X509Credential with UZI server certificates

Care organizations authenticate with an X509Credential, created from their UZI server certificate. This page explains how the certificate ends up in the credential, and what a use case's Discovery Service and policy definitions need to check.

The X509Credential is defined in the credential catalog (normative). For `did:x509` and the background, see RFC023. For how to create one, see "Get an X509Credential from a UZI certificate" in Implementing a Nuts v2 Use Case.

Why X509Credential

- **Reuses existing checks:** CIBG already checks every care organization and its URA before it issues a UZI server certificate. No new issuer is needed.
- **The key holder speaks for the care organization:** the credential is signed with the UZI certificate's private key. Ideally the care organization holds that key; in practice the vendor often manages it.
- **Links the DID to the URA:** the credential is issued to the care organization's DID (`did:web`).
- **Limit:** only organizations with a URA and a UZI server certificate can get one.

> **HealthcareProviderCredential:** the HealthcareProviderCredential (credential catalog, draft) is seen as the next version of the X509Credential. It is built from the same UZI server certificate through `did:x509`, but contains the URA as a typed identifier instead of the raw certificate fields. This page covers only the X509Credential.

Which CA to trust

UZI server certificates are issued by the UZI register (CIBG), under the PKIoverheid hierarchy. The X509Credential's issuer is a `did:x509` DID, which contains the fingerprint (`ca-fingerprint`) of one CA in the certificate's chain. A definition trusts a CA by listing its fingerprint.

The CA that issues the UZI server certificates is called the **UZI CA** below. It is an intermediate CA itself: the lowest CA in the chain, directly above the UZI server certificate (marked ◄ in the trees).

```
Production G1
Staat der Nederlanden Private Root CA - G1
└── Staat der Nederlanden Private Services CA - G1
    ├── UZI-register Private Server CA G1              ◄ ca-fingerprint
    │   └── UZI server certificate
    └── KPN, DigiCert, QuoVadis, ... (other providers, no URA)

Production G4 (from 12 November 2026)
Staat der Nederlanden - G4 Root Priv G-TLS - 2024
└── Staat der Nederlanden - G4 Intm Priv G-TLS SYS - 2024
    ├── UZI Server - G4 PKIo Priv G-TLS SYS - 2025      ◄ ca-fingerprint
    │   └── UZI server certificate
    └── KPN, DigiCert, Digidentity, ... (other providers, no URA)

Test G1
TEST Zorg CSP Private Root CA G1
└── TEST Zorg CSP Level 2 Private Services CA G1
    └── TEST UZI-register Private Server CA G1          ◄ ca-fingerprint
        └── UZI server certificate

Acceptance G4
ACCEPTATIE Zorg CSP - G4 Root Priv G-TLS - 2024
└── ACCEPTATIE Zorg CSP - G4 Intm Priv G-TLS SYS - 2024
    └── ACCEPTATIE UZI Server - G4 Priv G-TLS SYS - 2025 ◄ ca-fingerprint
        └── UZI server certificate
```

UZI CA	Environment	`ca-fingerprint` (SHA-256, base64url)
UZI-register Private Server CA G1	Production	`vdhg746H4rLH67NN1unhdxo6PF3shQunCA4-KQTb2Jc`
UZI Server - G4 PKIo Priv G-TLS SYS - 2025	Production, from 12 November 2026	`sDTL__yv54TtrvSf7601TtvHgj2NZSLT2m-Okf8oqkI`
TEST UZI-register Private Server CA G1	Test	`GwlhBZuEFlSHXSRUXQuTs3_YpQxAahColwJJj35US1A`
ACCEPTATIE UZI Server - G4 Priv G-TLS SYS - 2025	Test, acceptance	`svKzNxhS07V1dKkekWOZV0d6Ao7_qvjqXegWB32YjZE`

Rules for definitions

A definition **should** trust exactly the UZI CAs for its environment:

- **Production:** UZI-register Private Server CA G1 and UZI Server - G4 PKIo Priv G-TLS SYS - 2025. CIBG starts issuing G4 certificates on 12 November 2026; G1 certificates stop working on 12 November 2028. Until then, trust both.
- **Test and acceptance:** TEST UZI-register Private Server CA G1 and ACCEPTATIE UZI Server - G4 Priv G-TLS SYS - 2025.

A definition **should not** trust:

- a CA higher in the chain than the UZI CA (the "Staat der Nederlanden" or "Zorg CSP" root and intermediate CAs): those also issue certificates through other providers (e.g. KPN, DigiCert), which don't contain a URA;
- a test or acceptance CA in a production definition: test certificates are easy to get.

Deviate only with a reason, and write it down in the use case's IG.

