ONE Record Security

Developer Guide

This document describes user stories to fulfil requirements of IATA about the application of the x509 certificates in a complex environment, in which there are several OAUTH IAPs and multiple of Multi-User/Multi-Apps.

# Definitions

* **ONE Record ID**: URI that refers to a ONE Record identifier from the Internet of Logistics, with the form **https://<FQDN>/<COMPANYID>.**
* **ONE Record Server**: Application that accepts ONE Record API requests from a client.
* **ONE Record Client**: Application that sends ONE Record API requests to a server.
* **IAP**: Identity and Authentication Provider.

# Purpose

This document describes the concepts to incorporate OAUTH and SSL Mutual Authentication to bring strong and flexibility to protect 1R APIs.

# Scenario

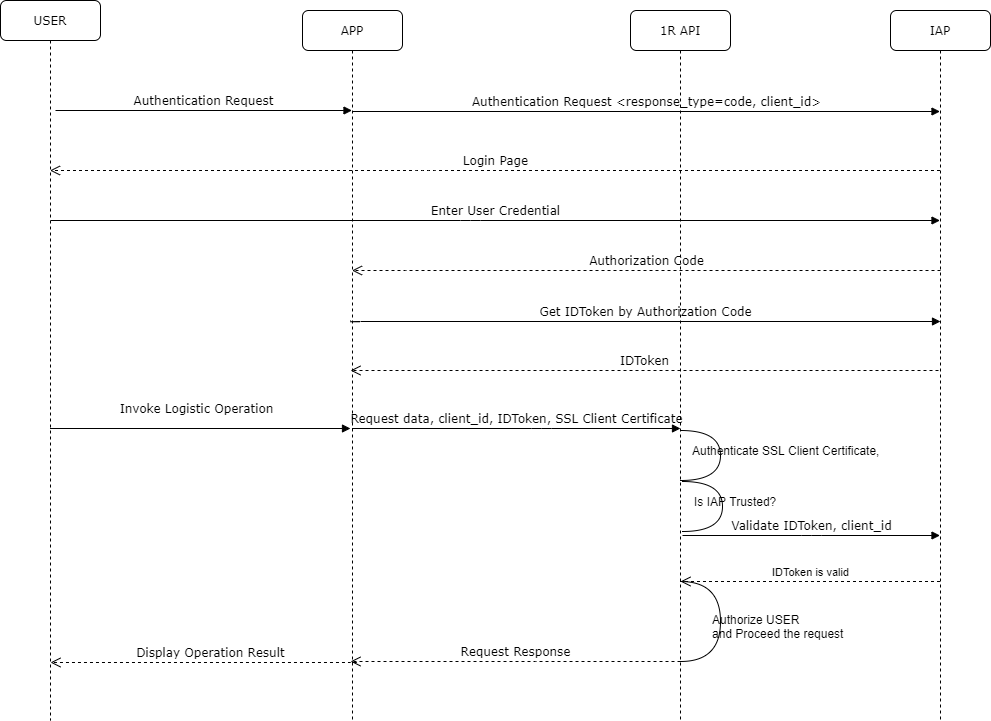
## Actors

For the purpose of this document, there will be following actors:

1. End-users: They are registered or created by some IAP. The end-users mainly interact with Logistics Applications which consumes 1R APIs. John Doe is an end-user. We will simply call end-users USER.
2. Logistics Applications: Applications interface with end-users to deal with Logistics Business Operations. These applications can call 1R APIs. These application can authenticate end-users through an IAP. We will simply call Logistics Applications APP.
3. 1R APIs: RESTful APIs implement IATA Protocols and Specification designed for Logistics. The APIs can authenticate API calls through IAP and SSL Mutual Authentication.
4. IAP: Identity and Authentication Providers, they MUST support OpenID Connect (OIDC).

## Sequence diagram

The sequence diagram below depicts the incorporation between OIDC Code Flow and SSL Mutual Authentication.



