Authorization request

Note: Work in progress!

The process of requesting the authorization to perform operations on a set of resources. The custodian must check if all the requirements are met, therefore the response will be a-sync.

Relevant standards

- OpenIDConnect for Credential Issuance

Terms

The following information is needed:

scope The scope of the authorization request. In case of patient data, a patient identifier such as a BSN should be provided.

legal-basis A set of legal basis which can be used to grant the request. Examples: implied consent, explicit consent, legal task, emergency.

custodian Party who has the legal obligation to keep the data safe and only share under certain conditions.

requestor Party who wants to perform the operations on the data.

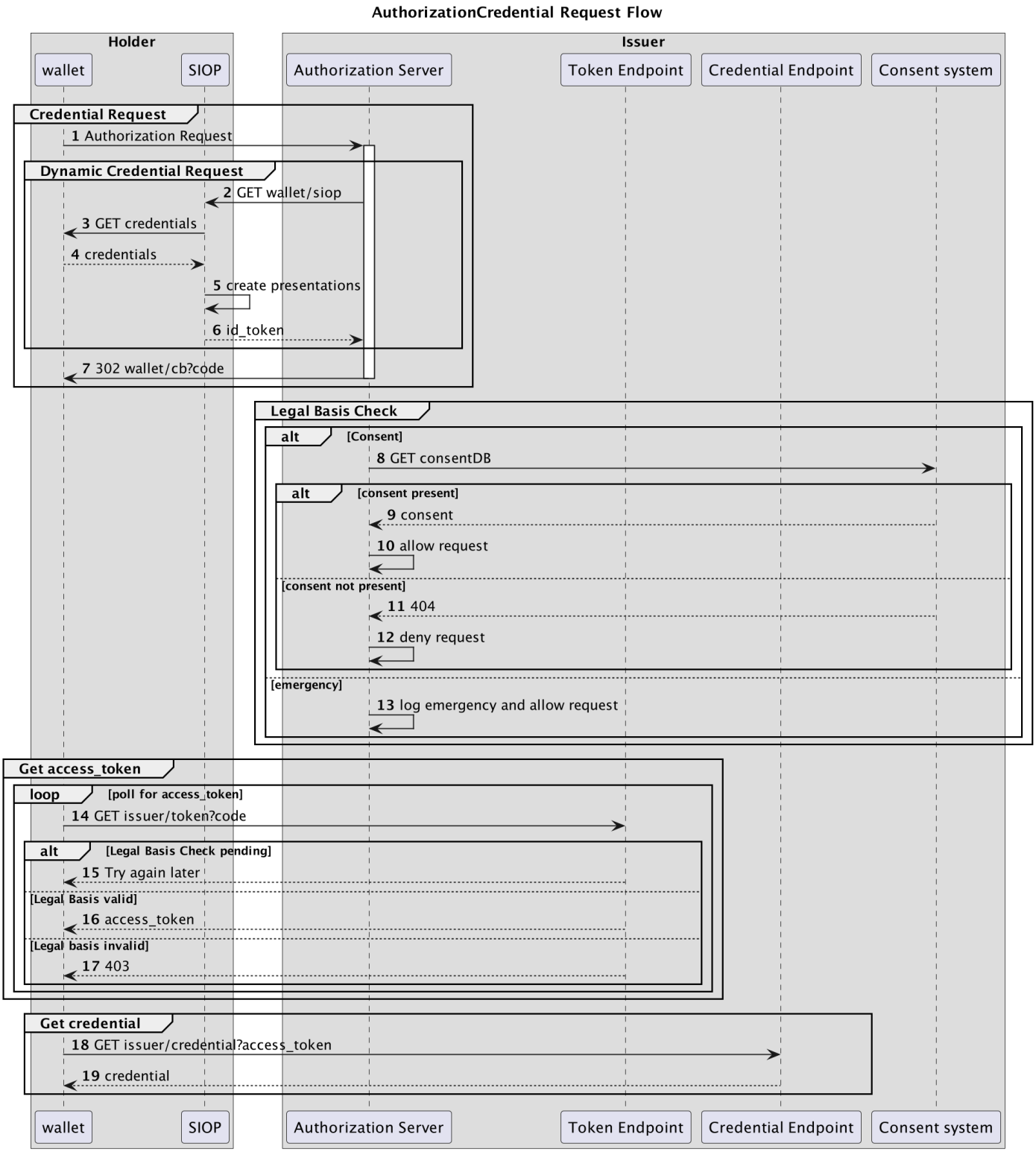
Use-case

A GP wants to read a report about a patient written by a caregiver at another care organization (custodian). The GP requests access to this report(scope). The custodian needs a patient consent in order to grant this request. It will look into its own consent system(s) to find it. If it is there, the GP receives an authorization in the form of a `AuthorizationCredential`.

