Finnish National Vaccination Certificate

Verification process and technical specification

Version 1.2

26.5.2021

Kela

Kela
Kanta-palvelut
PL 450, 00056 Kela

FPA
Kanta tjänsterma
PB 450, 00056 FPA

Kanta 26.5.2021
www.kanta.fi
kanta@kanta.fi

Version History

Version	Change	User	Date
0.5	Initial document	Kela	30.4.2021
0.8	Json and production certificate information updated	Kela	12.5.2021
1.0	Python examples added	Kela	18.5.2021
1.1	Minor typo fixes	Kela	19.5.2021
1.2	Prodiction Certificate information added Test Certificate information removed	Kela	26.5.2021

Table of Contents

1 Finnish National Vaccination Certificate

The purpose of this specification is to provide necessary information to read, interpret and verify the issued **Finnish National Vaccination Certificates**.

Finnish National Vaccination Certificate contains signed machine-readable data as Aztec code. The data contains person identification information, details of the COVID-19 Vaccination and necessary metadata to identify a valid certificate.

- Person identification information
  - Family name
  - Given name(s) as list
  - Date of Birth
- Information of the vaccine given
  - Manufacturer of the vaccine
  - Administering Centre name where the vaccine was given
  - Lot number of the vaccine
  - Date when the vaccine was given
  - The Number of the vaccine given instance
  - The number of vaccinations to complete the series
  - Name and coding of the disease targeted with the vaccination (ICD-10)
  - Name and coding of the vaccine product given
- Finnish National Vaccination Certificate metadata

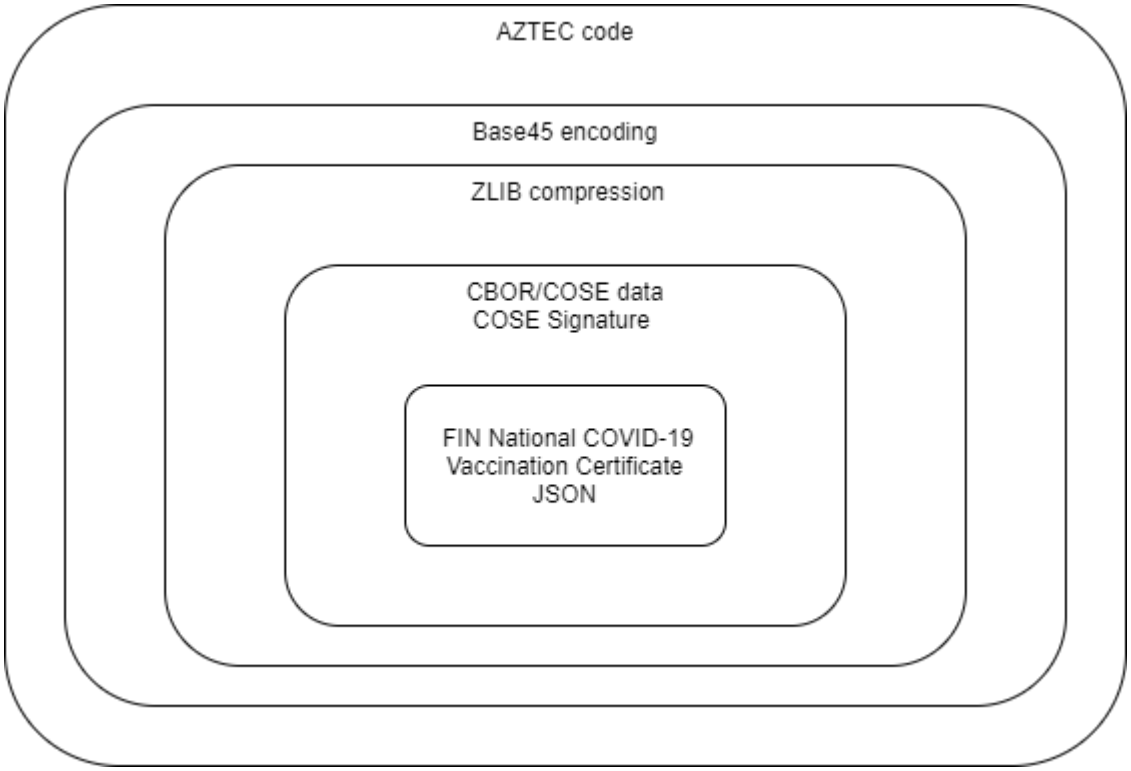
- Issuer of the Certificate
- Validity period of the Finnish National Vaccination Certificate as start end dates
- Finnish National Vaccination Certificate identifier to identify issued Certificates uniquely

