made active immediately, or the RPS should wait for the user identity verification to be completed. When the identity verification is completed, the RPA should make a POST /user/<mpin-id> request, providing the *activateKey*, to be able to proceed further with the flow (see M-Pin Setup flow).

The actual RPA endpoint for the POST /verify request is configurable (see Configuration).

Parameters: <none>

Data:

```
{
        "userId": <user-id>,
        "deviceId": <device-id>,
        "mobile": <0|1>,
        "regOTT": <registration-ott>,
        <user-data>
}
```

<*user-id*> - The string identifying the end-user. It might be the end-user e-mail address or any other system-unique string.

<*device-id*> - (Optional) Depending on the Customer' preferences, the Client might provide a friendly name describing the device from which the request is coming. This name will be further forwarded to the RPA so it can attach it to the end-user information that it stores.

<*mobile*> - Indicates (1 or 0) whether the flow is carried out by the mobile client (1), or not (0).

<*registration-ott*> - (Optional) In case an already started setup flow should be re-started, this should be the original <registration-ott> that was returned in the initial response.

<*user-data*> - (Optional) The Customer might pass to the PIN Pad Client some customer-specific data that would help to verify the end-user in the RPA. This data is an opaque, passed to the RPA in the POST /verify request.

Response: 200 OK on success, 4xx otherwise

Response Data:

```
{
        "expireTime": <utc-formatted-expiration-time>,
        "active": <true|false>,
        "regOTT": <registration-ott>,
        "nowTime": <utc-formatted-current-time>,
        "mpinId": <mpin-id>
}
```

<*utc-formatted-expiration-time*> - Expiration time for the user setup flow in case the RPS should wait for the user verification to be completed.

<*active*> - Indicates (true or false) whether the user has been made active already, and no further user verification is required.

<*registration-ott*> - A reference number identifying the setup process for the given <mpin-id>. This number is valid only until the flow is complete or expired, and serves as a type of OTT.

<*utc-formatted-current-time*> - The current system time.

<*mpin-id*> - The formed hex-encoded M-Pin ID for the end-user identified by <user-id>. If */<mpin-id>* is appended in the request, the same <mpin-id> will be returned here. Otherwise the RPS generates a new <mpin-id>.

POST /user/<mpin-id>

This request is made by the RPA when the end-user identity verification is complete.

<mpin-id> is the hex-encoded M-Pin ID

Parameters: <none>

Data:

```
{
        "activateKey": <activation-key>
}
```

<*activation-key*> - The activation key provided by the RPS during the POST /verify request to the RPA.

Response: 200 OK on success, 4xx otherwise

Response Data: <none>

GET /rps/signature/<mpin-id>?regOTT=<registration-ott>

This request is made by the Client (through a proxy) and serves several purposes:

- To obtain the Customer share of the Time Permit. As part of this request the RPS will get the Time Permit share from the Customer D-TA and will return it in the response.
- To obtain the parameters that should be used to request the CertiVox share of the Time Permit. Those parameters include a unique signature without which the CertiVox D-TA will not fulfill the request.
- To generate obfuscated (hashed) M-Pin ID, which is further used by the CertiVox-hosted Services to identify the end-user.

<mpin-id> is the hex-encoded M-Pin ID.

Parameters: regOTT=<registration-ott>

<*registration-ott*> - The registration reference number (OTT) provided in the response of PUT /rps/user[/<mpin-id>]

Data: <none>

Response: 200 OK on success, 4xx otherwise

Response Data:

```
{
        "clientSecretShare": <client-secret-share>,
        "params": <client-secret-request-params>
}
```

<*client-secret-share*> - Client Secret Share from the Customer-hosted D-TA Service.

<*client-secret-request-params*> - Parameters (including signature) that should be used to request the other Client Secret Share from the CertiVox-hosted D-TA Service. Those parameters are formatted as

request query parameters in the
form: `app_id=<app_id>&hash_mpin_id=<hash_mpin_id>&\nexpires=<expires>&mobile=<mobile>&signature=<signature>`

POST /rps/setupDone/<mpin-id>

This request is made by the Client (through a proxy) to announce that the end-user has finalized the setup process, providing his/her PIN Number. Currently no action is taken by the RPS during this request.

