Set up a care organization

Do this once per care organization, on the vendor's Nuts node: create a subject, get an X509Credential from the organization's UZI server certificate, and load it into the wallet.

- Create a subject
- Get an X509Credential from a UZI certificate
- Load the credential into the wallet
- Obtaining credentials

Create a subject

Each care organization gets its own subject on the vendor's Nuts node. A subject is the name you use for the care organization in all calls to the Nuts node's internal API. Behind a subject, the node creates one or more DIDs (decentralized identifiers) and their keys.

Do this once per care organization, per environment.

Before you start

- The Nuts node is deployed (see chapter "Deploy").
- did:web is enabled by default. We recommend setting `didmethods` to `web`, so the node doesn't also create a did:nuts DID, which spec v2 use cases don't need.

Decide: generated or your own subject ID

You have to choose one of these:

Option	How	What you do
Let the Nuts node generate the subject ID	Leave out `subject` in the request	Store the subject ID from the response in your own system, with the care organization
Bring your own subject ID	Set `subject` in the request, e.g. to the ID your system already uses for the care organization	Nothing to store: you already know the ID

Keep in mind:

- The subject ID is public. It's part of your authorization server URL (`https://<your node>/oauth2/<subject ID>`), which is published on the Discovery Service and is in every access token. Don't use an ID you want to keep private.
- It may only contain letters, digits, `.`, `_` and `-`.

If you bring your own subject ID and it isn't random (e.g. a KVK number or an auto-incrementing database key), consider adding the environment at the end, e.g. `12345678-acc`. The same ID then can't exist in two environments, which reduces the chance of leaking data because environments got mixed up. The examples on this page use `careorg-a-acc`.

Using the API

```
POST <internal Nuts interface>/internal/vdr/v2/subject
Content-Type: application/json

{"subject": "careorg-a-acc"}
```

The response contains the subject ID and its DID documents:

```
{
  "subject": "careorg-a-acc",
  "documents": [
    {
      "id": "did:web:nuts.example.com:iam:657f064a-ebef-4f0f-aa87-88ed32db3142",
      ...
    }
  ]
}
```

Using Nuts Admin

1. Log in to Nuts Admin.
2. Click "Identities".
3. Click the plus sign on the right.
4. Enter the subject ID, e.g. `careorg-a-acc`.
5. Click "Create Identity".

Check that it worked

Other parties must be able to resolve the DID from the internet. Test that with a public DID resolver:

1. Take the DID from the `id` field in the response above, e.g. `did:web:nuts.example.com:iam:657f064a-ebef-4f0f-aa87-88ed32db3142`.
2. Paste it into the Universal Resolver at https://dev.uniresolver.io/ and click "Resolve".
3. It should return the DID document. If it fails, check that the node's public interface is reachable from the internet through your reverse proxy (see chapter "Deploy"). The resolver fetches the document from `https://nuts.example.com/iam/657f064a-ebef-4f0f-aa87-88ed32db3142/did.json`, so you can also open that URL from outside your network.

You need this DID in the next step, to issue the X509Credential. To look up a subject's DIDs later, use `GET <internal Nuts interface>/internal/vdr/v2/subject/careorg-a-acc`.

API reference: Verifiable Data Registry (v2)

Get an X509Credential from a UZI certificate

Many use cases identify care organizations by their URA number. The CIBG doesn't issue Verifiable Credentials with the URA yet, so you create one yourself from the care organization's UZI server certificate: an X509Credential.

The node uses the X509Credential to:

- register the care organization on the Discovery Service, so other parties can find it;
- prove the care organization's identity when it requests an access token from another party.

The X509Credential is:

- **issued by** the UZI certificate: the issuer is a did:x509 DID derived from the certificate, and the credential is signed with the certificate's private key;
- **issued to** the care organization's did:web DID, which belongs to its subject on your Nuts node. That subject holds the credential in its wallet.

It contains the UZI certificate, and the URA, name and city taken from it. It does **not** contain the private key.

For background, such as how the URA ends up in the credential and which certificate authority a use case trusts, see X509Credential with UZI server certificates. The credential's definition: X509Credential in the credential catalog.

Do this once per care organization, per environment.

