Authenticatie

Hoe stellen we de identiteit van een gebruiker / persoon vast binnen het Nuts netwerk?

- Proposal for EmployeeIdentity means
- UZI Certificate Credential

Proposal for EmployeeIdentity means

> This document is still in draft and is subject to change.

> Some parts are still WIP and both the English ad Dutch language is used.

In this document we propose a new *means* to identify a user. The identity is issued by the employer about the employee, hence the name `EmployeeIdentity`. The goal of this means is to reuse the information form the current session the user has within the application. This will result is a smoother user experience with the cost of less trust in the correctness of the information.

The official PR for the RFC can be found here. Check the date of the latest change to see which version is newest. Pull request on RFC002.

Uitgangspunten

Uitgangspunten bij de voorgestelde oplossing zijn als volgt:

1. Minimale impact op de ontwikkelcapaciteit van leveranciers. We weten dat er nog veel moet gebeuren voor de eOverdracht en dat de deadline snel nadert. We stellen een oplossing voor die idealiter in één dag is in te bouwen bij de leveranciers.
2. Faciliteren van een groeipad. We weten dat het gebruik van persoonlijke authenticatiemiddelen binnenkort verplicht gaat worden via wetten als de WDO en de Wegiz icm diverse NEN normen. De oplossing maakt het mogelijk voor zorginstellingen en leveranciers om eenvoudig het niveau op te hogen.
3. Voldoen aan de NEN7513. Om te kunnen voldoen aan deze norm is het belangrijk te weten welke gebruiker welke gegevens heeft ingezien. De voorgestelde oplossing voldoet aan deze eis.

Oplossingsrichting

Assurance Level eis

De zorginstelling moet zelf kunnen bepalen welke eisen ze stellen aan het vertrouwensniveua van het gebruikte middel. Daarom stellen we een nieuw veld `authenticationAssuranceLevel` voor in het `AuthorizationCredential` waarin een betrouwbaarheidsniveau van het gebruikte middel wordt opgegeven.

Het Nuts afspraken stelsel schrijft vervolgens een lijst van levels en middelen voor, bijvoorbeeld:

middel	level
EmployeeIdentity	low
Yivi(IRMA)	middle/substantial
Uzi	high

Dit level moet worden gezet per resource door de bronhouder. De opvragende partij vraagt de gebruikersidentiteit uit op het gevraagde niveau.

Voorbeeld gebruik

Onderstaand object is een voorbeeld van een credentialSubject van een `AuthorizationCredential` waar het `authenticationAssuranceLevel` veld wordt gebruikt:

```
{
  "id": "did:nuts:SjkuVHVqZndMVVJwcnUzbjhuZklhODB1M1M0LW9LcWY0WUs5S2",
  "legalBase": {
    "consentType": "explicit",
    "evidence": {
      "path": "pdf/f2aeec97-fc0d-42bf-8ca7-0548192d4231",
      "type": "application/pdf"
    }
  },
  "localParameters": {...},
  "resources": [
    {
      "path": "/DocumentReference/f2aeec97-fc0d-42bf-8ca7-0548192d4231",
      "operations": ["read"],
      "userContext": true,
      "authenticationAssuranceLevel": "low"
    }
```

```
  ],
  "purposeOfUse": "test-service",
  "subject": "urn:oid:2.16.840.1.113883.2.4.6.3:123456780"
}
```

Introductie van een nieuw EmployeeIdentity middel

Het probleem wat we op proberen te lossen is de problematiek bij implementatie, acceptatie en uitrol van persoonlijke authenticatiemiddelen. We verwachten dat zorginstellingen meer tijd nodig hebben intern beleid, ondersteuning en uitrol van deze middelen in te richten en op te tuigen. Ook is de keuze uit middelen nog beperkt waardoor deze nog niet aansluiten bij de werkprocessen van zorginstellingen. Zorginstellingen geven aan terug te willen vallen op bestaande afspraken onderling en elkaar daarin te vertrouwen met het identificeren van medewerkers. Daarom introduceren we het `EmployeeIdentity` middel. Dit middel is van niveau `low` en gebruikt de identiteit van de ingelogde gebruiker.

Dit middel is een drop-in replacement voor het Yivi(IRMA) middel.

Create signing session

Current payload for the IRMA means:

```
POST /internal/auth/v1/signature/session
{
  "means": "irma",
  "payload": "EN:PractitionerLogin:v3 I hereby declare to act on behalf of CareBears located in CareTown. This declaration is valid from Monday, 2 January 2006 15:04:05 until Monday, 2 January 2006 17:04:05."
}
```

Proposed payload for the EmployeeIdentity means:

```
POST /internal/auth/v1/signature/session
{
  "means": "employeeIdentity",
```

```
  "params": {
    "employer":"did:123",
    "emplopyee": {
      "identifier": "481",
      "roleName": "Verpleegkundige niveau 2",
      "initials": "J",
      "familyName": "van Dijk",
      "email": "j.vandijk@example.com"
    }
  },
  "payload": "EN:PractitionerLogin:v3 I hereby declare to act on behalf of CareBears located in CareTown. This declaration is valid from Monday, 2 January 2006 15:04:05 until Monday, 2 January 2006 17:04:05."
}
```

Response:

IRMA:

```
{
  "sessionID": "123",
  "sessionPtr": {QRCode},
  "means": "irma"
}
```

The frontend renders the contents of the `sessionPtr` as a QRCode.

EmployeeIdentity:

```
{
  "sessionID": "123",
  "sessionPtr": { "url":"nuts.example.com/public/auth/employeeID/123" },
  "means": "irma"
}
```

De frontend uses the `url` from the `sessionPtr` object to open a new browser window or render the page in a iframe.

Polling for the session status

IRMA en EmployeeIdentity:

`GET /internal/auth/v1/signature/session/{sessionID}`

Pending, as long as the user is signing the contract:


```
{
   "status": "pending"
}
```


Completed:


```
{
  "status": "completed",
  "verifiablePresentation": {
    "@context": [
      "https://www.w3.org/2018/credentials/v1",
      "https://nuts.nl/credentials/v1",
      "http://schema.org/",
      "https://w3c-ccg.github.io/lds-jws2020/contexts/lds-jws2020-v1.json"
    ],
    "type": ["VerifiablePresentation", "NutsSelfSignedPresentation"],
    "verifiableCredential": [
      {
      "issuer": "did:nuts:careOrg",
        "type": ["VerifiableCredential", "NutsEmployeeCredential"],
        "expirationDate": "2023-04-03T20:34:17.687862+01:00",
        "credentialSubject": {
          "type":"Organization",
          "id":"did:nuts:123456789",
          "member":{
            "type":"EmployeeRole",
            "identifier": "481",
            "roleName": "Verpleegkundige niveau 2",
            "member": {
              "type": "Person",
              "initials": "J",
```

```
            "familyName": "van Dijk",
            "email": "j.vandijk@example.com"
          }
        }
      }],
    "proof": {
      "type": "JsonWebSignature2020",
      "verificationMethod": "did:nuts:123456789#key-1",
      "created": "2023-04-03T16:34:17.687862+01:00",
      "jws":
"eyJhbGciOiJFUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19..hKcboC8m6YnZPi6ReJAYs0J0Ztn5nxcx2EavoXdtr
kWxmE1JZmImW89_8IIgjvfI8XtGeDlEnGywAuY2u7y9Bw"
    }
  },
  "proof": {
    "challenge": "LOGIN CONTRACT",
    "type": "JsonWebSignature2020",
    "verificationMethod": "did:nuts:123456789#key-1",
    "created": "2023-04-03T16:34:17.687862+01:00",
    "jws":
"eyJhbGciOiJFUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19..hKcboC8m6YnZPi6ReJAYs0J0Ztn5nxcx2EavoXdtr
kWxmE1JZmImW89_8IIgjvfI8XtGeDlEnGywAuY2u7y9Bw"
  }
 }
}
```


Usage of the identity token

The usage of the returned identity is the same for IRMA and EmployeeIdentity: the value of the `verifiablePresentation` field is base64 encoded and added to the access token request in the `usi` field.

TODO

- **Remove evidence (done)**
  Evidence in the presentation does not provide enough evidence and might even expose internal IPs.

- **Add new types to the Nuts JSON-LD context**
  Add both NutsEmployeeCredential and NutsSelfIssuedPresentation to the JSON-LD context.
- **Create RFC for NutsSelfSignedPresentation**
  This presentation hints to a verifier that the containing credential(s) have the same subject and issuer.

(Open) Issues

- **Fix backwards compatibility of JSON-LD context**
  How to make sure signatures on the new NutsAuthorizationCredential are valid when the new field `assurenceLevel` is added but not all nodes are updated and have the new context?
- **Proof of ownership for the credential**
  How to validate the proof signature of presentation? Normally the holder == subject and signs the presentation. Now the presentation is signed by the issuer. Solution: add the NutsOrgCredential to the presentation, add extra `NutsSelfIssuedPresentation` type to the presentation. e.g. `trusted(vendor)? && Verify(vendor -> org -> employeeRole)`
- **Make sure that the form can be embedded inside an iframe**
  Do research for how to configure CORS. Do we need to provide css for usage with and without iframe?

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 thechniques, 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 tipical 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 leverange the existing trust in the certigicate 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	`subject:O`	`2.5.4.10`
City	`subject:L`	`2.5.4.7`
Identifiers	`san:otherName`	`2.5.5.5`

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, hiarchy, 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