EHIC PDF QR Code PoC
Technical Specifications

Date:	6 October 2025
Version:	0.5

Table of Contents

1. CONTEXT

We refer to the *eEHIC – Overview.docx* for the general context of the project. This document is intended to provide the technical specifications of the project's different component. The specifications described in this document should enable the IT department of the different stakeholders (issuers and verifiers) involved in the project to implement and integrate the project into their own infrastructure.

The following specifications are explained in this document

Issuing:

1. Technical Specs on the signed JWT;
2. Technical Specs on creating the QR code from the signed JWT;
3. Format and specs for the PDF containing the QR code.

Verifying:

4. Functional Description of the verification flow;
5. Technical Specs of the Resolver API for fetching the public keys for signature verification;
6. Format and specs for the export of the verification result.

In annex we elaborate on the following topics:

1. EUDI Compliance of the signed ePRC Signature;
2. QR code security assessment.

Note on future evolutions of the specifications

This document is still a draft. Following the conclusions of the PoC some changes are still possible.

More specifically, certain data modelling choices can be opened for debate. If necessary to align with other approaches, such as the digitalisation of the European Disability Card (EDC), these choices can be reconsidered.

Note on eEHIC versus ePRC

For practical and timing reasons, the proof of concept focuses on the issuance and subsequent verification of the PRC (Provisional Replacement Certificate) document. Additionally, changing the way the PRC is issued and verified has a more limited legal impact compared to changing the EHIC document, which specifications are currently described in Decision S2 from the Administrative Commission. However, **all the findings and conclusions of this proof of concept** – from a procedural and technical point of view, including security features – **will also be applicable to the EHIC document** and other Portable Documents.

2. TECHNICAL SPECS ON THE SIGNED JWT

The key data of the electronic Provisional Replacement Certificate (ePRC) of the European Health Insurance Card will be included in the PDF document in the form of a signed JSON Web Token (JWT), graphically represented by a QR code.

The JWT consists of a header, payload and signature as described in the JSON Web Signature (JWS) and JSON Web Token specifications.

2.1. Requirements/Design choices

The design of the JSON Schema describing the data model of the header and payload of the JWT has been inspired by several axiomatic prerequisites. Each of them having certain design choices as consequence.

- Duplication of information is limited as much as possible to reduce as much as possible the size of the object.
    - The packaging of the PRC data as a credential inside a verifiable credential payload, as described in the verifiable credential data model (W3C VCDM 2.0), was not withheld for this reason.[1]
- All data visible on the PRC, as described in Decision S2, should be present in the token, to be able to secure the entire dataset.
    - The 'certificate validity period From/To' and the 'expiry date' are included as separate fields in the card-related information.
    - Dates can be represented in ISO 8601 format in the token itself, but must be visually represented on the PDF in the format YYYY/MM/DD as described in Decision S2.
- The main focus of the data model is to allow checking the validity of the information at the business level, which is equivalent to a human validation of the information visible on the PRC, as described in Decision S2. Any confusion between technical validity dates and business validity dates should be avoided.
    - Technical date-times including time zones are omitted from the data model in favour of business dates that are already present in the data visible on the PRC as described in Decision S2.
        - Only one instance of the 'issuance date' of the token, which is equal to the 'certificate delivery date' visible on the PRC, as described in Decision S2, was withheld. No other issuance date (such as 'iat') is present in the token.[2]
        - Only one instance of the 'start/end date of validity', which is equal to the 'certificate from/to' visible on the PRC, as described in Decision S2, was withheld. No other start/end dates of validity (such as 'nbf' or 'exp') are present in the token.

[1] Since the verification of the PRC data itself requires additional specific types of validation (see section 5.5) the advantage of using a standard model like the W3C VCDM 2.0 which offers a generic framework for verifying credentials could not hold up against the disadvantage of the extra information needed to be represented in the restricted area we have at our disposition in the QR code.

[2] Notice, as mentioned further in the document, that this has as consequence that the ePRC can only be signed on the same date as issuing the actual PRC it represents. Caching PRC for future distribution thus requires also caching the PRC as a signed JWT, e.g. an ePRC.

- Business dates from the PRC, as described in Decision S2, do not include a time zone and are modelled as a local date in the data model.
    - Verification of the start and end dates of the PRC, with respect to the treatment date, will be carried out regardless of the time zone. This mean that even if the time zone of the country in which treatment is received has a different time zone from the country of origin, verification will ignore any time zone differences and only consider the 'local' treatment date against the 'local' start and end date. This is specifically done to avoid confusion for citizens.
- The restrictions imposed by the JSON Schema should uphold a balance between validating the data allowing differences between member states and changes to business rules without impacting the technical schema.
    - Some business validations are therefore excluded from the JSON schema, but should be ensured by both the issuer and verifier. Others should only be verified by the issuer, to allow flexibility between the member states; the verifier is not obliged to check these. See 2.4.1

2.2. Other alternatives considered

- Selective disclosure
    - This is not useful in the context of a printed PDF.
        - All the information comprised in the JWT is also (already) visible on the PDF.
        - It is not possible to interact with the PDF to request selections of the dataset.
    - Consequence
        - Use of a SD-JWT (selective disclosure) is not withheld
        - mDoc: mDoc, as per ISO 18013-5, is not withheld
- CBOR/CWT (EU Digital COVID certificate): CBOR encoded PRC has been considered. The size reduction in the case of PRC didn't significantly impact the QR code size, hence the simpler to implement, JWT format has been selected.
- Use of W3C VCDM is not appropriate for compact credentials due to extensive metadata.

2.3. Header and Payload

We refer to the Annex I in section 8 for the EUDI Compliance of this Compact Credential Signature Profile.

2.3.1. Header

Name	M/O[3]	Format	Description
alg	M	*"type": "string",* *"enum": ["ES256", "RS256"]*	The signing algorithm of the certificate used to sign the JWT. For the moment only ES256 and RS256 are allowed.

[3] Mandatory (M) or Optional (O)

typ	O	*"type": "string"*	Type of token. For the moment only 'JWT' is allowed, if specified.
kid	M	*"type": "string",* *"pattern": "^EESSI:x5t#S256:[A-Za-z0-9_-]+$"*	Signing key or certificate identifier. Consist of three parts: • **EESSI**: name of the repository of the trust framework where the public key can be found. At the moment this is EESSI only. • **x5t#S256**: Type of thumbprint taken from the certificate. For the moment only the X.509 certificate SHA-256 thumbprint (x5t#S256) is allowed, for the which the computation method is mentioned in https://datatracker.ietf.org/doc/html/rfc7517#section-4.9 and https://datatracker.ietf.org/doc/html/rfc7515#section-4.1.8. • the X509 thumbprint of the certificate used to sign the JWT.

2.3.2. Payload

The JSON schema of the data model is publicly available.