<*mpin-id*> is the hex-encoded M-Pin ID

Parameters: <none>

Data: <none>

Response: 200 OK on success, 4xx otherwise

Response Data: <none>

Authentication

The Authentication of a user starts with getting a Time Permit for that user identity for the current date. The Time Permit is combined from two shares - one from the Customer D-TA and one from CertiVox D-TA. For efficiency, CertiVox's share of the Time Permit is cached in two levels - on the Client side (1) and/or on a dedicated cache storage (2). The Client should first of all check whether there is a TP for the current date cached on the local machine. If so, it should use it. Otherwise, it should make a request to the second cache level - the TP storage. If the TP is retrieved from there, it should be cached locally and used until the end of the day. If there's no TP on the cache storage, a request should be made to CertiVox's D-TA, which will serve the TP, will generate new TP's for the next few days and will store them to the cache storage.

GET /rps/timePermit/<mpin-id>

This request is made by the Client (through a proxy) to obtain the Customer Share of the Time Permit for the given <*mpin-id*>. The RPS makes a request to the Customer-hosted D-TA Service to obtain the Time Permit Share and returns the result to the Client. Prior to sending the request to the D-TA, the RPS will make GET /permitUser?mpin_id=<mpin-id> request to the RPA, which might revoke the access to the service for the specified user. The RPA should respond with 200 OK, if the RPS should proceed with the time permit request, or with 403 if the user is revoked. The actual RPA endpoint for the GET /permitUser request is configurable (see Configuration). If it is not configured, the RPS will not make this request and will assume that no user should be revoked by the RPA.

Additional purpose of this request is to provide the Client with the current date and the location of the cache storage for CertiVox's share of the Time Permit. The Client should use those in order to check whether the CertiVox's TP share is stored locally, or on the cache storage, or it should be requested from CertiVox's D-TA.

As part of this request the RPS will also generate and return in the response a unique signature which should further be used in the request for the second Time Permit Share from the CertiVox D-TA Service.

<*mpin-id*> is the hex-encoded M-Pin ID.

Parameters: <none>

Data: <none>

Response: 200 OK on success, 4xx otherwise

Response Data:

```
{
        "date": <current-date>,
        "message": <response-message>,
        "version": <mpin-version>,
        "timePermit": <time-permit-share>,
        "storageId": <storage-id>,
        "signature": <signature>
}
```

<*current-date*> - The current date in days since the Epoch (1 January 1970, UTC).

<*response-message*> - A response message, usually "M-Pin Time Permit Generated".

<*mpin-version*> - The version of the M-Pin protocol, currently "0.3".

<*time-permit-share*> - Time Permit Share from the Customer-hosted D-TA Service.

<*storage-id*> - ID under which the CertiVox Time Permit Share is possibly stored on the cache storage. This ID is appended to the base Storage URL received in the Client Settings, to form the full URL for accessing the cache storage.

<*signature*> - A unique signature that should be used in subsequent request for the CertiVox share of the Time Permit.

PUT /token

This request is made by the M-Pin Server as a result of authentication attempt. The server passes a token which indicates whether the authentication was successful or not, as well as a reference number (serving as an OTT) for the authentication attempt. This reference number is also provided to the Client, which further posts it to the POST /authenticate request to verify the authentication. This call is assumed to be performed over a secured SSL channel.

Parameters: <none>

Data:

```
{
        "token": <auth-token>,
        "authOTT": <authentication-ott>,
        "signature": <signature>
}
```

<*auth-token*> - A Plain Authentication Token in the form of JSON object. Example:

```
{
        "WID": "0",
```

```
        "expires": "2014-03-18T09:00:33Z",
        "pinError": 0,
        "successCode": 0,
        "OTP": "0",
        "mpin_id": <mpin-id-plain>,
        "mpin_id_hex": <mpin-id-hex>,
        "pinErrorCost": 0
}
```

<*authentication-ott*> - Authentication reference number (OTT)

<*signature*> - HMAC signature over the next fields: `<mpin_id_hex><successCode><expires><WID><OTP><authOTT>`. The validity of this signature is verified by the RPS.

Response: 200 OK on success, 4xx otherwise

Response Data: <none>

POST /authenticate

This request is made by the RPA to validate successful authentication for an end-user. The RPA should provide the authentication reference number <*authentication-ott*>, in order for the RPS to validate the authentication result.