The person identification information is needed to enable the matching of the identification of the holder to the information presented in the Vaccination Certificate.

The validity proof is done by signing the machine-readable data with cryptographic signature (ECDSA). The signature of the **Finnish National Vaccination Certificate** uses a public key method to verify. These public keys are made available to the verifiers.

The organization technically issuing **Finnish National Vaccination Certificate** is the Social Insurance Institution of Finland, Kela. The Vaccination information is provided by the Heath care unit administering the vaccination.

2 Validating process of the Finnish National Vaccination Certificate



1 Decode process chain.

The process chain to decode and verify Finnish National Vaccination Certificate.

2.1 Read the Aztec code

The Finnish National Vaccination Certificate uses Aztec code as machine-readable data delivery mechanism. Read and store the Aztec code data. The data is in Base45-encoded alphanumeric format.

2-1 Code example: Read Aztec code

```
#!/usr/bin/env python3
import sys
import zxing
# Read AZTEC code from .png file
def AztecDecode(filename: str) -> str:
    # Read AZTEC code from .png file
    reader = zxing.BarCodeReader()
    barcode = reader.decode(filename, try_harder = True, possible_formats="AZTEC")
    return barcode.parsed

```

```
def main():
    filename = sys.argv[1]
    base45_msg = AztecDecode(filename)
    print(base45_msg)

if __name__ == '__main__':
    main()

Output Base45 data string:
NCFM%G29OTK4+S2XIQ%34Z34L0BEY8KPST2M-KM …
```

2.2 Check the content data format and Decompress the zlib data

The Base45 data contains zlib-compressed data. The zlib data can be identified with magic bytes. The magic bytes can be checked from the first 3 characters of the Base45 encoded text (Base45 encoded is NCF) or after Base45 decode the first 2 bytes of the binary file are hex(78da). Decompress the binary zlib payload to obtain CBOR/COSE binary data.

2-2 Code example: Decode Base45 and zlib decompress

```
#!/usr/bin/env python3
import base45
import sys
import binascii
import zlib

# Decode Base45 data to binary
def decode_data(data: bytes, encoding: str) -> bytes:
    if encoding == "base45":
        return base45.b45decode(data)

# Decode and Decompress data to binary
def decode_and_decompress(data: bytes, encoding: str = "base45") -> bytes:
    # base45 to binary data
    decoded_data = decode_data(data, encoding)
    # decompress binary data
    # check if the data has zlib magic header 0x78da
    if  binascii.hexlify(decoded_data[:2]).decode() == "78da":
        uncompressed_data = zlib.decompress(decoded_data)
    else:
        raise ValueError("NO valild data found")
    return uncompressed_data
```

```
23
24def main():
25    base45_msg = sys.argv[1]
26    uncompressed_data = decode_and_decompress(bytes(base45_msg))
27    print(binascii.hexlify(uncompressed_data).decode())
28
29if __name__ == '__main__':
30    main()
31
32Output COSE message as hex encoded data:
33d28455a2045065333664666436613564316266363563012 6a059044 …
```

2.3 CBOR/COSE

The CBOR/COSE uses cose sign1 structure. The Finnish National Vaccination Certificate uses:

- Protected header
    - Contains alg and kid values
    - alg is CBOR object value -7 (ES256). The algorithm is Elliptic Curve Digital Signature Algorithm (ECDSA) NIST P-256 (secp256r1) in combination of the SHA–256 hash algorithm.
    - kid identifies the signing key. The key identifier (kid) is the first 8 bytes from the SHA-256 fingerprint of the public certificate encoded in DER (raw) hex representation format saved as UTF-8 bytes. Several public keys may be used in parallel by an Issuer when performing key rollovers. The Key Identifier itself not a security-critical field.