Version *eessi:prc:1.0* can be found here :
https://api-pilot.ebsi.eu/trusted-schemas-registry/v3/schemas/z5fqhFPcuBeYZHnUbibhmtxDdoPhv4husBnvX4g5g8CeG

Encoding of strings is in UTF-8.Name	M/O	Format	Description
sid	M	*"type": "string",* *"pattern": "^eessi:prc:\\d+\\.\\d+$"*	Schema ID. Version/ID number pointing to the JSON schema for the JWT Payload. 10/2025 version: *eessi:prc:1.0*
jti	O	*"type": "string"*	Unique token identifier. This allows each issuer to add a way to uniquely identify their tokens if they feel the need for it. It could be used for auditing, support or revocation.
rid	O	*"type": "string",* *"format": "uri"*	Revocation list id. This URL contains the blacklist of jti's which have been revoked. These revocation lists are limited in size. Each jti is appointed a predefined revocation list id in advance upon which he must be published for

			revocation if needed. This list will typically live at the issuer side. 10/2025 version: For the moment the verifier provided by EC will not consult these revocation URL's yet.
prc	M		The actual prc, see below

The prc contains following elements:

Name	M/O	Format	Description
ic	M	*"type": "string", "enum": ["AT","BE","BG","HR","CY","CZ","DK","EE","FI","FR","DE","GR","HU","IE","IT","LV","LT","LU","MT","NL","PL","PT","RO","SK","SI","ES","SE","IS","LI","NO","CH","UK"]*	Country code of the card issuer
fn	M	*"type": "string", "maxLength": 40*	Family name of the holder
gn	M	*"type": "string", "maxLength": 35*	Given name of the holder
dob	M	*"type": "string", "pattern": "^[0-9]{4}-(0[0-9]|1[0-2]|00)-(0[0-9]|[1-2][0-9]|3[0-1]|00)$"*	Date of birth of the holder. Pattern based on ISO 8601, but flexible type that allows the absence of an exact day or month, for persons for which this information is not known. Format yyyy-mm-dd, yyyy-mm-00 or yyyy-00-00.
hi	M	*"type": "string", "maxLength": 20*	Personal identification number of the cardholder.
in	M	*"type": "string", "maxLength": 21*	Name of the issuing institution. Combined with 'ii' must not exceed 25 characters.
ii	M	*"type": "string", "minLength": 4, "maxLength": 10*	Identification number of the institution issuing the card.
ci	O	*"type": "string", "maxLength": 20*	EHIC card ID = Institution ID (4-10) + Card Serial Number (10). Optional for PRCs without EHIC.
sd	M	*"type": "string", "format": "date"*	Start date of PRC entitlement. ISO 8601 format (yyyy-mm-dd).
ed	M	*"type": "string", "format": "date"*	End date of PRC entitlement. ISO 8601 format (yyyy-mm-dd).
xd	O	*"type": "string", "format": "date"*	Expiry date of EHIC Card, if present. ISO 8601 format (yyyy-mm-dd).
di	M	*"type": "string", "format": "date"*	Date of issuance of the PRC. ISO 8601 format (yyyy-mm-dd).

The correspondence with the fields as described in S2 is as such:

Name	Field in S2	
ic	Field 2	*Issuing Member State* 2. ...
fn	Field 3	*Card holder-related information* 3. Name: ..... 4. Given names: ..... 5. Date of birth: .../.../..... 6. Personal identification number:
gn	Field 4	
dob	Field 5	
hi	Field 6	
in	Field 7 part 2	*Competent institution-related information* 7. Identification number of the institution: .....
ii	Field 7 part 1	
ci	Field 8	*Card-related information* 8. Identification number of the card: 9. Expiry date: .../.../.....
xd	Field 9	
sd	Field (a)	*Certificate validity period* (a) From: .../.../..... (b) To: .../.../.....
ed	Field (b)	
di	Field (c)	*Certificate delivery date* (c) .../.../.....

As can be seen in the above, no other technical dates have been added other than the dates already present in the PRC.

- **Date of issuance (di)**: This is considered to be both the date of the issuance of the PRC itself and the date of signing of the JWT in which it resides. This means that it is not possible to sign the JWT on a later date than the issuing of the PRC itself. This date will be matched by the verifier to check whether the certificate used to sign the token was valid on that day. If the issuer wants to be able to cache the ePRC, they must cache the signed JWT as well.
- **Start date of validity (sd) + end date of validity (ed)**: These dates represent both the validity period of the PRC itself, as the validity period of the JWT in which it resides. This period will be matched against a health care treatment date. The date of issuance should lie inside this period.

2.3.3. Validations

The following business rules should be held up by the issuer and should be verified by the verifier:

- **dob** <= **sd**

- **sd** <= **ed**
- **sd** <= **di**
- **di** <= **ed**
- If present, then **xd** >= **ed**

The following business rules should be followed by the issuer, but are not verified by the verifier:

- confirm length payload/prc/ii + payload/prc/in = max 25
- confirm payload/prc/ci contains only digits
- confirm payload/prc/ii contains only digits

2.4. Signature

The signature should be done with the certificate known in the EESSI repository for the authentic source of the entitlement (PRC) present in the JWT.

The authentic source of the entitlement is defined by concatenation of the elements: '**ic**', '**:**' and '**ii**'. This combination will constitute the OfficialID of the institution accredited to issue EHIC as known in EESSI.

Notice that as such the JWT does not include a separate field to indicate the issuer (the holder of the private key of the certificate) of the JWT, being possibly different from the authentic source of the entitlement. Based on the authentic source present in the payload, and the key identifier in the header (**kid**), the verifier will resolve the public key used for signing entitlements from that authentic source at the time of issuing. The Resolver API (see section 6) will be responsible to correctly identify that the given key-thumbprint is linked to the given authentic source, with or without another intermediary institution acting as issuer on their behalf.

example John Doe :

Given the following PRC header and payload:

```
{
  "typ": "JWT",
  "alg": "RS256",
  "kid": "EESSI:x5t#S256:bX5DsQFQ6zHa2om0QFs5nD0AEpesqoSZZb3TzqqSKuw"
}
{
  "sid": "eessi:prc:1.0",
  "jti": "49f095e2-2dbf-4525-99b9-3c8956083e97",
  "rid": "https://example.org/revocation/list",
  "prc": {
    "ic": "BE",
    "fn": "Doe",
    "gn": "John",
    "dob": "2011-11-11",
    "hi": "11111111111",
    "in": "CM",
    "ii": "0120",
    "ci": "12345678910111213141",
    "sd": "2025-09-01",
    "ed": "2025-12-01",
    "xd": "2025-12-31",
    "di": "2025-09-22"
  }
}
```

The authentic source of this payload is the EESSI institution with OfficialID `"BE:0120"`.

The JWT should be signed with the certificate known in EESSI (identified by the `"kid":"EESSI:x5t#S256:bX5DsQFQ6zHa2om0QFs5nD0AEpesqoSZZb3TzqqSKuw"`), allowed to be used for

this institution `"BE:0120"` and for accrediting EHIC at the moment of issuing (`"di":"2025-09-22"`).

Whether the actual owner of the private key of the certificate (issuer) is equal to the authentic source or not, is not represented in this dataset.[4] In fine, all that matters is that the certificate used to sign the token is recognized as a certificate allowed for use for the authentic source in question at the moment of issuing, regardless of whether this authentic source has delegated the signing part to another issuer. This information is present in the EESSI IR and will be available to the Resolver API.

[4] In Belgium these will in fact be different. The Crossroads Bank of Social Security (CBSS) will act as the issuer (holder of the private key) for the all the health insurance funds, which are the different authentic sources known in EESSI. As can be seen, as such CBSS is not represented in the PRC datasets.