Parameters: <none>

Data:

```
{
        "authOTT": <authentication-ott>
}
```

<*authentication-ott*> - The authentication reference number (OTT) provided in the final result of the M-Pin dance between the Client and the M-Pin Server

Response:

200 Authentication successful - on success;

401 Wrong PIN - on unsuccessful authentication;

410 Wrong PIN - after N unsuccessful authentication attempts.

408 Expired authentication request - the authOTT is invalid or expired.

Response Data:

```
{
        "status": <response-status>,
        "message": <response-message>,
        "userId": <user-id>,
        "mpinId": <mpin-id>
}
```

<response-status> - The status of the authentication, either 200, 401 or 410. The same as the HTTP status code

<response-message> - Response message for the authentication. The same as the message in the HTTP status - "Authentication successful" or "Wrong PIN"

<user-id> - The identity of the user being authenticated.

<mpin-id> - The hex-encoded M-Pin ID of the user being authenticated.

Mobile Authentication

POST /loginResult

This request is made by the RPA to inform the RPS for a potential login restriction for the end-user that is currently being authenticated, and to provide optional logout data. This request is mostly useful for the Mobile authentication flow, where the Mobile App should receive as a response to the POST /rps/authenticate request a valid status, as well as the required data for a subsequent logout operation. If the *waitForLoginResult* configuration option is set to *True*, then the RPS will wait for this request before returning a response to POST /rps/authenticate, otherwise it will return the response right after returning response to the POST /authenticate request from the RPA.

Parameters: <none>

Data:

```
{
        "status": <status>,
        "authOTT": <authentication-ott>,
        "logoutURL": <logout-endpoint>,
        "logoutData": <logout-data>
}
```

<status> - If the RPA doesn't band the user from logging-in, the status should be 200. Otherwise the status might be 401 or 410. See valid response of POST /rps/authenticate

<authentication-ott> - Authentication reference number (OTT)

<logout-endpoint> - (Optional) Endpoint to which the Mobile App should make request to logout the end-user. This endpoint might be alternatively set in the Configuration file.

<logout-data> - (Optional) Any data that the Mobile App should provide while making request to the *<logout-endpoint>*. This data might be a JSON object as well. If the *logoutData* is not present, then the Mobile App will not present to the end-user the option to logout.

Response: 200 OK on success, 4xx otherwise

Response Data: <none>

POST /rps/getAccessNumber

This request is made by the Client (through a proxy) to obtain the Access Number that is required for authentication through the Mobile Client.

Parameters: <none>

Data: <none>

Response: 200 OK on success, 4xx otherwise

Response Data:

```
{
        "localTimeStart": <access-number-expiration-start>,
        "ttlSeconds": <access-number-expiration-in-seconds>,
        "localTimeEnd": <access-number-expiration-end>,
        "webOTT": <web-ott>,
        "accessNumber": <access-number>
}
```

<*access-number-expiration-in-seconds*> - Access Number expiration period in seconds.

<*access-number-expiration-start*> - Start time of Access Number expiration in seconds since the Epoch.

<*access-number-expiration-end*> - End time of Access Number expiration in seconds since the Epoch.

<*web-ott*> - Reference number (serves as OTT) for the mobile authentication

<*access-number*> - The Access Number.

POST /rps/accessnumber

This request is made by the Client (through a proxy) to check whether an end-user has authenticated through the Mobile App. The Client provides the *webOTT* to refer to the relevant mobile authentication transaction. When an end-user has authenticated successfully through the Mobile Client, an *authOTT* is returned, so subsequent authentication requests to the RPA might be executed.

Parameters: <none>

Data:

```
{
        "webOTT": <web-ott>
}
```

Response:

200 OK - Successful Authentication.

401 Unauthorized - User has not been authorized successfully yet.

Response Data:

Sent only with 200 OK response

```
{
        "authOTT": <authentication-ott>
}
```

<*authentication-ott*> - The authentication reference number (OTT) provided in the final result of the M-Pin dance between the Client and the M-Pin Server.

POST /rps/authenticate

This request is made by the Mobile Client (through a proxy) to authenticate an end-user. The Mobile Client passes the *authOTT* that was returned by the M-Pin Server as a result of the M-Pin Dance. The RPS obtains the Authentication Token using the *authOTT* and verifies the result of the authentication.

Parameters: <none>

Data:

```
{
        "mpinResponse": {
                "authOTT": <authentication-ott>,
                "version": <mpin-version>,
                "type": "PASS2"
        }
}
```

<*authentication-ott*> - The authentication reference number (OTT) provided in the final result of the M-Pin dance between the Client and the M-Pin Server.