2-3 Code example: Key Identifier

```
1Key Identifier
2UTF-8 "aa0bbc6ae3464510"
3bstr "0110 0101 0011 0011 0011 0110 0110 0100 0110 0110 0110 0100 0011 0110 …"
4utf-8|hex 61=a |hex 61=a |hex 30=0 |hex 62=b |hex 62=b |hex 33=c |hex 36=6 |…
5
```

- Payload
    - Contains cbor serialized UTF-8 encoded Json data
- Signature
    - The digital signature of the protected COSE header and payload data

2-4 Code example: Decode CBOR/COSE and verify signature

```
#!/usr/bin/env python3
import sys
import binascii
import json

from cryptography import x509
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.backends import default_backend

import cbor2
import cose.headers
from cose.keys.cosekey import CoseKey
from cose.keys.ec2 import EC2
from cose.messages import CoseMessage
from cose.messages.sign1message import Sign1Message

from typing import Dict, List, Optional

from cryptojwt.jwk.ec import ECKey
from cryptojwt.jwk.x509 import import_public_key_from_cert_file
from cryptojwt.utils import b64d

# Return Cbor data as string (json)
def cose_message_data(cose_msg: Sign1Message) -> str:
    cose_payload = cbor2.loads(cose_msg.payload)
    return cose_payload

# Verify the signature as valid/unvalid
def verify_signature(uncompressed_data: bytes, sigature_certificate):
    cose_msg: Sign1Message = CoseMessage.decode(uncompressed_data)
```

```
    protected_header = cose_msg.phdr
    verify_status = False
    vaccination_json = ""
    kid = cose_msg.phdr.get(cose.headers.KID)
    alg = cose_msg.phdr.get(cose.headers.Algorithm)
    key = read_cosekey(sigature_certificate)
    if key.kid == kid:
        cose_msg.key = key
        if cose_msg.verify_signature():
            verified_key = key
            verified_status = cose_msg.verify_signature()
            vaccination_json = cose_message_data(cose_msg)
        else:
            raise ValueError("Signature not verified")
    else:
        raise ValueError("No valid keys found")
    return vaccination_json, verified_key, verified_status

# Create CoseKey from JWK
def cosekey_from_jwk_dict(jwk_dict: Dict) -> CoseKey:
    #Read key and return CoseKey
    if jwk_dict["kty"] != "EC":
        raise ValueError("Only EC keys supported")
    if jwk_dict["crv"] != "P-256":
        raise ValueError("Only P-256 supported")
    else:
        key = EC2(
            crv=cose.curves.P256,
            x=b64d(jwk_dict["x"].encode()),
            y=b64d(jwk_dict["y"].encode()),
        )
        key.key_ops = [cose.keys.keyops.VerifyOp]
    if "kid" in jwk_dict:
        key.kid = bytes(jwk_dict["kid"],"UTF-8")
    return key

# Create JWK and valculate KID from Public Signing Certificate
def read_cosekey(cert_file: str) -> CoseKey:
    # Read certificate, calculate kid and return EC CoseKey
    if cert_file.endswith(".pem"):
        with open(cert_file, 'rb') as f:
            cert_data = f.read()
            # Calculate Hash from the DER format of the Certificate
            cert = x509.load_pem_x509_certificate(cert_data, default_backend())
```

```
            keyidentifier = cert.fingerprint(hashes.SHA256())
            f.close()
        key = import_public_key_from_cert_file(cert_file)

        jwk = ECKey()
        jwk.load_key(key)
        # Use first 8 bytes of the hash as Key Identifier (Hex as UTF-8)
        jwk.kid = binascii.hexlify(keyidentifier[:8]).decode()
        jwk_dict = jwk.serialize(private=False)
    else:
        raise ValueError("Unknown key format. Use .pem keyfile")
    return cosekey_from_jwk_dict(jwk_dict)

def main():
    uncompressed_data = sys.argv[1]
    certificate_file = sys.argv[2]
    vaccination_json, verify_key, verify_status  = verify_signature(uncompressed_data,
certificate_file)
# The kid of the Certificate used in verification
    print(" verify_key: ",  verify_key.kid.decode("UTF-8"))
# The status of verification of the data signature (True/False)
    print(" verify_status: ",verify_status)
    print(json.dumps(vaccination_json,indent=4,ensure_ascii=False))

if __name__ == '__main__':
    main()

Output messages:
verify_key: aa0bbc6ae3464510
verify_status: True
{
    "contained": [
        {
…
```