3. TECHNICAL SPECS ON CREATING THE QR CODE FROM THE SIGNED JWT

Once the signed JWT is made the issuer will need to transform it into a QR code.

3.1. Specifications

These specific steps are needed to create the QR code:

- `encoded_jwt` = The signed JWT as created in section 0 encoded as a triplet base64 string delimited with a dot `.` between the parts: header, payload and signature.
- `compressed_encoded_jwt` = Compress the `encoded_jwt` using ZLib (encoding UTF-8)
- `base45_compressed_encoded_jwt` = Transcode the `compressed_encoded_jwt` using Base45.
- QR = Create a QR coding from the `base45_compressed_encoded_jwt` with
    - QR Code Model 2 format (ISO/IEC 18004:2015 international QR code specifications)
    - Encoding mode: Alphanumeric encoding
    - Maximal error correction level: L
    - Smallest version for the number of characters that need to be encoded, given the aforementioned error correction level.[5]
    - Mask pattern with the lowest penalty score.[6]

Note

When displaying the QR code, it is recommended to use as much of the available space (on the chosen display or medium) as possible. If displaying the QR code on a smartphone, use the full width of the screen. When placing the QR code in a PDF, as described in section 4, allocate an area of at least 6cm².

Met opmerkingen [GDM1]: Capacity of what?

Met opmerkingen [WD2R1]: I reworded. Is it better?

3.2. Design Rationale

The reasoning behind this structure is that alphanumeric encoding for QR codes uses the same 45 characters as used (and designed for) base45 encoding, allowing for a more optimal creation of the QR image. Although the base45 encoding is less dense (given it only has 45 characters to represent the data), and hence requires more bytes (e.g., longer strings) to represent the same amount of information, a QR code created from a (longer) base45 string will be more optimal than the (shorter) base64 string representing the same data.

Since the signature of a JWT results in a triplet base64 string, we need to transform that information into a base45 string, to benefit from the optimality of the QR code's alphanumeric encoding. However, to counteract the loss in density caused by the reduction in character set size (64 -> 45), it is first compressed using ZLIB before being transcoded into base45.

[5] https://www.thonky.com/qr-code-tutorial/character-capacities and https://www.qrcode.com/en/about/version.html

[6] https://www.thonky.com/qr-code-tutorial/data-masking#choose-the-lowest-penalty-score-for-the-eight-mask-patterns

Despite efforts to avoid duplicate information in the JWT itself, the number of characters in the base45 string for our use case could still be quite large (950+). Therefore, we opted for a small level of error correction to ensure the QR code had as much information capacity as possible.

3.3. Example

example John Doe:

Given the PRC header and payload as described in section 2; and given a dummy certificate, for which we include the private key in order for the audience to try this out themselves, the different steps for creating the QR code would give the following output.

Private key:

```
-----BEGIN PRIVATE KEY-----
MIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKYwggSiAgEAAoIBAQDZKEnq48Besrfe
5BM0uHDyXQZTCnwdCnbJPcOPN8dkm1pWCgWuU59saNPxmqXOjsyQJ/SOBWMa8nc4
StScWydFl1E/YbSk3gzGQn5kbwjnLRuU7qP0QhhFpqS/rWtHPqL9An6HEgvLnPiM
s9TSSOCjLobV9By1SrUsSXF+MjGFdpWvOVwAcgJHQiZzJ1F1T+LNW52IZQdNAgad
CLD8LxBZ3qAp3g6JaerGlJtdORNObzO/pY2aG9x+E0729ThkxLOcyzuznpjUffv7
t7SVBuiIAeF++zHDsZo08EeQSrehPeWGRhV06HmJSiFv6XqCHFQzqAX/EvBzKQbE
ByADiRfDAgMBAAECggEABHDy1Ln5tODE1UB8NWd8RTNVLhztyDju5ai8t3t1jtTa
i8iYBdEHVqlCLxPG8QAx+39wXKM25qsveFhCHDyFA22tJ2Bb2G9Y0DPgmonmzbY6
H9lNmHDtnCcPVtg4yck503YvQ5tBNdXXriZTY5rtSokTZYwg4NhOokGSRPlz0n3p
D+s8f0iCTgpOFwxmF8fvaJ/njw85OpTGoCaqy6kGexxgoUeVDR52pGs8PSN6z4Fb
/pCTHW05TPPgC7Olr+i8rowyequ828hWhH1j7ul/9T3qvkEGVr8MZ6MV+keOanFV
9VePGnIOTVgTWWZ+ha37LzzAIgXqk4zyvifuH5q2uQKBgQD0a7OJ35hwUO9wRsP3
C9WtEZGnLYpiMKZi6TwC2MsFVMJDwl6dtq9W+rSDiB5HTRKPf9XMRc0A4Tk3RwV5
QVJh8j1PACuY/t0foo2p7TecCJK9W38bYPmARN0rjNw7OJKT8GMfU3NddPoZlf9K
Rz5qeD7a2ICcFEEmHZ/B7BQUvQKBgQDjcfEU8RlxIhKC+3aNoOzFo3jxxyCfSnN5
gYLm3ZVrs8XjgT3t2ylmvlaDWyzCpjlGe+4/pFW44ZPOTDsdxFAEgztonqcaqmzq
c4oYmqzxsjMbconKKGQY6mDa8N3U99ZYA3vdfb4OH7NHzt3qa0i02Qcdwquubqi J
qaZHlEXmfwKBgD3NOcFj2wyTOme+TPCbenTfW1/VIQeA26Fo7faeUH9RlilB7WCd
f6pTi+8yLM3EjCvim/GhuJs/V5NFjvnA7Y68jIn8Ngnib5D/LijIRWZbuJbOms4W
cB/hybaoUEtdG5ryjm2uUR5RZKexiLKEXyDWtLXrRmGwsWRs2ydShSVVAoGAGIlM
NrQjyH6ifJrgGTtJD7Eo9PxEWqrPdkYwdvNkkS/do0NoCObhuoHCF32YgI8qEp5/
/0GdNcqwBUtX/UgWUcZOfGTRJiFIiqpYNXtKOPC+brGWh8Z/tehi9nuRks4Bw5Ds
peJQxHlK/+4jJozYVRosmFtN5g1uBRvWIlaui8UCgYAftP2s0kzu0vtuj8ee+nvO
TBfovmtlcuDm+xE7tL8Is+wCZH34Hsz/V4m2Opn48jbL9C5U9/A4gC8L+hKIY2GZ
PAj44jvC45U9fziMxCQ2NMqrYVJJ3YQSXOzwiNIRnnfHY54lu6sNndAKjhxpWj6L
JxaDN4okGVQVITxUVrc/PA==
-----END PRIVATE KEY-----
```