**OAUTH Authentication**

* In order to balance between security and developer-friendly, the OpendID Connect Code flow will be used in this document. You can find more information about OpenID Connect Code flow here:

[https://openid.net/specs/openid-connect-core-1\_0.html#CodeFlowSteps](https://openid.net/specs/openid-connect-core-1_0.html)

* It is very important that APP must have the client\_id (issued by IAP) value to create authentication requests.

**Below are steps in details**:

* USER launches the APP
* The APP prepares the Authentication Request containing the desired request parameters to send the IAP to authenticate USER.

*Below is a sample request*:

<https://IAP/authorize?audience=YOUR_API_AUDIENCE&scope=YOUR_SCOPE&response_type=code&client_id=YOUR_CLIENT_ID&redirect_uri=https://YOUR_APP/callback&state=YOUR_OPAQUE_VALUE>

*Where*:

* + response\_type: As the Code flow will be used, thus the value MUST be “**code**”
  + scope: The value MUST include “**openid**”

More information about parameters can be found here and many other resources:

<https://auth0.com/docs/api-auth/tutorials/authorization-code-grant>

* IAP opens the Login Page to authenticate the USER.
* The USER provide the credential.
* IAP validates the credential and issue an AUTHORIZATION CODE (aka CODE) and returns the CODE to the APP through the call back URL passed by **redirect\_uri** param.
* The APP requests the IDToken by the CODE.
* The IAP issued the IDTOKEN. It is recommended to use RS256 signature algorithm.
* The APP validates IDTOKEN.
* The USER is showed to be AUTHENTICATED.
* The USER execute a Logistic Operation in the APP.
* The APP calls 1R API using:
  + Client ID
  + IDTOKEN
  + SSL Client Authentication Certificate.
  + Request Data
* The 1R API does:
  + SSL Mutual Authentication including manually validate SSL Client Authenticate
  + Parse IDTOKEN
  + Validate the IDTOKEN
  + (Optional) Validate IAP’s RSA Signing Certificate
  + Proceed the request.

## Implementation

The key implementation in the approach described above is how a 1R API authenticate an API call. It is described again below:

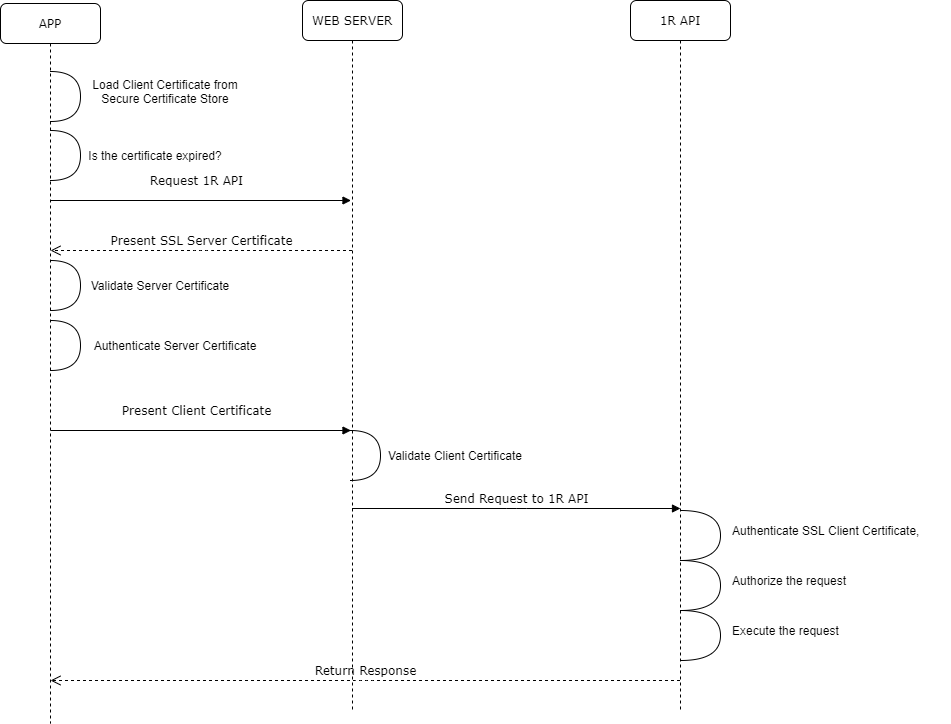
* SSL Mutual Authentication including manually validate SSL Client Authenticate.
* Parse IDTOKEN
* Validate IAP’s RSA Signing Certificate
* Validate the IDTOKEN
* Proceed the request.

