Validation Protocol and Wallet Developers Guide - Spec 1.2 (NL flavour)

Contents:

[Open issues](#_7oglhbbo2bk4)

[Backlog](#_30j0zll)

[Introduction](#_nngo5ocl1ccn)

[Signing and Verification](#_m5ymei4w1dvs)

[What is signed and what verification is performed?](#_u2ouldtny4wg)

[Algorithm](#_k6ly55tkrf8y)

[Verification Keys](#_ke9pw2li0y4d)

[DCC Encryption](#_dirumioxjnua)

[Wallet Application –Sequence Overview](#_fz6f50btw6vf)

[Wallet Application – Detailed Sequence](#_7buwt3y4xryh)

[1 - 3 Start](#_kz7xzkdjg09s)

[Wallet gets Airline Identity Document](#_7pn4n8i53fkc)

[Wallet gets Trip Information](#_75rrpf6h7ttu)

[Obtain user consent](#_c6r8b3np8adq)

[User provides wallet with a DCC](#_139mrjm7cjmc)

[Validate DCC](#_9io6li4fwioq)

[Wallet displays the results to the user.](#_mv5hz9io4qxs)

[Send the successful result to the airline](#_ov0s9g88oxg4)

[interaction between airline and validation service](#_tyqg2opdhoq9)

# Open issues

|  |  |  |
| --- | --- | --- |
| **What** | **where** | **comment** |
| Callback to airline | chapter: Send successful result to airline | proposal:  Endpoint: extend list of ID values for service in chapter 3.8.2.3 in EU-DCC-v1.2 with “ConfirmationService”  Payload: “confirmation”: confirmation-token-string |
| exchange encrypting key from validation service, to airline, to wallet | chapter: interaction between airline and validation service | carry as response header in /initialize call  X-Crypto-Enc base64 of UTF-8 of JSON of JWK for encrypting key  Optional, carry X-Crypto-Sign base64 as above for signing key |
|  |  |  |

# Backlog

|  |  |  |
| --- | --- | --- |
| **Item** | **discussion** | **priority** (1, 2, 3|must, may, nice) |
| Publish at github  In docker repo | fixed | 1 |
| Build and containerize instructions starting from github | fixed | 1 |
| OAS-specification aligned with deployment | fixed, review requested | 1 |
| validation service | clarify what is actually delivered, which version of protocol, specific deviations, what data feeds are expected (mandatory, optional)  action: KLM/Airlines? | 1 |
| html wallet | Fixed, review requested   * clarify what wallet functionality is actually delivered (activation, ingesting invitation, ingesting DCC, etc) * move image parsing function (PDF or other images) to validation service, keeping open the option of processing already parsed DCC contents. | 2 |
|  |  |  |

# Introduction

This document describes the NL-specific protocol based on 1.2 of the EU spec document.

The NL protocol differs in the following ways:

1. The Validation Access Token contains information about the trip, e.g. country, region and port of arrival and departure so the NL business rules can be processed.
2. A limited amount of personal information is extracted from the DCC – the transliterated given and family names and date of birth – so that the identity of the traveller can be reconciled with the identity of the owner of the DCCs.
3. Wallet signature checking – see below.
4. The Confirmation Token is sent directly from the wallet to the airline (instead of using a polling mechanism)

# Signing and Verification

In the short term, to simplify the protocol, the wallet will not exchange a signing key pair with the Validation service. Consequently, the wallet will not sign requests to the Validation Service, nor will it check the signatures of responses from the Validation Service.

The URL of the validation service is provided by a trusted source – the airline - so another layer of verification should not be necessary.

With that assumption in mind, this section is not relevant to Wallet developers.

## What is signed and what verification is performed?

The following checks must be made by the services:

1. Airline Services will sign the Invitation and Validation Access Tokens. The Validation Service must check the signature of the Airline’s Validation Access Tokens. The Airline must check the signature of Invitation Token.
2. Validation Service will sign the Result Token and the contained Confirmation Token. The Airline must check the signature of the Validation Service’s Confirmation Token.

## Algorithm

All signatures must use the RS256 algorithm. Use keys of either length 3072 or 4096 bytes.

## Verification Keys

As per the 1.2 spec, all JWKs will place public key certificates in the x5c field in PEM format (examples in Appendix X).

In the short term, only a single key – the first one in the array as per RFC7517 – will be used. Multiple keys – a certificate chain - can be sent but only the first key will be used for verification. Full support for certificate chains will come in a later version.

Kids are generated by taking the first 8 bytes of the SHA256 of the value of the x5c as bytes. This is to ensure a high probability the kid is unique across the set of public keys provided by the airlines using the protocol.

# DCC Encryption

Wallet implementations must use the encryption scheme specified by the value in the key DccEncryptionScheme2021Validation in the Validation Service’s Identity document. This will be RSAOAEPWithSHA256AESGCM in the first release.