<*mpin-version*> - The version of the M-Pin protocol, currently "0.3".

Response:

200 Authentication successful - on success;

401 Wrong PIN - on unsuccessful authentication;

410 Wrong PIN - after N unsuccessful authentication attempts.

408 Expired authentication request - the authOTT is invalid or expired.

Response Data: <none>

Expected API for the RPA

The RPA is the only part in the M-Pin system that is strictly specific to each Customer and implements the logic of the specific Web Application. In order for this Web Application to serve as RPA, it should implement the bellow RESTful endpoints. Note that the actual URL's for the endpoints are customizable, and therefore example endpoint names are shown in the brackets. The actual endpoint URL's should be configured in the RPS (see Configuration).

End-user Verification Callback Endpoint (POST /verify)

This request is made as part of the end-user registration and activation flow. The request is made by the RPS to the RPA to either verify the end-user identity in-place, or to initiate end-user identity verification process. If the RPA is able to verify the identity in-place, then it should return 200 OK with response data `{ "forceActivate": true }`. If a verification process has been started (e.g. via sending a verification e-mail to the end-user), then the RPA should make a POST /user/<mpin-id> request to the RPS when the user identity has been verified, providing the same *activateKey* that was received in this request.

Parameters: <none>

Data:

```
{
        "activateKey": <activate-key>,
        "mpinId": <mpin-id>,
        "mobile": <0|1>,
        "userId": <user-id>,
        "expireTime": <utc-formatted-expiration-time>,
        "resend": <true|false>,
        "deviceName": <device-name>
        "userData": <user-data>
}
```

<*activation-key*> - An activation reference number, identifying the specific end-user verification and activation process.

<*mpin-id*> - The hex-encoded M-Pin ID of the user being registered/set-up.

<*mobile*> - Indicates (1 or 0) whether the flow is carried out by the mobile client (1), or not (0).

<*user-id*> - The identity of the user being registered/set-up.

<*utc-formatted-expiration-time*> - Expiration time for the user setup flow in case the RPS should wait for the user verification to be completed.

<*resend*> - Indicates (true or false) whether the setup flow for that user was just started, or re-started. If the end-user failed to receive the a verification e-mail (for instance) and requests to re-send the e-mail, this flag will be `true`.

<*device-name*> - A friendly name describing the device from which the user is trying to register. The RPA might associate and save this name with the *mpin-id* in order to be able later to recognize the *mpin-id* given the device name.

<*user-data*> - Some opaque user data that is sent by the PIN Pad during the *PUT /rps/user[/<mpin-id>]* request to the RPS. The RPS just passes that data to the RPA "as is". This data might be used by the RPA as additional assistance the end-user verification process. The data might be set via the PIN Pad *registerRequestFormatter()* callback

Response:

200 OK - user identity is verified or a verification process has been started

4xx - user verification has failed.

Response Data:

```
{
        "forceActivate": <true|false>
}
```

forceActive should be set to `true` if the end-user has been verified in-place and the RPS should not expect further POST /user/<mpin-id> request to activate the user.

End-user Permission Callback Endpoint (GET /permitUser?mpin_id=<mpin-id>)

This request is optionally made as part of the end-user authentication flow. It is not mandatory to implement this endpoint. If implemented, the RPS will make this request to the RPA in order to assert that the end-user authentication might proceed on, and it can request the time permit shares for that user.

If this endpoint is not set in the RPS configuration, the RPS won't make this request and will assume that the RPA is not interested to revoke any end-users.

Parameters: mpin_id=<mpin-id>

<*mpin-id*> is the hex-encoded M-Pin ID.

Data: <none>

Response:

200 OK - end-user is permitted to proceed with authentication

4xx - end-user is not permitted not proceed with authentication

Response Data: <none>

End-user Authentication Endpoint (POST /authenticate)

This request is part of the end-user authentication flow. It is made by the PIN Pad to the RPA in order to verify the end-user authentication against the M-Pin Server. The PIN Pad first authenticates the end-user against the M-Pin Server and afterwards send the result of that authentication to the RPA for verification. The RPA should then make POST /authenticate request to the RPS, sending only the *authOTT* in the request data. The RPS verifies that this *authOTT* corresponds to a valid Authentication Token and returns status. The RPA should return the same status in the response to to the PIN Pad, but it can return also some custom response data that might be used on the client side.