X509 Public Certificate:

```
-----BEGIN CERTIFICATE-----
MIIC0TCCAbmgAwIBAgIULriDcYjpSgBZD9VYWvogZPWib0YwDQYJKoZIhvcNAQEL
BQAwIDEeMBwGA1UEAwwVRHVtbXkgSm9obiBEb2UgSXNzdWVyMB4XDTI1MDkyMjEx
MTc0NFoXDTMwMDkyMjExMTc0NFowEzERMA8GA1UEAwwISm9obiBEb2UwggEiMA0G
CSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDZKEnq48Besrfe5BM0uHDyXQZTCnwd
CnbJPcOPN8dkm1pWCgWuU59saNPxmqXOjsyQJ/SOBWMa8nc4StScWydFl1E/YbSk
3gzGQn5kbwjnLRuU7qP0QhhFpqS/rWtHPqL9An6HEgvLnPiMs9TSSOCjLobV9By1
SrUsSXF+MjGFdpWvOVwAcgJHQiZzJ1F1T+LNW52IZQdNAgadCLD8LxBZ3qAp3g6J
aerGlJtdORNObzO/pY2aG9x+E0729ThkxLOcyzuznpjUffv7t7SVBuiIAeF++zHD
sZo08EeQSrehPeWGRhV06HmJSiFv6XqCHFQzqAX/EvBzKQbEByADiRfDAgMBAAGj
EDAOMAwGA1UdEwEB/wQCMAAwDQYJKoZIhvcNAQELBQADggEBAC3CwLhsD4Bg9+ew
GTdEGkY0OKo/WLzbOObgiOIU1xb1vRjza0QQDTzVcFwnk1JDgUE+b9SdFvwKABl1
DsW0uZy65wou+0mEVQxAcIaRV6JMJirKXIFHe5QBx1uC+QSOyI5MVGsgFIVdweeX
LsyOjAKMuNEiy4cEyGjG7qhFufg+vnf9+/oj7ypUtQAyEO5Fse6N2JcQQeJSw7xe
2lyRjX6Rd4Myi3tShNUr4INm/JjR+PFS1IDHN4amNxCzzUnOfFSrQ4e3aShzP6Vk
PgHK3La8S2ILjhgTvuPJIH1Y1dWOSlLtGYP9kSOthoUm5ZbPjjEr8BzZjH1oZuc9
hwNuuhs=
-----END CERTIFICATE-----
```

encoded_jwt : (888 characters)

```
eyJhbGciOiJSUzI1NiIsImtpZCI6IkVFU1NJOng1dCNTMjU2Oms5ZFlhUUdsam0weEpSQWtiTWJHVWMzS1RzOHctN
1lyOC1aeEJrQ29MbHciLCJ0eXAiOiJKV1QifQ.eyJzaWQiOiJlZXNzaTpwcmM6MS4wIiwianRpIjoiNDlmMDk1ZTI
```

tMmRiZi00NTI1LTk5YjktM2M4OTU2MDgzZTk3IiwicmlkIjoiaHR0cHM6Ly9leGFtcGxlLm9yZy9yZXZvY2F0aW9u
L2xpc3QiLCJwcmMiOnsiaWMiOiJCRSIsImZuIjoiRG9lIiwiZ24iOiJKb2huIiwiZG9iIjoiMjAxMS0xMS0xMSIsI
mhpIjoiMTExMTExMTExMTEiLCJpbiI6IkNNIiwiaWkiOiIwMTIwIiwiY2kiOiIxMjM0NTY3ODkxMDExMTIxMzE0MS
IsInNkIjoiMjAyNS0wOS0wMSIsImVkIjoiMjAyNS0xMi0wMSIsInhkIjoiMjAyNS0xMi0zMSIsImRpIjoiMjAyNS0
wOS0yMiJ9fQ.sdWKtak_zI-
R3ik2pbKj4_UY_XKuWTZOvuIhN9O5SrfK0hhgjC9aQLWu1EdNVWZ_WvK5Zntokk6AjFhxOf3r8T1laRM47sI4yAMV
hFInEsVi-
IIovqXvNhkw5F6eLGtWZALWLFS8Hc5TpVpnjwYCNIh1yNt7k8TVq0thQQR6tV_aV_s4YA7Q7qNWof_pTtaVqSVJ-
tB48bXKYSNmIl7TUHgEaS4nPou40IOG27OcEktgiZzlVojaE_9YdbIR8z4wheRZbdli0tyUh-
UNjs2EtzfAdiZKxoxJmlCWi6qOehk6eLkcAN3MvTtzs00m-4qFagTWz8xhB86grz5801gllg

compressed_encoded_jwt: byte[]

base45_compressed_encoded_jwt: (948 characters) (note that space is a valid character in the base45 alphabet)

6BF*ZB8XP+MKIT27G4WA0$SGX.L8A84D3/R8QDESD42A0-H4R SN7MBZ6YZE74DI/JX9R0.6O-
TO3A5ZPU24.GG+57RA8XM35U0 24ML57KIO6R/NFVD4ORH1SEX.EJ.LS-GI-MRH5HYPN-
4YDHNEJ16HGDOXAAFKQHXKNLLYGSO.QIP66:55P6/0T$KEUOD$LR$M6YU3Z1R937S0A0GRTJ8671WKTF$H-:D+
M9M9LFT/ESZ1AR6O9:OGTKRFB.*JH85EE2J-78UB7121PLL-OS*5NVG5%QTQ3NAE/3IDVHX OSTLBW92.S
O5W60K.JO651-
N6XQX1CX%IDIQ+W7DUV8X6ED9RNUA%MJF9H3SWSE8HOASN391AQA**LXT9NKK+E9A9SI1TR0A61WG1T3MMEDTI37B
UELRHI+120T% 4MZV:CGI+TD1QIWQ%JV9/CZ*72HHYYT$ 1NKU2%347KQ+16LVNH3X0S$JT 0E$4O%KQP.7-V6-
18DH0O IW+FG8NV%UHAU/EW8TDB5GI-V*EWBBT1F75-SUQU3RK6J9$E6N9T6-PZ97PJBY97MSV2
H4ALISIJUTV1K3C47YBN4CRVKW7BOWDQ$OSEPIZ6E35Z*U8:BDW0$:SZDHC3L.AB28S$2C$
LWEHU0V0N1+BCHL4WIG8UDZ$7/JSNMFZ.B$GH$CCJM6Q.5M/R/%O0PG 947FSANDVOKB6NU56S68BZ4D67
55*GF*9OBKQ/IH%*LH3E68SEIG-
1WV:D/U9C:8SKRLNVP2C7/9057GU281BIV7U+CCKT7CO5OAFCG1AR6/DDF1NPE:2J -
NN*P$82T16S3CLNF:H6S/BJW4J$SROMYBHH+AJ0T-WRSA7/9V
6S7PPNOSK6H5E6SZ1/OQJPJ8:20RCD758U9B7BD2I2HNKOR+:UBBWA/24$8/UP0BWSYG/A6