Implementing using the OIDC4VCI credential request

We will base the implementation based on the new OpenID Connect standard. This way wallets and issuers will interact in a standardized way which makes this solution interoperable with other wallets and issuers.

The interactions are shown in the following diagram:



The issuer is the **Custodian** and the Holder is the **Requestor** (GP in our use-case example).

Let's walk through all the steps:

Credential Request

The goal for this group is the **Requestor** asking the **Custodian** for authorization to a specific (set of) resources. The authorization will be given in the form of a credential. This credential can later be used to acquire an `access_token` for the actual resource.

So why not directly issue an `access_token` for the resource instead? Multiple reasons:

1. The authorization request might take some time to process. An actual person might need to check the request for example. The OIDC4CI standard provides a flow for this, a normal OIDC flow doesn't.
2. The `AuthorizationCredential` is issued to the organization, but the `access_token` should be bound to a user. So, for each user session, a new `access_token` coupled to a verifiable user identity should be issued.
3. Access tokens should be short-lived, after they are issued it is no longer possible for the resource server to challenge the holder if it was the original requestor.

1. The requestor performs an AuthorizationRequest.

This is described in 5.1. Using the `authorization_details`:

```
[{
  "type":"openid_credential",
  "format": "ldp_vc",
  "credential_definition": {
    "@context": [
      "https://www.w3.org/2018/credentials/v1",
      "https://nuts.nl/credentials/v1"
    ],
    "types": [
      "VerifiableCredential",
      "NutsAuthorizationCredential"
    ],
    "credentialSubject": {
      "id": "did:nuts:123",
      "purposeOfUse": "careviewer",
      "patient":"bsn:9999992"
    }
  }
}]
```

Example request:

```
GET /authorize?
  response_type=code
  &client_id=s6BhdRkqt3
  &code_challenge=E9Melhoa2OwvFrEMTJguCHaoeK1t8URWbuGJSstw-cM
  &code_challenge_method=S256
```

```
&authorization_details=%5B%7B%22type%22%3A%22openid_credential%22%2C%22format%22%3A%22ldp_vc
%22%2C%22credential_definition%22%3A%7B%22%40context%22%3A%5B%22https%3A%2F%2Fwww.w3.org
%2F2018%2Fcredentials%2Fv1%22%2C%22https%3A%2F%2Fnuts.nl%2Fcredentials%2Fv1%22%5D%2C%22typ
es%22%3A%5B%22VerifiableCredential%22%2C%22NutsAuthorizationCredential%22%5D%2C%22credentialSu
bject%22%3A%7B%22id%22%3A%22did%3Anuts%3A123%22%2C%22purposeOfUse%22%3A%22careviewer%2
2%2C%7D%7D%7D%5D
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
Host: https://server.example.com
```

Dynamic Credential Request

The issuer wants to know proofs about the validity of the request. Using the dynamic credential request, as defined in section 5.

Steps 2 - 6

These steps are asking the requestor for additional proofs (VCs) about its identity and capabilities. The issuer sends a challenge and the holder embeds the VCs in a Verifiable presentation.

7. Authorization Response

The Authorization Server has received the request, creates "session" identified by a code and redirects the user-agent to the callback, providing the code in the request parameter.

Example Response:

```
HTTP/1.1 302 Found
  Location: https://client.example.org/cb?
    code=SplxlOBeZQQYbYS6WxSbIA
```

Legal Basis Check

After checking the identity and additional claims of the requestor, the issuer has to check if the request can be allowed. One of the things to check is the legal basis. Depending on the use-case different legal-basis are relevant. This example shows an emergency and an explicit consent provided by the patient. There exists other legal basis which are not shown in this example.

These steps might be almost instantaneously or take hours or days, depending on the use-case and possible manual checks. Meanwhile, the wallet will start polling and retrying based on the interval response from the AS.