Implementing this endpoint is mandatory, and its URL should be set in the RPS configuration. The RPS will propagate it to the PIN Pad within the Client Settings.

Parameters: <none>

Data:

```
{
```

```
        "mpinResponse": {
                "version": "0.3",
                "authOTT": <authentication-ott>,
                "pass": 2
        }
}
```

<*authentication-ott*> - Authentication reference number (OTT)

Response: The response should be basically the same as the one returned by the RPS *POST /authenticate* request

200 Authentication successful - on success;

401 Wrong PIN - on unsuccessful authentication;

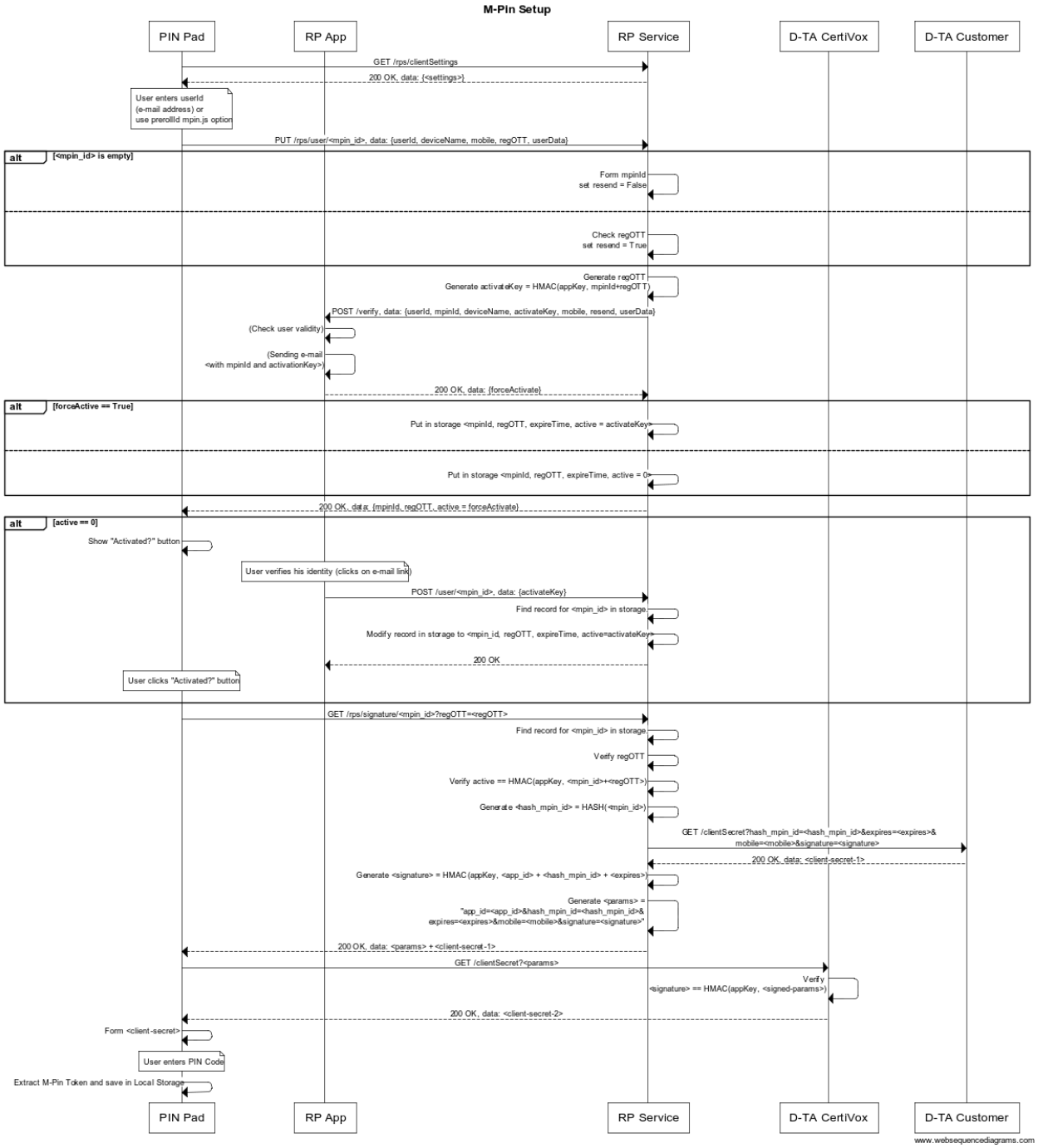
410 Wrong PIN - after N unsuccessful authentication attempts.