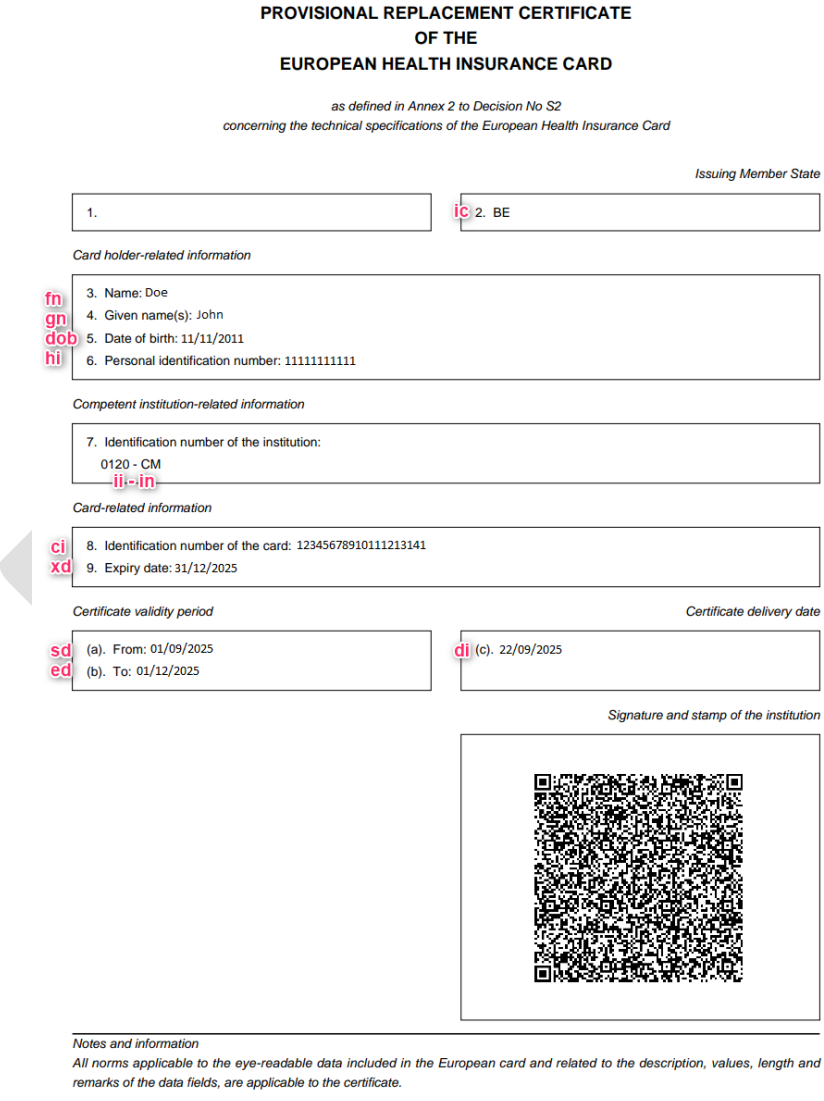
QR:

Version: 18
Error Correction: L

4. FORMAT AND SPECS FOR THE PDF CONTAINING THE QR CODE

The PDF format should adhere to the specifications set out in Decision S2. PDFs can be in either English or any official language of the relevant Member State. After scanning (and verifying) the data included in the QR code, the verifier should be able to display the labels in the language of the healthcare provider verifying the document.

The date fields should be represented in the format DD/MM/YYYY. The correspondence with the fields from the payload can be seen here:



PROVISIONAL REPLACEMENT CERTIFICATE
OF THE
EUROPEAN HEALTH INSURANCE CARD

as defined in Annex 2 to Decision No S2
concerning the technical specifications of the European Health Insurance Card

Issuing Member State

1.

ic 2. BE

Card holder-related information

fn 3. Name: Doe
gn 4. Given name(s): John
dob 5. Date of birth: 11/11/2011
hi 6. Personal identification number: 11111111111

Competent institution-related information

7. Identification number of the institution:
0120 - CM
ii in

Card-related information

ci 8. Identification number of the card: 1234567891011213141
xd 9. Expiry date: 31/12/2025

Certificate validity period

sd (a). From: 01/09/2025
ed (b). To: 01/12/2025

Certificate delivery date

di (c). 22/09/2025

Signature and stamp of the institution

Notes and information
All norms applicable to the eye-readable data included in the European card and related to the description, values, length and remarks of the data fields, are applicable to the certificate.

5. FUNCTIONAL DESCRIPTION OF THE VERIFICATION FLOW

The verification process entails the following basic activity flow. Additional features, such as identifying the human through a user system and adding extra verifications to confirm their verifier-capability (can they act as verifier) are not described here.

The following process focuses on the core functionality of the verification mechanism needed to be able to conclude that a signed JWT as read from a QR on an ePRC PDF constitutes a valid and usable insurability entitlement.

Some definitions

- Verifier: the human using the verification mechanism;
- Verifier App: the application implementing the verification flow described here.

5.1. Step 1: Get the treatment date

Verifier App needs to get the health care treatment date from the verifier. This date must be in the interval [1900-01-01; current date]. By default, the Verifier App can presume the treatment date being the current date.

5.2. Step 2: Decode the JWT present in the QR

Verifier App needs to

- Read the QR into a base45 string
- Uncompress this (ZLIB) to an encoded jwt
- Decode the jwt to the original content

example John Doe:


```
{
  "typ": "JWT",
  "alg": "RS256",
  "kid": "EESSI:x5t#S256:bX5DsQFQ6zHa2om0QFs5nD0AEpesqoSZZb3TzqqSKuw"
}
{
  "sid": "eessi:prc:1.0",
  "jti": "49f095e2-2dbf-4525-99b9-3c8956083e97",
  "rid": "https://example.org/revocation/list",
  "prc": {
    "ic": "BE",
    "fn": "Doe",
    "gn": "John",
    "dob": "2011-11-11",
    "hi": "11111111111",
    "in": "CM",
    "ii": "0120",
    "ci": "12345678910111213141",
    "sd": "2025-09-01",
    "ed": "2025-12-01",
    "xd": "2025-12-31",
    "di": "2025-09-22"
  }
}
Signature:
TLo2V7FGe1zCdr6nr3VJusOt0oQuDvgDcncUsphLHPKAgevBW_vb6aTCA4ed3e29sPMIWiiSm7oK8AQvU9
s_m6uCCO9_jqXUxFWTIf6QcRdVdPIzFEMstamTEitvOWlM9nF0TOT0DZeXyMRIlVm64JDlGECV9a41BBII
v4u2AjmAYKf5xMDVHkaUyAApXGuX1qGl251YjA3ygOynbM_OD99jFDOuwj8amfWyeyieLvlZk6_BLEMoxc
L6A_e1aVO2db8E5wmlkDcxdtDYMGakabGRU8zl2FJ8NhgVGNQ-
eTNBQeeApcKVQANaDk4KoFzsPnHQrdSJbssaq5Z3jn0V0Q
```

Note on scanning conditions

When scanning a QR code, factors such as lighting, camera resolution, the flatness of the surface and any damage to the code itself can affect its readability. These conditions should be considered within the verification process. For example, it may be advisable to advise the healthcare provider to use a smartphone camera or a dedicated scanning device.

5.3. Step 3: Validate the JWT header and payload

Verifier App needs to

- Resolve the schema mentioned in payload/sid
  - eessi:prc:1.0 :
    - https://api-pilot.ebsi.eu/trusted-schemas-registry/v3/schemas/z5fqhFPcuBeYZHnUbibhmtxDdoPhv4husBnvX4g5g8CeG
- Validate payload to the schema mentioned in payload/sid
- Validate header contains
  - kid with pattern "^EESSI:x5t#S256:[A-Za-z0-9_-]+$"
  - alg in enum [ES256, RS256]

Note on the offline verification

To enable offline verification, the Verifier App needs to cache the schema's fetched from the online sources. This cache should have a maximum time-to-live of 24 hours.