# Wallet Application –Sequence Overview

1. In the Airline’s Booking Application, the traveller (aka the user) is presented with the Initialization QR Code containing a token that contains just enough information to initiate the validation sequence. (Note the size of the data in the Initiating QR code is small so that devices with limited functionality are supported.)
2. Using the URL in the QR code the user navigates to the wallet application.
3. The wallet parses the QR Code for the Airline s Identity Document URL.
4. The wallet gets the Airline s Identity Document and obtains the URL for the /token endpoint.
5. The wallet GETs the Validation Service’s Identity document and parses it for the DCC encryption public key.
6. The wallet POSTs the Initiating Token from the QR Code to the /token endpoint to obtain the Validation Access Token and a nonce. The wallet now has the full trip information.
7. The wallet displays the trip information from the Validation Access Token and the UX info in the QR code to the user and obtains explicit consent to proceed.
8. The wallet enables the user the upload the QR code of a DCC
9. Using the nonce and the encryption key of the Validation Service, encrypt the DCC and POST to the /validate URL along with the Validation Access Token.
10. The wallet displays the results to user. If not successful, the user may go back to step 8 and try again.
11. If successful, the user may send the result to the airline. The wallet POSTs the Confirmation Token to the Airline s /callback endpoint.

# Wallet Application – Detailed Sequence

## 1 - 3 Start

(TODO Glossing over steps 1 and 2 for now)

It should be noted that travellers that cannot validate need to apply for exception from country of arrival (else they will be refused boarding)

## Wallet gets Airline Identity Document

(No user interaction)

Using the URL in the QR Code from field serviceIdentity, GET the airline service’s identity document.

Parse the resulting JSON for 3 items:

* Services/AccessTokenService - The full URL for POST/token
* Services/ResultTokenService - The full URL for POST/callback

## Wallet gets Trip Information

(No user interaction)

POST the field token from the QR Code in the 'authorization' header with value 'bearer' followed by the token value to the URL from AccessTokenService. The response body is the Validation Access Token. The header X-Nonce contains the 16byte IV used when encrypting the DCCs in Step x

The headers X-Crypto-Enc and X-Crypto-Sign contain base64 of encryption resp signing public key of validation service, expressed in json web key format

## Obtain user consent

The wallet displays the trip information from the Validation Access Token and the UX info in the QR code to the user and obtains explicit consent to proceed. The user click on the Next/OK button.

## User provides wallet with a DCC

Method tba.

The next step requires the DCC in PREFIX format e.g. “HC1:sdfsfsf”.

## Validate DCC

The wallet encrypts the PREFIX-format DCC and requests validation as follows:

* 1. Generate a 256byte secret key for AES (use a method suitable for cryptographic operations)
  2. Decode the DCC from its UTF-8 string into a byte array.
  3. Encrypt the byte array with the generated secret key and use the Nonce from Step 5 as the Initialization Vector using AES256 in GCM mode.
  4. Encrypt the secret key using RSA and the public key ValidationServiceEncryptionKey-1 from Step 6/
  5. POST the encrypted DCC to the URL given in the ‘aud’ field of the Validation Access Token using a body in the form:

Headers:

content-type:application/json, accept:application/json, x-version:2.00. authorization:bearer <Validation Access Token from Step 5.>

Body:

{

“kid”: kid of the key ValidationServiceEncryptionKey-1,

“dcc”: the encrypted dcc as base64,

“encScheme”: “RSAOAEPWithSHA256AESGCMCBC”,

“encKey”: base64 of the RSA encrypted secret key

}

The 2 fields related to signing the encrypted DCC will be ignored.

Validation result is expressed as ResultToken

## Wallet displays the results to the user.

Display the fields Result and the items from Results from the response to Step 8 to the user.

If successful, field result = “OK”, obtain the consent of the user and go to Step 10.

If not successful, prompt the user to go back to step 8 and try with a different DCC.

## Send the successful result to the airline

Display the fields Result and the items from Results from the response to Step 9 to the user.

On obtaining consent, simply POST the field ‘confirmation’ from the result from step 9 as below:

x-version:2.00, content-type:application/json

The body is a JSON with a single field ‘confirmation’ with the value of field ‘confirmation’ from Step 9.

# interaction between airline and validation service

to prepare for assessment of documents from wallet. Documented here to complement the wallet perspective

Case (subject) is announced, validation service presents a key for encryption which airline forwards to wallet, and a key to validate signature on result/confirmation token

Airline is in need of a source to discover

* Validation /initialize service endpoint
* Validation /validate service endpoint
* Verification methods (keys) can be static and presented in an identity document, or more dynamic and shared in context of a case.   
  For signing key, static approach is recommended as verification of signature may be long time after actual signing.  
  For encryption key choice is left to validation service implementor