**SSL Mutual Authentication**

The SSL Mutual Authentication can be used mainly in two cases:

* The 1R APIs implemented as a SINGLE-NODE and running as a typical Server Application.
* The 1R APIs implemented as Resource Servers under OAUTH Authentication.

Below is the sequence diagram about the SSL Mutual Authentication.



The SSL Mutual Authentication can be divided in steps below:

* The web server of the 1R API must be configured to support SSL Mutual Authentication.
* The APP loads Client Certificate.
* The APP checks if Client Certificate is expired.
* The APP request 1R API.
* The web server presents Server Certificate.
* The APP validate Server Certificate by common practices like validate certificate signature, building certificate trusted chain, validation certificate status through OCSP or CRL. This chain of operations for validation often done by CLIENT SDK.
* The APP Authenticate Server Certificate: This is to add an extra security layer to increase security. This authentication checks:
  + The SSL Server Certificate is issued by IATA CA
  + The 1R API endpoint is correct – checking through SAN DNS Names of the certificate.
* The APP present Client Certificate
* The web server validates Client Certificate by common practices like validate certificate signature, building certificate trusted chain, validation certificate status through OCSP or CRL.
* The web server send request to 1R API.
* The 1R API Authenticates Client Certificate. This is to add an extra security layer to increase security. This authentication checks:
  + The SSL Client Certificate is issued by IATA CA
  + The request come from right subscriber and corresponding 1R ID – checking through Common Name and SAN URIs

Important:

* The sample code provided along with the document is able to extract SUBJECT and SAN of certificates. 3rd parties building the APP or 1R API can use extracted SUBJECT and SAN to authenticate certificates.

**Parse IDTOKEN**

Specification for IDTOKEN can be found here:

<https://jwt.io/>

*Important*:

It is possible to find basic information about USER through CLAIMs in the IDTOKEN. These CLAIMS should be used for AUTHROIZATION.

The information the IAP which issues the IDTOKEN can be found in the IDTOKEN. The information should include:

* IAP’s APIs endpoints
* Public Key of the IAP’s RSA Signing Certificate.

**Validate IAPI’s RSA Signing Certificate**

If the IAP uses RS256, and signing key is associated with a X509Certificate, it is strongly that the certificate should be issued by IATA CMS. It is because:

* Simple in signature verification: Many mature libraries support development.
* Simple in manage signing keys: Management through CMS, CA

**Validate IDTOKEN**

The validation may include steps below:

* Validate IDTOKEN signature
* Validate IDTOKEN through APIs of the IAP’s API. This will require client\_id value to make the API call.

# Best Practices for IAP

* In order to strengthen security, IAP Signing Key should be rotated. It is suggested that the rotation interval is 1 day.
* The RS 256 Algorithm is recommended because:
  + The signing key can be associated with a X509 Certificate issued by IATA CMS.
  + The signature verification using RS 256 is mature and available in most programming languages.
  + By associating a signing key with a X509 Certificate, it helps:
  + Better in key management: Key Lifecycle, Revocation, etc
  + IATA CMS is a ready-made tool to manage certificates, thus signing keys.
  + Trusted IAP
    - Web-applications which implement IATA One-Record APIs, and have to deal with IDTokens issued by External/Public IAPs. Therefore, it is required for the web-applications to validate the IDTokens against the issuer (IAP).
    - By association a signing key with a X509 Certificate, it is easy for the web applications to recognize if the web-application should TRUST the IAP.
* Associate an IAP signing key with a X509 Certificate: This can be done as follows:
  + Generate a RS 256 signing key.
  + Generate a CSR – certificate signing request signed by the signing key.
  + Get a X509 Certificate by the CSR from the IATA CMS.