5.4. Step 4: Resolve public key and verify signature

Verifier App needs to use the Resolver API to fetch the public key for the kid given in the payload. Also see section 6. The parameters needed for the Resolver API can be found as such:

- kid = header/kid
- countryCode = payload/prc/ic
- officialID = payload/prc/ii

The Resolver API will return the single issuer for the given input and his accreditation periods for EHIC as described in section 6.

Verifier App then needs to

- Verify the single issuer is correctly returned for the officialId / countryCode given in input
- Verify the single issuer contains a certificate / public key for the requested hash
- Use the public key found to verify the JWT signature

Note on the offline verification

To enable offline verification, the Verifier App must fetch and cache the set of public keys from the Resolver API, as described in section 6. This cache should have a maximum time-to-live of 24 hours.

5.5. Step 5: Do business validations on the payload

Verifier App needs to check that the institution is allowed to issue and sign PRC on date of issuance for the given PRC period:

- Public key must be valid on payload/prc/di[7]
- Institution must be accredited for the portableDocument EHIC on payload/prc/di[8]
- Validity period of the PRC (sd->ed) must fall completely in the accreditation period for portableDocument EHIC.

Verifier App needs to check that the payload/prc is logically consistent with respect to the dates:

- **dob** <= **sd** (consider dob with '00' yyyy-00-00, yyyy-mm-00 as '01')
- **sd** <= **ed**
- **sd** <= **di**
- **di** <= **ed**
- If present, then **xd** >= **ed**

Optionally the verifier can also verify the following, but it should not block the further verification:

- confirm length payload/prc/ii + payload/prc/in = max 25
- confirm payload/prc/ci contains only digits
- confirm payload/prc/ii contains only digits

5.6. Step 6: Check for revocation

The Verifier App needs to check revocation if payload/jti and payload/rid are both present.

- verify that payload/jti is not present on the revocation list exposed on payload/rid

Note

This version of the PoC does not include revocation implementation.

[7] Notice that the validity period of the certificate is expressed in datetimes, while the payload/prc/di is only a local date. This has as consequence we cannot know at what time the issuance was done. Therefore, a choice is made to verify the payload/prc/di against the local date parts of the validity period of the certificate. So in the certificate of the example above ("validFrom" : "2024-08-14T02:00:00GMT+0200", "validUntil" : "2025-08-14T01:59:59GMT+0200") any payload/prc/di in the interval [2024-08-14, 2025-08-14] should be considered ok.

[8] Notice that the validity period of the accredidationPeriods is expressed in datetimes, while the payload/prc/di is only a local date. This has as consequence that we cannot know at what time the issuance was done. Therefore, a choice is made to verify the payload/prc/di against the local date parts of the accredidationPeriods. So in the accredidationPeriods of the example above ("validFrom" : "2004-01-01T00:00:00Z", "validUntil" : "2005-01-01T00:00:00Z" + "validFrom" : "2020-01-01T00:00:00Z") any payload/prc/di in the intervals [2004-01-01, 2005-01-01] and [2020-01-01,∞[should be considered ok.

Note on the offline verification

To enable offline verification, the Verifier App needs to cache the various revocation lists at its disposal. This cache should have a maximum time-to-live of 24 hours. If, during an offline verification, the revocation list referenced in rid is unavailable in the cache (or outdated), the report described in section 7 should indicate this.

5.7. Step 7: Verify the treatment date

The Verifier App needs to validate the treatment date with respect to the ePRC.

- verify treatment date lies in period [payload/prc/sd, payload/prc/ed]

Notice that this does not consider the payload/prc/xd

5.8. Step 8: Output the result

The Verifier App needs to output the result (OK / NOK) including

- In case NOK : the reason (e.g., which above step(s) failed)
    - Export of the verification result in the format as described in section 0
- In case OK :
    - Visualization of the data in payload/prc for the verifier to cross-check this with the actual data printed on the PDF. Ideally this visualization is done in the language of the verifier.[9]
        - The Verifier App can request the verifier to indicate whether following checks were done:
            - Visualized name(s) match name(s) on identity card of citizen
            - Visualized birthdate matches birthdate on identity card of citizen
    - Export of the verification result in the format as described in section 7. The rendering of the PDF containing the validation data should be in the language of the country of mentioned in payload/prc/ic. In case the country has more than one official language (Belgium, Malta, Ireland, Luxembourg, Switzerland) the output will be English.

[9] The interface of the Verification App could offer the possibility to the user to choose one of the EEA/EER/EFTA official languages.

6. TECHNICAL SPECS OF THE RESOLVER API

This specification is considered mature and provides a stable foundation for implementing the current use case today. Its design emphasizes interoperability with the Verifiable Credential (VC) ecosystem as it is understood today. However, because other VC framework specifications are still evolving and not yet fully mature, some degree of divergence is likely. This may result in interoperability gaps as new frameworks and standards emerge.

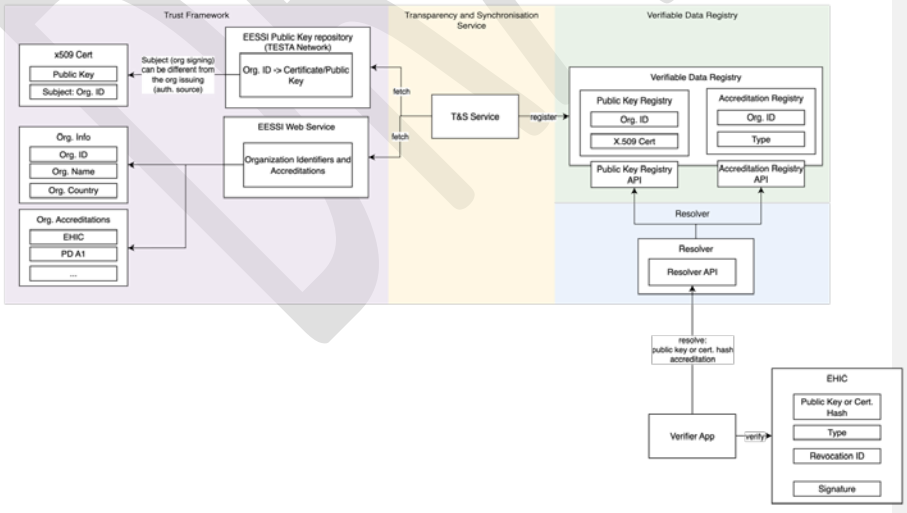
To minimize the impact on the use case implementation and making it future proof, we're introducing a component called **Resolver API**. Resolver API is a service that defines stable and future proof interfaces to fetch information about the Verifiable Credential issuer identities, identifier, their accreditations, and other additional information as required from the different use cases. The service has been designed to be able to fetch information from different trust registries. Hence, the Resolver API introduces an **abstraction layer** between the EU, national or sectorial trust frameworks, and the Verifier applications and service.

By moving the logic of fetching information from trust frameworks and serving it in a harmonised way, ensures that the implementations of Verifier Apps remain future proof, even if the format, location, or content of the trust frameworks changes.

Today, the Resolver API allows the Verifier App to fetch the certificate(s) / public key(s) and EHIC accreditation periods for the issuer of the JWT. This API is exposed by the European Commission and can be used by any actor implementing the/any Verifier App.