Who does what

The UZI certificate and its private key belong to the care organization. Ideally, the private key never leaves the care organization:

Step	Who
Create the subject and give its did:web DID to the care organization	Vendor
Create the X509Credential with the UZI certificate and private key	Care organization
Send the X509Credential (a JWT, without the private key) to the vendor	Care organization

Step	Who
Load the X509Credential into the wallet	Vendor

In practice, the vendor often already manages the care organization's UZI certificate, e.g. for client TLS connections. The vendor then has the private key anyway, and can create the X509Credential itself. The steps below work for the care organization and the vendor alike.

Before you start

- The care organization has a subject, and you have its did:web DID from the vendor (see "Create a subject").
- The care organization's UZI server certificate from the CIBG:
  - the full certificate chain (root CA up to the certificate itself) as a PEM file;
  - the unencrypted private key of the certificate as a PEM file.
- A tool to create the credential. Choose one:
  - The `nutsfoundation/go-didx509-toolkit` Docker image (source), to create the credential by hand. The steps below use this tool.
  - The Java library uzi-did-x509-issuer-java, to create the credential from your EHR or another Java application, without Docker. See its README for usage.

Create the credential

Put the certificate chain (`certificate.pem`) and private key (`privatekey.pem`) in a directory called `certs`. From the directory that contains `certs`, run:

```
docker run --rm -v "$(pwd)/certs:/certs" \
  nutsfoundation/go-didx509-toolkit:1.2.0 \
  vc /certs/certificate.pem /certs/privatekey.pem \
  "<CA name>" \
  <did>
```

- `<did>` is the care organization's did:web DID, from "Create a subject".
- `<CA name>` is the full name (subject DN) of the CIBG certificate authority that issued the UZI server certificate. It depends on the type of certificate:

Certificate	CA (`CN` part of the name)
Production, G1	`UZI-register Private Server CA G1`
Production, G4 (from 12 November 2026)	`UZI Server - G4 PKIo Priv G-TLS SYS - 2025`
CIBG test, G1	`TEST UZI-register Private Server CA G1`
CIBG acceptance, G4	`ACCEPTATIE UZI Server - G4 Priv G-TLS SYS - 2025`

The easiest way to get the exact name: run the command with `"<CA name>"` as it is. The tool then fails with `ca-fingerprint certificate not found in chain` and lists the names of all certificates in your chain under `Candidates:`. Copy the one for the CA and run the command again.

Check https://hub.docker.com/r/nutsfoundation/go-didx509-toolkit/tags for the latest version.

The output is the X509Credential as a JWT (a long string starting with `eyJ`). Send it to the vendor, who loads it into the care organization's wallet (see "Load the credential into the wallet").

Non-production environments

For non-production environments, the CIBG issues test certificates whose name, URA and city don't match the real care organization. Where the use case allows it, you can instead create a fake UZI server certificate with a test CA:

1. Check out https://github.com/nuts-foundation/go-didx509-toolkit. You need Bash and `openssl`.
2. In the `test_ca` directory, run:

   ```
   ./issue-cert.sh <domain (CN)> <name (O)> <city (L)> <uzi> <ura> <agb>
   ```

   The domain, UZI and AGB values can be anything; use cases don't use them.
3. The certificate chain and private key are written to the `out` directory. Copy them to your `certs` directory.
4. Create the credential as above, but with `"CN=Fake UZI Root CA"` as the CA:

   ```
   docker run --rm -v "$(pwd)/certs:/certs" \
     nutsfoundation/go-didx509-toolkit:1.2.0 \
     vc /certs/<domain>-chain.pem /certs/<domain>.key \
     "CN=Fake UZI Root CA" \
     <did>
   ```

Which CAs an environment accepts is set in the use case's Discovery Service definition.

360-graden-clientbeeld

The X509Credential is the only credential a care organization needs to obtain. Accepted certificates per environment:

Environment	Accepted certificates

Development, test	Fake UZI certificates and CIBG test certificates (G1 and acceptance G4)
Acceptance	CIBG test certificates (G1 and acceptance G4) and real G1 certificates. Not fake ones, and not real G4 ones.
Production	Real CIBG certificates only (G1 and G4)

Lifecycle

The X509Credential is only valid as long as the UZI certificate it was created from. Create and load a new X509Credential:

- before the UZI certificate expires, with the renewed certificate;
- when the UZI certificate is replaced or revoked;
- when the care organization's details in the certificate change (name, city).

Load the new credential as described in "Load the credential into the wallet", then remove the old one (see "Renew credentials").

Load the credential into the wallet

Load the X509Credential into the care organization's wallet on the Nuts node. The node uses credentials from the wallet when it registers on the Discovery Service and when it requests access tokens.

The same steps work for any other credential you receive for the care organization.

Before you start

- The care organization has a subject (see "Create a subject").
- You have the X509Credential as a JWT (see "Get an X509Credential from a UZI certificate").

Using the API

```
POST <internal Nuts interface>/internal/vcr/v2/holder/careorg-a-acc/vc
Content-Type: application/json


"eyJ..."
```

- Replace `careorg-a-acc` with the subject ID.
- The body is the JWT as a JSON string: the JWT in double quotes.
- The node returns `204 No Content` on success. It returns `400 Bad Request` if the credential is invalid, for example when it was issued to a DID that doesn't belong to this subject, or when it has expired.

Using Nuts Admin

1. Log in to Nuts Admin.
2. Click "Identities".
3. Click the care organization.
4. Under "Credentials in Wallet", click "Upload".
5. Paste the credential (the JWT) in the input field.
6. Click "Upload Credential".

Check that it worked

List the credentials in the wallet:

```
GET <internal Nuts interface>/internal/vcr/v2/holder/careorg-a-acc/vc
```

The result should contain a credential with type `X509Credential`.

API reference: Verifiable Credential Registry (v2)

Obtaining credentials

Some use cases require credentials besides the X509Credential. Check the use case's Discovery Service and policy definitions to see which credentials you need. Credential types are defined in the credential catalog.

> **360-graden-clientbeeld**
>
> No other credentials are needed. You can skip this page.

A credential is issued in one of two ways, depending on the use case:

Issued by	When	What you do
A trusted third party	The use case names an issuer, e.g. VZVZ or Vektis	Receive the credential from the issuer and load it into the wallet
The care organization itself (self-issued)	The use case allows it, e.g. because there's no trusted issuer yet	Issue it with your Nuts node, then load it into the wallet

Self-issue a credential

The steps below use the HealthcareProviderRoleTypeCredential as example, from the Huisartsinzage use case.

The HealthcareProviderRoleTypeCredential states the type of care organization, e.g. `H1` (huisartsinstelling) or `X3` (verplegings- of verzorgingsinstelling). Huisartsinzage requires it. VZVZ might issue it in the future; until then, care organizations issue it themselves.

Before you start

- The care organization has a subject and a did:web DID (see "Create a subject").

Issue the credential

```
POST <internal Nuts interface>/internal/vcr/v2/issuer/vc
Content-Type: application/json

{
  "@context": "https://www.w3.org/2018/credentials/v1",
  "type": "HealthcareProviderRoleTypeCredential",
  "issuer": "did:web:nuts.example.com:iam:657f064a-ebef-4f0f-aa87-88ed32db3142",
```

```
  "credentialSubject": {
    "id": "did:web:nuts.example.com:iam:657f064a-ebef-4f0f-aa87-88ed32db3142",
    "roleCodeNL": "X3"
  },
  "expirationDate": "2027-10-27T00:00:00Z",
  "format": "jwt_vc"
}
```

- `issuer` and `credentialSubject.id` are both the care organization's DID: it issues the credential to itself.
- `format` must be `jwt_vc`. The default format (`ldp_vc`, JSON-LD) fails, because no JSON-LD context defines `roleCodeNL`.
- Set `expirationDate`. Without it, the node requires `withStatusList2021Revocation`.
- To make the credential revocable, add `"withStatusList2021Revocation": true`.

The response is the credential as a JWT, in double quotes.

Load the credential into the wallet

Issuing doesn't put the credential in the wallet. Load it as described in "Load the credential into the wallet", using the response from the step above as the request body.

Lifecycle

Issue and load a new credential before `expirationDate`, and when the care organization's type changes.

API reference: Verifiable Credential Registry (v2)