From certificate to credential

These certificate fields end up in the X509Credential:

Certificate field	Credential field	Contains
`subject.O`	`credentialSubject.subject.O`	Name of the care organization
`subject.L`	`credentialSubject.subject.L`	City
`san.otherName`	`credentialSubject.san.otherName`	UZI number, URA and more, see below

The URA in `san.otherName`

`san.otherName` is one string with seven parts, separated by `-`:

```
<OID CA>-<version>-<UZI number>-<pass type>-<URA>-<role>-<AGB code>
2.16.528.1.1007.99.2110-1-900025039-S-90000382-00.000-00000000
```

Part	Example	Meaning
OID CA	`2.16.528.1.1007.99.2110`	The CIBG
Version	`1`	Version of this format
UZI number	`900025039`	Number of the certificate holder, same as `subject.serialNumber`
Pass type	`S`	Always `S` (server)
URA	`90000382`	The care organization's URA ("abonneenummer")
Role	`00.000`	Always `00.000`

Part	Example	Meaning
AGB code	`00000000`	The care organization's AGB code

To get the URA in a definition, use a pattern with a capture group on this field. The node then returns only the captured part, e.g. in search results and token introspection:

```
{
  "id": "organization_ura",
  "path": ["$.credentialSubject.san.otherName"],
  "filter": { "type": "string", "pattern": "^[0-9.]+-\\d+-\\d+-S-(\\d+)-00\\.000-\\d+$" }
}
```

Appendix: examples

Fictional care organization "De Regenboog" in Hengelo, URA `90000380`, with a UZI server certificate from the TEST UZI CA.

did:x509 DID

```
did:x509:0:sha256:GwlhBZuEFlSHXSRUXQuTs3_YpQxAahColwJJj35US1A::subject:O:De%20Regenboog:L:Hengelo::san:otherName:2.16.528.1.1007.99.2110-1-900030787-S-90000380-00.000-11223344
```

Part	Meaning
`sha256:GwlhBZuE…`	`ca-fingerprint`: SHA-256 hash of the DER-encoded TEST UZI-register Private Server CA G1 certificate, base64url-encoded
`subject:O:De%20Regenboog:L:Hengelo`	Certificate fields `subject.O` (OID 2.5.4.10) and `subject.L` (OID 2.5.4.7), URL-encoded
`san:otherName:…`	Certificate field `san.otherName` (OID 2.5.5.5), with the URA `90000380`

Resolved DID document

The node builds the DID document from the certificate chain in the credential's JWT header (`x5c`):

```
{
  "@context": ["https://www.w3.org/ns/did/v1"],
  "id":
"did:x509:0:sha256:GwlhBZuEFlSHXSRUXQuTs3_YpQxAahColwJJj35US1A::subject:O:De%20Regenboog:L:Hengelo::san:otherName:2.16.528.1.1007.99.2110-1-900030787-S-90000380-00.000-11223344",
  "controller":
["did:x509:0:sha256:GwlhBZuEFlSHXSRUXQuTs3_YpQxAahColwJJj35US1A::subject:O:De%20Regenboog:L:He
```

```
ngelo::san:otherName:2.16.528.1.1007.99.2110-1-900030787-S-90000380-00.000-11223344"],
  "verificationMethod": [
    {
      "id":
"did:x509:0:sha256:GwlhBZuEFlSHXSRUXQuTs3_YpQxAahColwJJj35US1A::subject:O:De%20Regenboog:L:Hen
gelo::san:otherName:2.16.528.1.1007.99.2110-1-900030787-S-90000380-00.000-11223344#0",
      "type": "JsonWebKey2020",
      "controller":
"did:x509:0:sha256:GwlhBZuEFlSHXSRUXQuTs3_YpQxAahColwJJj35US1A::subject:O:De%20Regenboog:L:Hen
gelo::san:otherName:2.16.528.1.1007.99.2110-1-900030787-S-90000380-00.000-11223344",
      "publicKeyJwk": { "kty": "RSA", "n": "<modulus>", "e": "AQAB" }
    }
  ],
  "assertionMethod":
["did:x509:0:sha256:GwlhBZuEFlSHXSRUXQuTs3_YpQxAahColwJJj35US1A::subject:O:De%20Regenboog:L:He
ngelo::san:otherName:2.16.528.1.1007.99.2110-1-900030787-S-90000380-00.000-11223344#0"]
}
```


X509Credential (JWT payload)


```
{
  "iss":
"did:x509:0:sha256:GwlhBZuEFlSHXSRUXQuTs3_YpQxAahColwJJj35US1A::subject:O:De%20Regenboog:L:Hen
gelo::san:otherName:2.16.528.1.1007.99.2110-1-900030787-S-90000380-00.000-11223344",
  "sub": "did:web:nuts.vendor.example:iam:deregenboog",
  "vc": {
    "@context": ["https://www.w3.org/2018/credentials/v1", "https://nuts.nl/credentials/v1"],
    "type": ["VerifiableCredential", "X509Credential"],
    "credentialSubject": {
      "id": "did:web:nuts.vendor.example:iam:deregenboog",
      "subject": { "O": "De Regenboog", "L": "Hengelo" },
      "san": { "otherName": "2.16.528.1.1007.99.2110-1-900030787-S-90000380-00.000-11223344" }
    }
  }
}
```


The JWT header contains the full certificate chain in `x5c`.

Sources for the CA certificates: production and acceptance on zorgcsp.nl.

Revision #9
Created 2025-03-04 10:13:06 UTC by Roland Groen

Updated 2026-10-28 10:30:22 UTC by Rein Krul