6.1. Resolver Architecture

In the diagram below, we present the high-level Resolver architecture as it operates for the EHIC use case.



The architecture is built around four key components that work together to ensure trust, transparency, and long-term interoperability:

1. Trust Frameworks
   These are EU, national, or sector-specific frameworks that publish information about organisations and their accreditations.
2. Transparency and Synchronisation Service
   This service retrieves information from different trust frameworks and publishes it into the Verifiable Data Registry (VDR) in a harmonised format.
3. Verifiable Data Registry (VDR)
   The VDR is a public, cryptographically protected repository that provides secure, transparent, and resilient access to verified information about organisations, accreditations, and public keys.
   - It ensures data consistency across different trust frameworks.
   - It preserves both harmonised data and original records (including signatures).
   - It offers transparency and auditability, similar to the role of Certificate Transparency in the web ecosystem.
4. Resolver API
   The Resolver API, which can be implemented by any service provider, delivers information from the VDR to verifier applications in a standardised and future-proof format. This abstraction shields verifier apps from changes in trust framework implementations.

The architecture has been designed to address three fundamental challenges: interoperability, futureproofing, and trust at scale. Each component contributes specific technical capabilities to achieve these goals.

1. Separation of Concerns

   By introducing the Transparency and Synchronisation Service, we decouple the verifier applications from the heterogeneity of existing and future trust frameworks. Trust frameworks differ in their data models, publication methods, and governance structures. Without this layer, each verifier app would need to implement and continuously maintain integration logic for every framework, which is both inefficient and error prone.

   The Synchronisation Service normalises these different sources into a harmonised representation before publishing them to the VDR.

   This ensures that changes in framework governance, APIs, or data formats do not propagate downstream to verifier applications.

2. Cryptographic Transparency and Auditability

   The Verifiable Data Registry (VDR) is the cornerstone of the architecture. Its design is inspired by Certificate Transparency mechanisms in the web PKI ecosystem but extended for a broader trust context. All records stored in the VDR are cryptographically verifiable, ensuring data integrity and authenticity. Append-only principles provide immutability, preventing retroactive tampering or deletion of historical records. Independent third parties can audit the VDR, ensuring accountability of framework operators and data publishers. Historical snapshots enable retrospective validation of credentials, which is critical in dispute resolution or compliance audits.

3. Interoperability Across Frameworks

   Because multiple trust frameworks coexist at EU, national, and sectorial levels, direct interoperability is a challenge. The VDR addresses this by enforcing a common, harmonised data model while still retaining:

   - Original signed data, to preserve evidentiary value and legal validity.
   - Framework-specific metadata, enabling precise traceability back to the source.

   This dual approach provides flexibility for simple automated verification while also supporting advanced compliance and legal use cases.

4. Resolver Abstraction

   The Resolver API provides a clean and future-proof interface for verifier applications. It abstracts away the complexity of interacting with the VDR or individual frameworks directly and allows to implement additional, use case-specific interfaces and services.

   Service providers can operate their own resolver instances, which allows for distributed and decentralised implementations while maintaining interoperability. The standardised API ensures verifier applications receive data in a consistent format, regardless of framework evolution or changes in the underlying registry technology.

5. Security, Reliability, and Scalability

   Security is achieved through cryptographic protections, ensuring that no entity can forge or alter published trust data. Reliability and resilience are built into the VDR by design, with distributed deployment options to prevent single points of failure. Scalability is ensured by separating data ingestion (via Synchronisation Services) from data serving (via Resolver APIs). This modular approach allows the ecosystem to grow as more frameworks and credentials are integrated, without degrading performance for verifiers.

6.2. Data model

The source of data behind the Resolver is the Central Trust Framework which

- Registers the names, data and accreditation periods of the organizations allowed to issue ePRC.
- Holds the public keys for the issuers allowed to sign ePRC.

For this version of the PoC, the EESSI Institution Repository is used as Central Trust framework. Later other Central Trust frameworks could be added as additional data source for the Resolver.

The Resolver acts as a bridge to allow the/any Verifier App to connect to the Central Trust framework in a secure, decoupled way and making abstraction of the actual Trust framework with which it is connected.

The central elements of information to be held by the Resolver per issuer are

- Country(code) of the issuer
- Official ID of the issuer
- Name of the issuer
- List of all accreditation periods in which the issuer is allowed to issue EHIC/ePRC.
- List of all (historical) certificates used by the issuer. This allows the verification of ePRC issued at any point in time in the past with a certificate valid at that time.

Important remark on the definition of the issuer

It is possible that the institution holding the actual information on the ePRC, the competent institution, is not the same institution holding the private key to sign the electronic attribute. In the context of the Resolver, however, the concept of the issuer is defined as the competent institution of ePRC enriched with the certificates of the institutions allowed to sign on their behalf. Abstraction is made from the fact that both institutions may not be equal, or that one institution holding a private key may even sign the electronic attributes of multiple competent institutions.

The link between the competent institution and the signing institution is ensured (and abstracted) by the Resolver. This is the reason why it suffices that the payload of the ePRC (section 2.4) only mentions the competent institution. Together the identifier of the used certificate (**kid**) and the identification of the competent institution (**ic+ii**) will be enough to 'resolve' the issuer.

6.3. Resolver API

Input for the Resolver to fetch the issuer at hand is:

- kid = header/kid
- countryCode = payload/prc/ic
- officialID = payload/prc/ii

The Resolver is then responsible to determine from the **kid** the Trust Framework, the thumbprint and match that with the country and id to search for the issuer.

Output are the issuers found for the given criteria:

- issuers: list of issuers filtered on countryCode + officialID. Per issuer[10] :
  - officialID
  - countryCode
  - name
  - certificates: list of certificates filtered on the thumbprint from the kid. Per certificate[11]
    - x5t#S256
    - validFrom
    - validUntil
    - certificatePEM
  - accreditationPeriods : complete list of accreditationPeriods for the given issuer. Per period:
    - portableDocument
    - validFrom
    - validUntil

[10] If countryCode + officialID are given, this is maximum 1 issuer

[11] If countryCode + officialID + kid are given, this is maximum 1 certificate

For a particular verification the Verifier App will always be able to provide the 3 parameters, which should result in a single issuer with a single certificate.

For caching reasons and enabling the/any Verifier App to work in an offline mode, the Resolver API will also provide the possibility to fetch the entirety of the list of issuers with their entire list of (historical) certificates.

The SwaggerUI for the TEST version of the Resolver API can be found here : https://resolver-test.ebsi.eu/api/v1/docs#/Issuers/IssuerController_resolveIssuers[12]

Note on future evolutions

Currently, the input parameter **kid** exists in the pattern as described in section 2.3.1.

The first part of the **kid** indicates the Trust Framework (currently only 'EESSI'), while the second and third parts give the type and actual thumbprint of the certificate that the Verifier App is searching for.

However, when other Trust Frameworks are added to the Resolver in the future, the value (and pattern) of the kid will also change to reflect this.

The Verifier App can continue to use this implementation by providing the Resolver with the value of the **kid** found in the JWT, leaving the Resolver responsible for decoding the kid and using that information to search for the intended issuer's public key.

[12] In the current version of the Resolver API, the input parameter kid is not present yet. Only the x509Thumbprint from the header/kid part after x5t#S256 is present as parameter. This will be changed, shortly to allow the full kid as input parameter.