# 

# Appendix

## CA Certificates

### Root CA Certificate

* The root CA for development environment can be found here:

<http://public.wisekeydemo.com/crt/demorootca1.crt>

* The guides to put Root CA Certificate into appropriate TRUST Certificate store should be available in the sample source code packages.

### Issuing CA Certificates

* The Issuing CA for development environment to issue SSL Server Certificates can be found here:

<http://public.wisekeydemo.com/crt/demopersca1.crt>

* The Issuing CA for development environment to issue SSL Client Certificates can be found here:

<http://public.wisekeydemo.com/crt/demosslca1.crt>

* The guides to put Issuing CA Certificates into appropriate TRUST Certificate store should be available in the sample source code packages.

## Request Certificates

* Refer the IATA CMS User Manual to request certificates.
* Below is a reference about how to generate certificate signing requests of some popular web servers

<https://support.wisekey.com/portal/kb/wisekey/ssl-install-documentation>

## Certificate Stores

Client and Server Certificates can be securely stored in following certificate stores:

* HSMs or USB Tokens: Managed by pkcs11 libraries provided by hardware providers.
* Pkcs12 (pfx) Files: Managed by native windows tool or various open source tool like openssl.
* Windows Certificate Stores in Microsoft Windows Operating Systems: Managed by Windows MMC interfaces.
* Java Key Store Files: Managed by Java keytool command
* Pem Files: Managed by UNIX-based text editor like vi or nano, etc.

## Web Server Configuration for SSL Mutual Authentication

### Microsoft IIS

Refer official document of the IIS and README.doc file in .NET source code sample.

<https://docs.microsoft.com/en-us/iis/manage/configuring-security/how-to-set-up-ssl-on-iis#SSL>

### Apache

Refer official document of the Apache web server

<https://httpd.apache.org/docs/2.4/ssl/ssl_howto.html#allclients>

### Nginx

Refer official document of the Nginx web server

<http://nginx.org/en/docs/http/ngx_http_ssl_module.html>

### JBoss

Refer official document of the JBoss web server

<[TBD](https://developer.jboss.org/thread/279429)>

# References

* All standard OIDC IAP should provide following endpoints:

/[.well-known/openid-configuration](https://iata.certifyiddemo.com:11443/iap-a/.well-known/openid-configuration)

To discover OIDC APIs implemented by the IAP

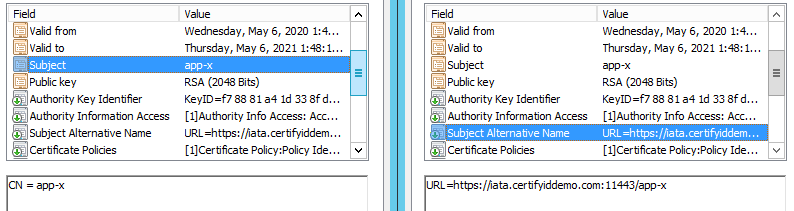
In which, this endpoint is to download the public key of the IAP’s signing certificate

/.well-known/openid-configuration/jwks

Below is a sample

<https://iata.certifyiddemo.com:11443/iap-a/.well-known/openid-configuration>

* Below is a sample of SUBJECT and SAN of a SSL Client Certificate issued by the Preview IATA CA. Where ***app-x*** is right a **ONE Record ID**

****

* Below is a sample of SUBJECT (empty) and SAN of a SSL Server Certificate issued by the Preview IATA CA:

