

# UZI Certificate Credential

## Abstract

This proposal describes a method for issuing Verifiable Credentials by leveraging the existing chains of trust provided by X.509 certificates. For this it uses the `did:x509` did-method.

## Status of this document

This document has the status draft.

## Introduction

With the introduction of the Self Sovereign Identity approach and techniques, high trust application can leverage the possibility of combining several identity claims. This allows for more flexible and fine grained authorization rules and thus data protection. These techniques however are quite new and we find ourselves in a typical chicken-egg situation: personal or company wallets can be the solution for SSI authentication needs, but without available credentials, these wallets have no real value. Issuers are reluctant with issuing credentials until wallets are a tried and proven technology. How can we overcome this catch-22?

Digital trust is not new. There are already a lot of parties who act as a QTSP and provide trust attributes in the form of X.509 certificates. In the Netherlands for the care domain this is done by the CIBG who issues UZI certificates for individuals and systems.

This specification introduces a method of bridging this the gap by issuing UZI Verifiable Credentials based on the [did:x509 method](#). With this we leverage the existing trust in the certificate issuing proces and translating this into a Verifiable Credential.

## Chain of trust

The goal of this method is creating a verifiable chain of trust from the UZI Verifiable Credential back to the trusted UZI Certificate Authority.

Often a did identifies an person or organisation this is not limited to those. Everything can be identified by a did. With the `did:x509 method`, the certificate subject is the issuer of the credential. It can sign the credential using its private key. A verifier can resolve and verify the certificate by parsing the certificate chain, checking the validity and checking the values given in the did string. If this checks out, the verified knows that there exists a valid certificate, issued by a specified CA

which contains certain values.

Example did:

```
did:x509:0:sha512:3oeULL9TgHNiKTamKoYdWnJXuxV_5ICu0sA8SGYUwerek-  
xY4Zgr5vaFuMwMPkAomtHOnHQRk5oVYpXcFgBLOg::san:otherName:2.16.528.1.1007.99.2110-1-88899801-S-  
88899901-00.000-11122201
```

The above did specifies that the certificate should be issued by a CA with the fingerprint

`3oeULL9TgHNiKTamKoYdWnJXuxV_5ICu0sA8SGYUwerek-xY4Zgr5vaFuMwMPkAomtHOnHQRk5oVYpXcFgBLOg` and the certificate should contain a field `san:otherName` with the value `2.16.528.1.1007.99.2110-1-88899801-S-88899901-00.000-11122201`.

When resolving the credential, the complete chain should be provided. The resolve operation can be interpreted as follows: resolve a DID document for a x509 certificate where the issuer ca can be identified a fingerprint and contains certain fields with certain values.

## UZI Server certificates

The UZI production chain is described on the [zorgcsp website](#). The UZI test chain is described [here](#)

### Issuer

Server certificates are issued by the `UZI-register Private Server CA G1` CA intermediate.

The `sha256` fingerprint of these intermediate CA's can be generated by making a sha256sum of the DER encoded files. WARNING: don't trust the values on this page, they are for

Test: `1b0961059b841654875d24545d0b93b37fd8a50c406a10a89702498f7e544b50`

Production: `bdd860ef8e87e2b2c7ebb34dd6e9e1771a3a3c5dec850ba7080e3e2904dbd897`

### Claims

The goal is to create a credential to uniquely identify a care organisation. We want to use the following relevant fields from the certificate:

Claim	path	oid
Organisation name	<code>subject:O</code>	<code>2.5.4.10</code>
City	<code>subject:L</code>	<code>2.5.4.7</code>
Identifiers	<code>san:otherName</code>	<code>2.5.5.5</code>

The identifiers field is unfortunately a bit cumbersome since it contains a concatenated string of a lot of relevant identifiers in the following form:

<OID CA>-<version-nr>-<UZI-nr>-<pastype>-<Subscriber-nr>-<role>-<AGB-code>

Example of a `san:otherName`:

2.16.528.1.1007.99.2110-1-900030787-S-90000380-00.000-11223344

## Resolving a DID document

The DID document can be resolved based on the DID and a certificate chain.

1. Check if the CA-fingerprint of the DID matches with the root or one of the intermediate certificates.
2. Validate the chain such as is common practice: validity, cryptography, hierarchy, revocation status etc.
3. Check if the leaf certificate contains all policy keys and values from the DID
4. Create the DID document with the `id` field set to the DID and a `assertionMethod` containing the correctly encoded public key from the leaf certificate.

Example DID document:

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/jws-2020/v1"
  ],
  "id":
    "did:x509:0:sha256:1b0961059b841654875d24545d0b93b37fd8a50c406a10a89702498f7e544b50::subject:O:De%20Regenboog:L:Hengelo::san:othername:2.16.528.1.1007.99.2110-1-900030787-S-90000380-00.000-11223344",
  "verificationMethod": [{
    "id":
      "did:x509:0:sha256:1b0961059b841654875d24545d0b93b37fd8a50c406a10a89702498f7e544b50::subject:O:De%20Regenboog:L:Hengelo::san:othername:2.16.528.1.1007.99.2110-1-900030787-S-90000380-00.000-11223344#0",
    "type": "JsonWebKey2020",
    "controller":
      "did:x509:0:sha256:1b0961059b841654875d24545d0b93b37fd8a50c406a10a89702498f7e544b50::subject:O:De%20Regenboog:L:Hengelo::san:othername:2.16.528.1.1007.99.2110-1-900030787-S-90000380-00.000-11223344",
    "publicKeyJwk": {
      // JSON Web Key
    }
  ]
}
```

```
}]
}

}
```

## Creating a Verifiable Credential

The credential can only contain fields which are also part of the issuer did. The names must match.

Example credential:

```
{
  "issuer":
    "did:x509:0:sha256:1b0961059b841654875d24545d0b93b37fd8a50c406a10a89702498f7e544b50::subject:O:De%20Regenboog:L:Hengelo::san:othername:2.16.528.1.1007.99.2110-1-900030787-S-90000380-00.000-11223344",
  "credentialSubject": {
    "subject:O": "De Regenboog",
    "subject:L": "Hengelo",
    "san:otherName": "2.16.528.1.1007.99.2110-1-900030787-S-90000380-00.000-11223344"
  }
}
```

## Credential format

In order for the credential to contain the certificate chain, we need it to be in the `jwt_vc` format. The header of the `JWT` must contain the `x5c` field with the complete chain.

## Verifying a credential

0. Extract the certificate chain from the credential proof
1. Resolve the issuer did document based on the certificate chain
2. Resolve the public key from the DID document
3. Check the credential signature
4. Check the credential validity
5. Check if the fields and values of the credentialSubject match the issuer did
6. Check if the issuer fingerprint matches the list of trusted issuers

---

Revision #5

Created 11 October 2024 13:11:46 by Steven van der Vegt

Updated 5 November 2024 09:06:42 by Steven van der Vegt