2.4 Check validity of the public certificate

To select the right public certificate, match the Key Identification (kid) to the first 8 bytes of the SHA-256 fingerprint of the public certificate encoded in DER (raw) hex representation format. The data is saved as UTF-8 bytes.

The openssl command can be used to get the sha256 hash of the der format of the pem certificate:

- openssl x509 -in <pem filename> -outform der | openssl dgst -sha256
- The production certificate sha256 fingerprint is: aa0bbc6ae3464510f533fc8ee753e885098aa5e909684cae29573db134542294

To validate the public certificate check at least the following:

- Issuer:
  - The prodiction certificate Issuer is:
    - Issuer: C = FI, O = Vaestorekisterikeskus CA, OU = Sosiaali- ja terveydenhuollon testipalveluvarmenteet, CN = VRK CA for Social Welfare and Health Care Service Providers
- Validity:
  - Check the validity timeframe of the public certificate
    - Not Before:
    - Not After:
- Subject:
  - In the production public certificate subject is:
    - Subject: C = FI, ST = Finland, L = Helsinki, O = Kansanelakelaitos, OU = Kanta, serialNumber = 1.2.246.556.12.21.1, CN = Todistuspalvelu
  - Revocation status:

- Check the revocation status of the Certificate

- OCSP or CRL can be used

If the public certificate does not pass the validity tests, the Finnish National Vaccination Certificate must be considered invalid.

2.5 Check validity of the FIN National Vaccination Certificate

For checking the validity of the **National Vaccination Certificate json data**, the needed metadata is in the json "extension:" structure. The json structure is shown in the **Example Json data**.

The Issuer of the National Vaccination Certificate holds the issuing entity.

- `{ "valueCodeableConcept" : { "coding" : [{ "code" : "FI", "system" : "urn:iso:std:iso:3166", "display" : "Finland" }]}}`

- `{ "url" : "CertificateIssuer", "valueString" : "Kela"}`

The validity period of the National Vaccination Certificate contains start and end times are in Date type ISO 8601 - date part only format. The verifier must check that the current date is in the range (start <= current date >= end). The technical issuing period is 1 year.

- `{ "url" : "CertificateValidationTime", "valuePeriod" : {"start": "2021-05-25", "end" : "2022-05-25" }`

If the above issuing entity do not match or the current time is outside of the valid time frame, Vaccination Certificate must be considered invalid.

3 Example json data

Json data

```
{
    "contained": [
# Personal Identification Information
        {
            "id": "patient1",
            "resourceType": "Patient",
# Family and given name(s)

            "name": [
                {
                    "family": "Jääskeläinen",
                    "given": [
                        "Matti-Pekka",
                        "Sakari"
                    ]
                }
            ],
            "birthDate": "1999-05-18"
        }
    ],
    "patient": {
        "reference": "#patient1"
    },
# Information of the vaccine given
# Vaccine lot number to identify a specific patch of vaccination given
    "lotNumber": "78901",
    "protocolApplied": [
        {
            "targetDisease": [
                {
                    "coding": [
                        {
# ICD-10 code, OID and display name for the target disease of the vaccination
                            "code": "U07.1",
                            "system": "1.2.246.537.6.1.1999",
                            "display": "COVID-19-virusinfektio, varmistettu"
                        }
                    ]
                }
            ],
# Number of Vaccination doses given and the total number of Vaccinations in the series
```

```
            "doseNumberPositiveInt": 1,
            "seriesDosesPositiveInt": 2
        }
    ],
# Vaccination name
    "vaccineCode": {
        "text": "ASTRAZENECA COVID-19 VACCINE ",
        "coding": [
# ATC code, OID and displayname for the Vaccination
            {
                "code": "J07BX03",
                "system": "1.2.246.537.6.32.2007",
                "display": "COVID-19-rokotteet"
            }
        ]
    },
# Administring date of the Vaccination
    "occurrenceDateTime": "2021-03-11",
    "status": "completed",
# The reason for the operation (Vaccination)
    "resourceType": "Immunization",
    "extension": [
        {
            "url": "http://todistuspalvelu.kanta.fi/StructureDefinition/ext",
            "extension": [
# The manufacturer of the Vaccination
                {
                    "valueString": "AstraZeneca AB",
                    "url": "VaccineMarketingAuthorizationHolder"
                },
# The Organization and unique OID that administered the Vaccination
                {
                    "valueCodeableConcept": {
                        "coding": [
                            {
                                "code": "1.2.246.99.9999999.90.1",
                                "display": "Terveyskeskus testausta varten 1"
                            }
                        ]
                    },
                    "url": "AdministeringCentre"
                },
# Finnish National Vaccination Certificate metadata
                {
```

```
                    "valueCodeableConcept": {
                        "coding": [
# Entity Country Issuing the Finnish COVID-19 Vaccination Certificate
                            {
                                "code": "FI",
                                "system": "urn:iso:std:iso:3166",
                                "display": "Finland"
                            }
                        ]
                    },
                    "url": "CountryOfVaccination"
                },
# Entity Technically Issuing the Finnish COVID-19 Vaccination Certificate
                {
                    "valueString": "Kela",
                    "url": "CertificateIssuer"
                },
# Technical Validity Period of the Finnish COVID-19 Vaccination Certificate
                {
                    "url": "CertificateValidationTime",
                    "valuePeriod": {
                        "start": "2021-05-18",
                        "end": "2022-05-18"
                    }
                }
            ]
        }
    ],
# Unique identifier of the Finnish COVID-19 Vaccination Certificate
    "identifier": [
        {
            "value": "01/FI/9F2FTYRMRDHTTGAU7UC7XWJQD#K"
        }
    ]
}
```

4 Technology stack

4.1 Aztec code

Finnish National Vaccination Certificate uses Aztec code as machine-readable code to enable verification of the Vaccination Certificate data.

- ISO - ISO/IEC 24778:2008 - Information technology — Automatic identification and data capture techniques — Aztec Code bar code symbology specification
- ZXing – opensource.google

4.2 Base45

The data is encoded with Base45 to provide alphanumeric data to Aztec code.

- draft-faltstrom-base45-04 - The Base45 Data Encoding (ietf.org)

4.3 Zlib compression

The data size is minimized with zlib compression.

- RFC 1950 - ZLIB Compressed Data Format Specification version 3.3 (ietf.org)
- zlib - Wikipedia
- List of file signatures - Wikipedia search “zlib” for the “magic bytes”

4.4 CBOR/COSE

The Concise Binary Object Representation (CBOR) is used to encode and serialize data to extremely small code size.

- RFC 8949 - Concise Binary Object Representation (CBOR) (ietf.org)
- RFC 8152 - CBOR Object Signing and Encryption (COSE) (ietf.org)

4.5 JSON

JavaScript Object Notation (JSON).

- RFC 8259 - The JavaScript Object Notation (JSON) Data Interchange Format (ietf.org)

4.6 FHIR

The Finnish National Vaccination Certificate data is in modified Fast Healthcare Interoperability Resources (FHIR) JSON format.

- Index - FHIR v4.0.1 (hl7.org)

4.7 X509v3 Certificate

To verify the validity of Finnish National Vaccination Certificate data, a public certificate to verify the signature is made available from https://www.kanta.fi website.

- RFC 5280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile (ietf.org)

4.8 Elliptic Curve Digital Signature Algorithm

The ECDSA signature with NIST P256 (secp256r1) is used to sign the Finnish National Vaccination Certificate.