Configuration for Travel Validation Services

# Overview

Deploy and configure the services in the following order:

Wallet

Validation Service

Airline service

The 2 services are Spring Boot applications and use the standard configuration file mechanism (application.properties) and profiles for configuring different environments

This Wallet is a .NET/Blazor application and uses the standard configuration file mechanism (appSettings.json) and its environments mechanism configuring different environments.

Configuration for the services is present in both the configuration files and their identity documents which themselves are JSON files.

The two services require at least one suitable key pair (RSA4096 or ECxxxx) each for signing and verifying tokens. The public verification keys must be added to the service’s identity document.

The intended endpoints of each service must also be added to the corresponding entry in the services identity document or configuration file. Care should be taken when configuring the endpoint values – the URL must be reachable by the ultimate user of the value (which is often NOT the location of the service which has the value in its configuration). Point of View (POV) is given for each configured URL.

Logging is log4j.

In general, start with the basic profile or nearest file to your profile/environment and change the values as per the following sections.

Note the encoding (the one that starts with ASN1…) specified for public and private keys are the most commonly used formats and are usually obtained simply by using the default encoding of the key from Bouncy Castle, .NET or Java.

Several working examples of configuration files and identity files are present in the repositories.

## Notes for Wallet

This application requires a suitable .NET linux, windows or Docker host.

The only configuration option is whether to run it in Development or Production environment (standard .NET web environment configuration applies) which affects the mechanism for displaying debug information on the client.

Make a note of the URL it is hosted on for the Airline service (POV – from Airline Service)

## Notes for Validation Service

### Authorised list of service providers

Config file entry: validation-service.airlineIdentityUris

Comma separated list of service provider identity URIs. If the list is empty or at least one public key is not found, the service will not start. (TODO recent change request – service will start if at least one url is in the accept list, no attempt to obtain a public key.) All URIs are from the POV of the Validation Service

## Key Pair for Result Token Signature

Generate a RSA4096 key pair.

For the private key, get the base 64 string of the byte array of the key in Asn1/Der/Pkcs8 format.

For the public key, which is configured in JWK format as:

x5c: base 64 string of byte array of the private key in Asn1/Der/Pkcs1/X509 format

kid: base64 string of first 8 bytes of the SHA256 of private key in Asn1/Der/Pkcs1/X509

Configuration file entry:

validation-service.validationResultJwsSigningKey: private key

Identity file elements:

verificationMethod with id ending ‘#ValidationServiceSignKey-1’ or other digit: public key

## Key Pair for DCC Encryption

Generate a RSA4096 Key Pair.

For the private key, get the base 64 string of the byte array of the key in Asn1/Der/Pkcs8 format.

For the public key, which is configured in JWK format as:

x5c: base 64 string of byte array of the private key in Asn1/Der/Pkcs1/X509 format

kid: base64 string of first 8 bytes of the SHA256 of private key in Asn1/Der/Pkcs1/X509

Configuration file entry:

validation-service.dccEncryptionRsaPrivateKey: private key

Identity file elements:

verificationMethod with id ending ‘#ValidationServiceEncKey-1’ or other digit: public key

## Infrastructure and other values

## Session Lifetime

The maximum time, in seconds, a session should last, where a session is the time between the /initialize and /validate call. A reasonable value e.g. 15mins= 900, should take into account the length of time an unprepared user may take to obtain the DCC for upload (e.g. not necessarily the lifetime of the validation access token set by the airline, although it would be preferable for all parties to use the same lifetime value)

validation-service.sessionMaxDurationSecondsString = 900

## Result Token Expiry

The maximum time, in seconds, between issuing a result token when it expires. TODO does this just have to last until after the confirmation token is posted to the airline? Or would they be re-evaluated at a later date? Are there not other mechanisms for this e.g. valid at? Currently set to an hour = 3600.

validation-service.resultTokenLifetimeSeconds = 3600

Folder and name of the the identity.json file  
Configuration file entry:

validation-service.configFileFolderPath = build\\resources\\main\\dev

Do not change this entry:

validation-service.publicKeysFileName = public\_keys.json

Host name of the redis service  
Note. If hosted in docker, this will be the name of the redis service in the compose file.

Configuration file entry:

validation-service.redisHost = localhost

URI of the DCC verification service

The URI must be in the format <http://HOSTNAME:PORT/verify_signature> where HOSTNAME and PORT are the configurable values.