408 Expired authentication request - the authOTT is invalid or expired.

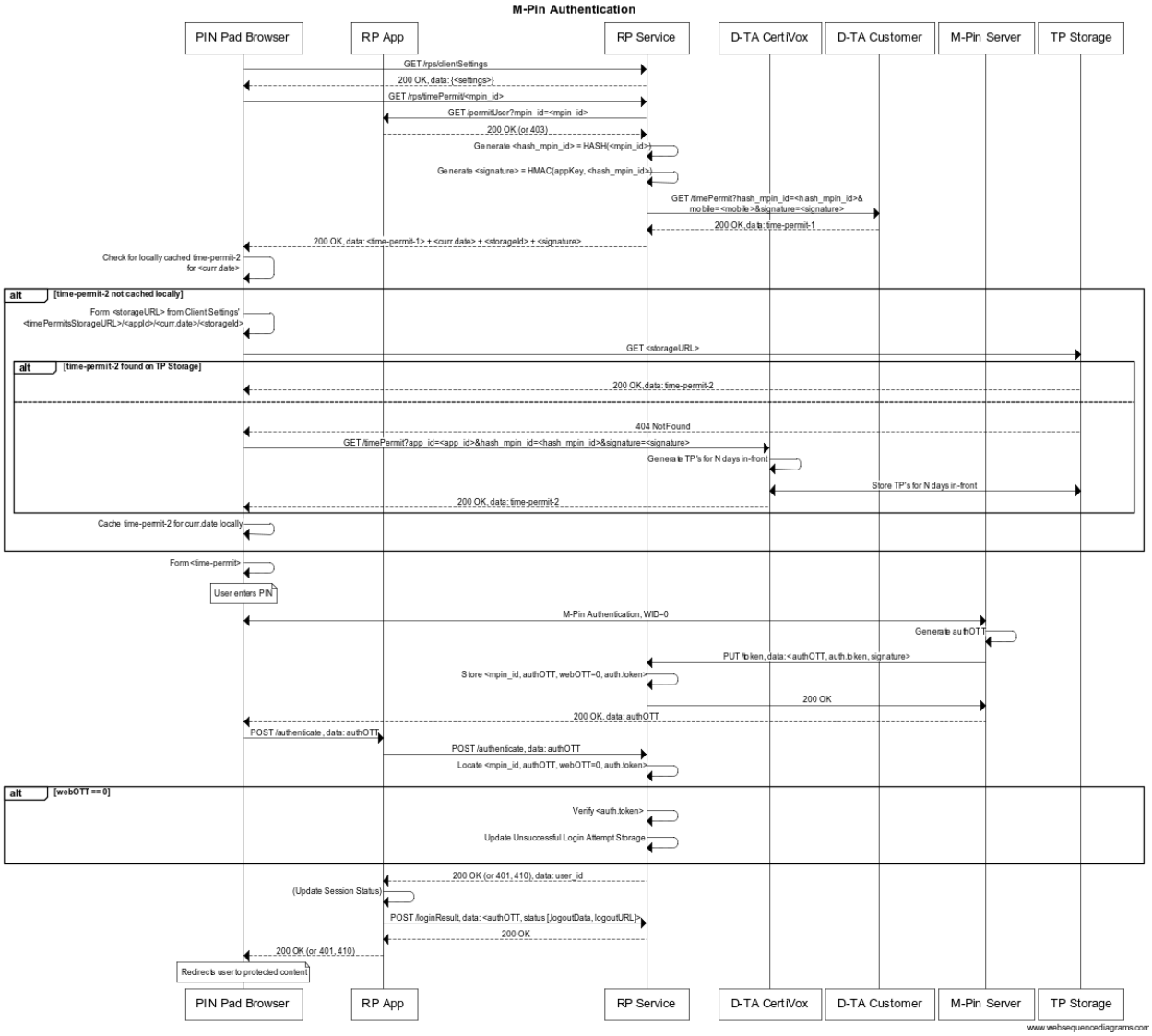
Response Data: <none> or any JSON-formatted application-specific data. This data might be used on the front-end side to implement any application-specific logic.