8. GET consent from local consent system

Steps 8 to 12 describe the flow where a consent is the legal basis.

The custodian has the responsibility to administer patient consent for sharing data. The custodian has to check its administration if there is a consent.

This request must be standardized so all consent systems can support the API and make integrations between Authorization Servers and consent systems universal.

9. Consent found

The consent system response positively on the query.

10. The Authorization Server allows the request

After receiving positive confirmation from the consent system, the AS registers this request as valid.

Steps 11 - 12 No consent

No consent was found, deny the request

13. Emergency

This is an example where an emergency can be a legal basis. The request is logged and allowed instantaneously.

Get access_token

After step 7, when the wallet receives the authorization code, the wallets starts polling the token endpoint for the `access_token`.

14. GET AS/token?code

The wallet tries to get the access_token from the token endpoint. According to OIDC4VCI section 6.1

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

  grant_type=authorization_code
  &code=SplxlOBeZQQYbYS6WxSbIA
  &code_verifier=dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
  &redirect_uri=https%3A%2F%2FWallet.example.org%2Fcb
```

Note: the value of the Authorization header must be specified. Probably based on RFC7523

15. Legal basis check pending

The legal basis checks are not finished yet, the token endpoint instructs the client to try again later by responding with the `authorization_pending` field.

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

 {
    "authorization_pending": true,
    "interval": 300
 }
```

16. Legal basis valid

The legal base is valid, the wallet now receives the access_token:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

 {
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6Ikp..sHQ",
  "token_type": "bearer",
  "expires_in": 86400,
  "c_nonce": "tZignsnFbp",
  "c_nonce_expires_in": 86400
 }
```

17. Legal basis invalid

OIDC4VCI section 6.3 There was not valid legal basis, the token endpoint responds with an error response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store

{
  "error": "legal basis invalid"
}
```

Note: this should be specified in more detail.

18. Get credential

Using the `access_token` and the `n_once`, the wallet can request the credential:

```
POST /credential HTTP/1.1
Host: server.example.com
Content-Type: application/json
Authorization: BEARER czZCaGRSa3F0MzpnWDFmQmF0M2JW

{
  "format": "ldp_vc",
  "credential_definition": {
    "@context": [
      "https://www.w3.org/2018/credentials/v1",
      "https://nuts.nl/credentials/v1"
    ],
    "types": [
      "VerifiableCredential",
      "NutsAuthorizationCredential"
    ],
    "credentialSubject": {
      "id": "did:nuts:123",
      "purposeOfUse": "careviewer",
      "patient":"bsn:9999992"
    }
  },
  "proof": {
    "proof_type": "jwt",
    "jwt": "eyJraWQiOiJkaWQ6ZXhhbXBsZ...KPxgihac0aW9EkL1nOzM"
  }
}
```

Note: the value of the Authorization header must be specified. Probably based on RFC7523

19. Credential response

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
```

```
{
   "format": "ldp_vc",
   "credential": {
      "@context": [
         "https://www.w3.org/2018/credentials/v1",
         "https://nuts.nl/credentials/v1"
      ],
      "id": "did:web:issuer#123",
      "type": [
         "VerifiableCredential",
         "NutsAuthorizationCredential"
      ],
      "issuer": "did:web:custodian",
      "issuanceDate": "2010-01-01T00:00:00Z",
      "credentialSubject": {
         "id": "did:web:requestor",
         "patient":"bsn:999992",
         "purposeOfUse": "careviewer",
      },
      "proof": {
         "type": "Ed25519Signature2020",
         "created": "2022-02-25T14:58:43Z",
         "verificationMethod": "https://example.edu/issuers/565049#key-1",
         "proofPurpose": "assertionMethod",
         "proofValue": "zeEdUoM7m9cY8ZyTpey83yBKeBcmcvbyrEQzJ19rD2UXArU2U1jPGoEt
                        rRvGYppdiK37GU4NBeoPakxpWhAvsVSt"
      }
   },
   "c_nonce": "fGFF7UkhLa",
   "c_nonce_expires_in": 86400
}
```


Revision #7
Created 10 March 2023 09:01:46 by Steven van der Vegt
Updated 11 April 2023 12:37:22 by Steven van der Vegt