Note. If hosted in docker, the hostname will be the name of the verification service in the compose file.

validation-service.dccVerificationServiceUri = <http://verifier:4003/verify_signature>

## Business Rules and DCC Verification Keys

The following URI must be present and correspond to the OTAP/DTAP set required:

In orderf:

Configuration for color codes  
 NL custom business\_rules  
 EU distributed business\_rules  
Values Sets referred to in the business rules  
Trustlist (public keys) issuers

Configuration file entries (ACC shown):

validation-service.configUri = https://verifier-api.acc.coronacheck.nl/v4/dcbs/config  
validation-service.customBusinessRulesUri = https://verifier-api.acc.coronacheck.nl/v4/dcbs/custom\_business\_rules  
validation-service.businessRulesUri = https://verifier-api.acc.coronacheck.nl/v4/dcbs/business\_rules  
validation-service.valueSetsUri = https://verifier-api.acc.coronacheck.nl/v4/dcbs/value\_sets  
validation-service.publicKeysUri = https://verifier-api.acc.coronacheck.nl/v4/dcbs/public\_keys

### Config items for other services

Make a note of the identity URL for the Airline service (POV – from Wallet) which will be used when generating Initiating QR Code Tokens.

## Notes for Airline Stub

Apologies in advance for the inconsistent approaches that follow. This is not ‘production’ quality code.

## Key Pair for Validation Access Token Signature

NB. Please make yourself familiar with the recent (at time of writing) ‘psychic paper signature’ vulnerability before using ES256 in production. See https://cve.mitre.org/cgi-bin/cvename.cgi?name=2022-21449.

For ES256 signatures, generate an ECDSA key pair. For RS256 and PS256, generate an RSA4096 key pair. Generate the associated configuration values. See

For the private key, same this as a file called ‘accesstokensign-privatekey-1.pem’ in Asn1/Der/Pkcs8/X509/PEM format.

For the public key, which is configured in JWK format as:

x5c: base 64 string of byte array of the private key in Asn1/Der/Pkcs1/X509 format

kid: base64 string of first 8 bytes of the SHA256 of private key in Asn1/Der/Pkcs1/X509

Configuration file entry:

airline-stub.configFileFolderPath : the folder containing the private key pem file – see below.

Identity file elements:

verificationMethod with id ending ‘#AccessTokenSign-1’ or other digit: public key

## Infrastructure and other values

Folder of the the identity.json and accesstokensign-privatekey-1.pem files  
Configuration file entry:

validation-service.configFileFolderPath = build\\resources\\main\\dev

## Wallet Process URI

The Initiating QR Token will be processed by a wallet recommended by the Airline. To derive the full URI (= this prefix + base64 of the Initiating QR Token), use the URI where the wallet is hosted as noted earlier.

Configuration file entry:

airline-stub.walletProcessUrl = <your wallet URI>/process

## Validation Service URI

The airline also recommends which validation service to use.

The Validation Access Token will specify which validation service is used by including the identity URI of the Validation Service as noted earlier.

Configuration file entry:

airline-stub.validationServiceIdentityUri = http://192.168.178.12:8080/identity

Host name of the redis service  
Note. If hosted in docker, this will be the name of the Redis service in the compose file.

This service uses Redis cache to store validation attempts and whether they have been notified of a result token.

Configuration file entry:

airline-stub.redisHost = redis

# Scenario – Development

All on same box, running 3 IDE instances, Docker for DCC Verifier and Redis.

# Scenario - Local Demo on Docker

All in the same docker service, 1 image for the wallet, 1 compose for each service.

# Examples

In each repo, there are examples of the scenarios below.

The wallet image can be configured to change to production or development using environment variables. See <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/environments?view=aspnetcore-6.0>

TODO The Airline Stub and Validation Service do not make use of environment variables or a separate layer for configuration. The required configuration must be present before an image are built.

WORKAROUND

This requires compiling the source code using IntelliJ. Once this is done, the configuration can be changed and a new image built with that new configuration:

1. Add a new configuration file with a new SpringBoot profile name and add the identity document and pem file if required to a new folder.
2. Set that folder in the configuration file.
3. Build the image and docker-compose up to deploy.

# Local Development

Deploy a single Redis docker instance using the image redis:6.2.6-alpine without a password. The validation service and airline stub can share this instance.

For the dcc verifier, use the docker image http://xxx.xxx.

Alternatively, build the image from from <http://sadsdasd> which will require Gnu C Compiler and GoLang on your dev machine.

Run/debug each service in its own IDE instance using Visual Studio and IntelliJ.

As can be seen in the repos, most configured URI are of the form http://localhost:8080, 1, or 2.

# Scenario – Host each service using docker-compose

[The wallet does not require a docker-compose]

Change the configuration files to account

As per Local Demo on Docker, with the configured endpoints changed to account for the changes in network POV. Examples of the configured URIs will be of the form http://<dev machine ip address>:9000, 1, or 2. The machine ip address is used because of the POV of one service to another – if the validation service URI is set to localhost in the airline stub, it will fail as the service is simply not hosted in the same docker image.

# Scenario - Local Demo on Docker exposed to public endpoints

Similar to the above scenario – Host each service using docker-compose with the relevant URIs set to the endpoints visible to an user on the internet.

# DTAP/OTAP Environment Scenarios

Best practice to use different key pairs for each environment.

Endpoints are the most likely items likely to differ. The respective POVs will remain the same and the values will likely on differ by containing the text ‘acc’, ‘tst’, etc.