Workflows

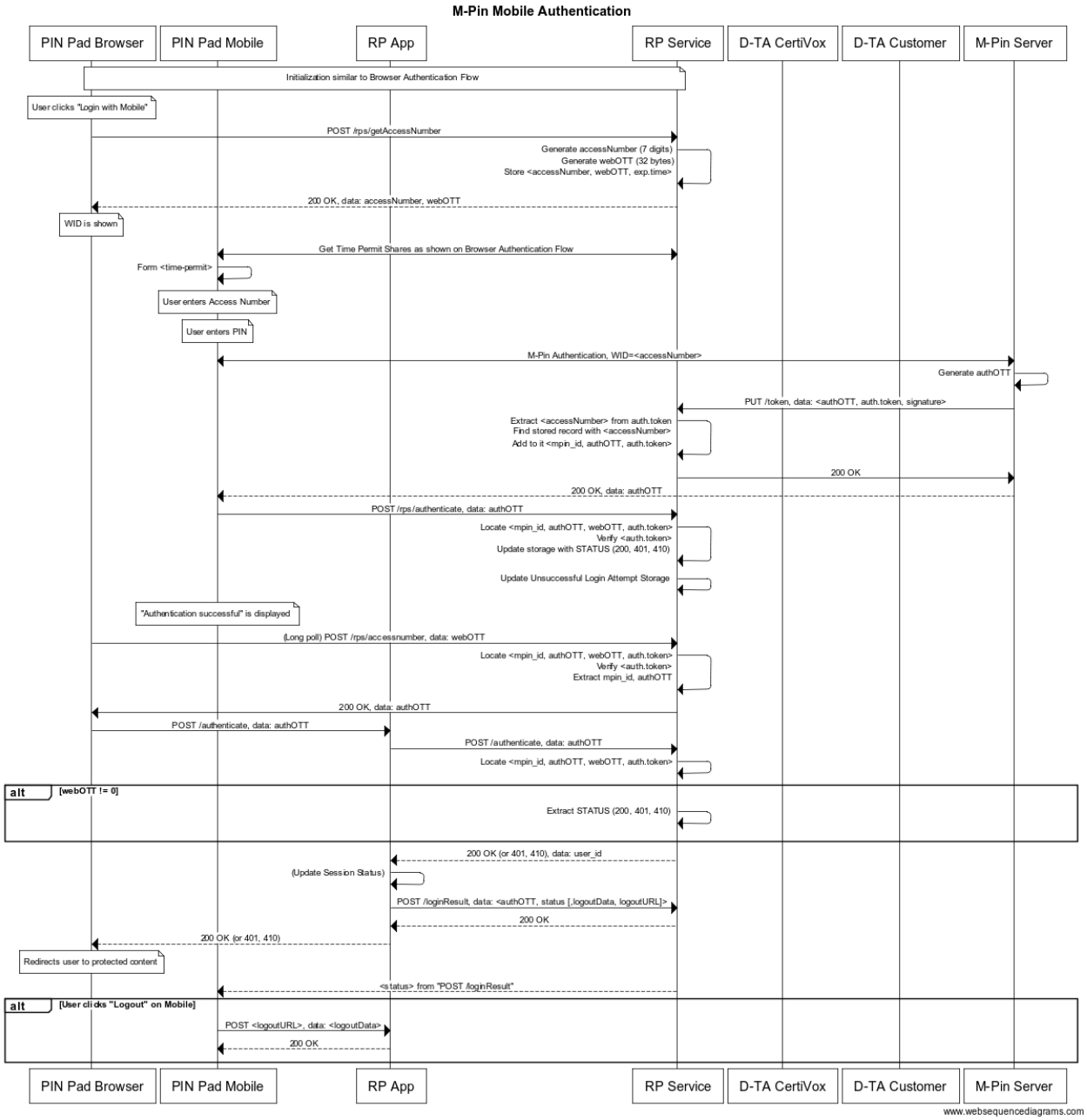
M-Pin Setup



M-Pin Authentication



M-Pin Mobile Authentication



Installation

The RPS is installed together with the rest of the Customer-hosted Services by the following command:

```
> python -c "$(curl -fsSL <installation-script>)"
```

Please see the M-Pin Core Quick Installation Guide for details on how to install and the M-Pin Core Configuration Guide for details on how to configure the M-Pin server.

Configuration

<installation-folder>/servers/rps/config.py

```
# RPS Configuration
port = <rps-port>
address = <rps-listen-address>
rpsBaseURL = <rps-public-requests-base-url>
rpsPrefix = <rps-public-requests-prefix>
keysFile = <keys-file>
MPinAuthenticationServer = <mpin-auth-endpoint>
# URLS
RPAVerifyUserURL = <rpa-verify-user-endpoint>
RPAAuthenticateUserURL = <rpa-auth-validation-endpoint>
RPAPermitUserURL = <rpa-permit-user-endpoint>
DTALocalURL = <local-dta-url>

waitForLoginResult = <wait-for-login-result>
logoutURL = <logout-endpoint>
setDeviceName = <set-device-name>
# Client settings
requestOTP = <request-otp>
accessNumberExpireSeconds = <access-number-expiration-in-seconds>
maxInvalidLoginAttempts = <max-invalid-login-attempts>
# User registration options
#VerifyUserExpireSeconds = 3600 # 3600 = 1 hour
VerifyUserExpireSeconds = <user-verification-expiration-in-seconds>
# Comma separated list of headers; Empty string - do not forward headers; * - Forward
all headers.
RegisterForwardUserHeaders = <forward-headers>

logLevel = <log-level>

storage = <storage>

# Redis options are for redis storage backend
redisHost = <redis-host>
redisPort = <redis-port>
redisDB = <redis-db>
redisPassword = <redis-password>
redisPrefix = <redis-prefix>
```

<rps-port> - The port on which the RPS listens. Example: 8011

<rps-listen-address> - The Network Interface IP Address on which the RPS listens. "0.0.0.0" means listen to all interfaces. Example: "127.0.0.1" - listen only to loop-back interface

<rps-public-requests-base-url> - Base URL for the Public RPS API. This is address of the proxy, through which RPS requests might be done. Example: "http://192.168.10.138:8005"

<rps-public-requests-prefix> - The prefix for Public RPS API. Example (Default): "rps"

<keys-file> - The file that includes the Customer details as App ID and App Key.
Example: "/opt/mpin/keys.json"

<mpin-auth-endpoint> - M-Pin Server WebSocket or RESTful endpoint for end-user authentication.
Example: "ws://192.168.10.138:8002"

<rpa-verify-user-endpoint> - The RPA endpoint for end-user identity verification.
Example: "http://192.168.10.138:8005/mpinVerify"

<rpa-auth-validation-endpoint> - Endpoint implemented by the RPA
for authentication validation. Example: "/mpinAuthenticate"

<rpa-permit-user-endpoint> - Endpoint implemented by the RPA for end-user revocation. Example: "http://192.168.10.138:8005/mpinPermitUser"

<local-dta-url> - The URL for the Customer-hosted D-TA Service. Example: "http://127.0.0.1:8001"

<wait-for-login-result> - Indicates (True or False) whether the RPS should wait for the POST /loginResult request before returning response to POST /rps/authenticate.

<logout-endpoint> - Default endpoint, typically implemented by the RPA, to which the Mobile App should make the logout request. This endpoint might be overwritten with the POST /loginResult, or alternatively might not be set as configuration option, but provided only during the request. Example: "http://192.168.10.138:8005/logout"

<set-device-name> - Indication (True or False) that is send from the RPS to the Client within the Client Settings. If set to *True*, the Client should obtain a friendly device name and send it to the RPS within the PUT /rps/user request. The RPS will then forward it to the RPA within the POST /verify request.

<request-otp> - Indicates (True or False) whether One-Time Password should be generated on successful authentication. Used for the Arion Server.

<access-number-expiration-in-seconds> - Access Number expiration period in seconds.

<user-verification-expiration-in-seconds> - Expiration time in seconds for the end-user identity verification completion. Example: 3600

<forward-headers> - Comma separated list of headers that should be forwarded to the RPA's POST /verify endpoint upon end-user registration. Empty string - do not forward headers, * - Forward all headers.

<log-level> - Level/Severity of the messages being logged. Valid values are:
"ERROR", "WARN", "INFO", "DEBUG".

<storage> - The storage mechanism that the RPS should use. Currently available storage's are "memory" (default), "redis"

<*redis-host*> - The address of the Redis storage to be used in case that storage option is set to "redis". Example: "127.0.0.1"

<*redis-port*> - The communication port of the Redis storage to be used in case that storage option is set to "redis". Example: 6379

<*redis-db*> - An integer indicating which Redis database to be selected. Example: 0

<*redis-password*> - Password for the Redis connection. "None" means no password.

<*redis-prefix*> - Prefix to be used when storing elements on Redis. Example: "mpin"