7. FORMAT AND SPECS FOR THE EXPORT OF THE VERIFICATION RESULT

After the Verifier App returns a positive/negative result, this verification needs to be exportable in a format usable in the reimbursement process between the health care provider and the competent institution mentioned on the ePRC.

The export consists of the following basic information blocks

1. The original ePRC
    a. The JWT
    b. Visualization of the payload (PRC)
2. The metadata of the verification
    a. Identification of the verifier (email address, …)
    b. The treatment date used to verify the JWT
    c. The timestamp of verification
3. The result of the verification steps
    a. Each verified step is indicated with a verification-result.
        i. PASSED: Passed Verification
        ii. NOT PASSED : Did not pass verification
        iii. WARNING : This is used for the non blocking validations from section 5.5 and for indicating that revocation is not present in the JWT.
        iv. N/A : Not implemented or not possible to verify
    b. Following verification steps are included. Once a NOT PASSED is reached further steps are not verified, nor mentioned in the export.
        i. Technical
            1. [PASSED/ NOT PASSED]
                a. Decodable QR / base45 / zlib / JWT.
                b. Recognized schema and valid JWT header/payload.
                c. Resolved public key.
                d. Valid JWT
            2. [PASSED / WARNING]
                a. Presence of Revocation URL and JTI in the JWT.
                    i. ‘PASSED’ means the URL and JTI are present in the JWT.
                    ii. ‘WARNING’ means the URL and JTI are not present in the JWT.
            3. [PASSED/ NOT PASSED / N/A]
                a. Verification result of the Revocation URL and JTI present in the JWT.
                    i. ‘PASSED’ means the URL and JTI are present in the JWT and no revocation was found.
                    ii. ‘NOT PASSED’ means the URL and JTI are present in the JWT and revocation was found.
                    iii. ‘N/A’ means either that revocation has not been implemented by the verifier or was not available (or outdated) in his cache, or that the URL and JTI are not present in the JWT.
        ii. Business
            1. [PASSED/ NOT PASSED]

        a. Treatment date is <= current date.
        b. Public key valid on issuance date
        c. Issuer accredited for EHIC on issuance date / validity period
        d. dob <= sd
        e. sd <= ed
        f. sd <= di
        g. di <= ed
        h. If present, then xd >= ed
        i. Treatment date in [sd,ed]
    2. [PASSED / WARNING]
        a. length ii + in = max 25
        b. ci contains only digits
        c. ii contains only digits
        d. Visualized name(s) match name(s) on identity card of citizen
        e. Visualized birthdate matches birthdate on identity card of citizen
4. The conclusion of the verification
    a. Approved (all steps done and result for all either PASSED, WARNING or N/A). In this case, also the identification of issuer is added:
        i. Country
        ii. OfficialID
        iii. Name
        iv. Kid
    b. Rejected (at least one step NOT PASSED)

The format of the export exists in two distinct ways

1. PDF with (in lang of CountryCode, EN if multiple lang exists there)
    a. [only when PASSED] The reconstructed ePRC from the QR
    b. The originally scanned QR
    c. The information blocks 2 + 3 + 4 above
2. Json with
    a. [only when PASSED] The payload of the original JWT
    b. The original QR code in base45 format
    c. The information blocks 2 + 3 + 4 above

Note

It is still up for debate whether the Verification App itself should sign the PDF with the export. In the current version of the PoC this is not done.

A schema for the Json included in the export has not yet been provided.

8. ANNEX I – EUDI COMPLIANCE

8.1. ePRC signature compliance with ETSI TS 119 182-1 V1.2.1

In this section we review the compliance of the digital signature of compact credentials with the ETSI TS 119 182 standard. Compact credentials are credentials that are aimed to be rendered as QR codes, such as ePRC.

8.1.1. Compact Credential Signature Profile (CCSP)

The Compact Credential Signature Profile (CCSP) signature defined in section 2 is a compact serialised JWS, compliant with the RFC 7515 standard.

The signature uses the following header parameters

- **alg**
- **kid**

The **kid** parameter in our current solution has the following structure:

pattern: `^EESSI:x5t#S256:[A-Za-z0-9_-]+$`

8.1.2. Requirements for JAdES Baseline-B

The following header parameters MUST be present

- **alg**
- **x5t#S256**
- **iat**

The following header parameters MAY be present

- **cty**: The cty header parameter should not be present if the content type is implied by the JWS Payload.

8.1.3. Mapping between the CCSP and JAdES Baseline-B

8.1.3.1. alg

In CCSP **alg** is required, hence it meets the ETSI requirements.

8.1.3.2. x5t#S256

In CCSP, the **x5t#S256** is present in the **kid** and the value maps directly to the **x5t#S256** claim, making it compliant with the ETSI requirements.

8.1.3.3. iat

In ePRC, the **di** payload claim maps to the **iat** header parameter with time set to 23:59 59s.

8.1.3.4. cty

As credential type is defined within the payload using the **sid** claim, it satisfies the ETSI requirement.

8.1.4. CCSP design rationale

Aim of the compact credentials is to render them as QR codes, hence we must avoid information duplication whenever possible. The definition of the **kid** and issuance date **di** allow us to both meet the business requirements and to avoid duplication of claims. This way we achieve that the signature is as small as possible, yet compliant with the ETSI requirements.

8.1.5. References

- ETSI 119 182-1 v1.2.1

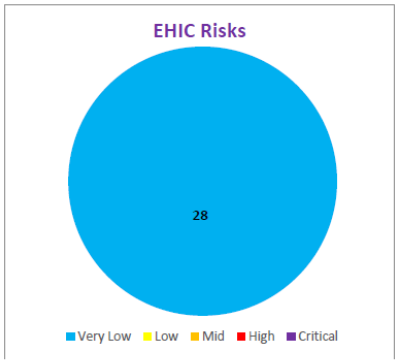
9. ANNEX II – QR CODE SECURITY ASSESSMENT

We refer to the public security study conducted by DG DIGIT on the use of QR codes, which can be found as a dedicated annex III to the note XXX/YYY. This study provides an in-depth threat assessment analysis on the security of QR codes in general and a dedicated section on the EHIC use case.

Met opmerkingen [GDM3]: Please add note number

The conclusion of the relevant section on the **'EHIC use case'** is that no risks rise above the 'Very Low' level. In fact, since the QR code comprises only the business data that is already visible on the PRC document itself, as well as it is in the form of a signed JWT for which only the issuer holds the private key, the risk of QR code fraud is very low.

After applying the security mechanisms, the Risk Profile is as follows:



The risk repartition shows that after applying security mechanisms, no risks rise above the "Very Low" level.
The most common Attack Paths are targeting the payload of the QR code. This is either by leveraging general flaws in the parsing of the encoded data [AP6], or by targeting potential vulnerabilities in the zlib implementation [EHIC-AP11].
The possible damage scenarios found can affect the Confidentiality of the private information on the QR code (such as 'personal identification number') [EHIC-D1], the Integrity of the accreditation for health care [EHIC-D2] and its Availability.
It is important to highlight that for those risks to materialise, very favourable conditions must be met, such as vulnerabilities in the QR code Reader's implementation and the ability to exploit it with a reduced payload size limited by the QR code.