EUROPEAN COMMISSION
DIRECTORATE-GENERAL INFORMATICS

Directorate D - Digital Services
DIGIT D3 - Trans-European Services

eIDAS Bridge
WP2 - Software requirements specification and eIDAS bridge development

SSI Framework Integration Guidelines

TABLE OF CONTENTS

1. OBJECTIVE

The following document is intended for developers and integrators that wish to add eSealing functionality to an existing SSI architecture (a wallet).

It briefly describes the components and use cases that can be accomplished, the requisites, the setup steps and how to perform signatures and validations.

2. OVERVIEW

eIDAS Bridge is a component that will be used by a Wallet in order to provide electronic sealing functionalities. The main reason to do so is to abstract and separate the responsibilities of each of the components. Generally speaking, Wallets are supposed to be built to support SSI-related use cases, that do not necessarily include full eSealing functionality (as defined under eIDAS). The eIDAS Bridge component will provide "the bridge" between the user's wallet and allow to call esealing-capabilities.

This "bridge" component is first and foremost seen as a component that will be deployed "next to the wallet" as to support the holder's wallet to do eSealing and (ideally) to call the holder's HSM to execute this eSealing. In future releases this design will need to be elaborated as to ensure both "local" deployment and to support "remote eSealing" with all the security measures required under eIDAS for remote eSealing.

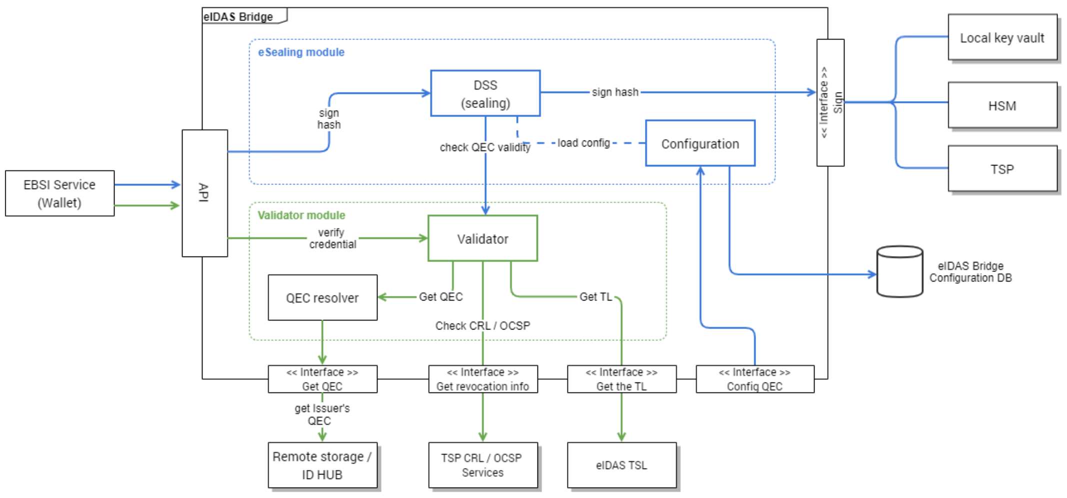
2.1. Electronic sealing components

There are 2 main flows related to the electronic sealing of VCs:

- the creation of the electronic seal by the Issuer (of credentials).
- the verification, by either the Holder or by the Relying Party (the verifier).

2.2. The logical structure of the eIDAS bridge for electronic seal component

Below we can see the detail of the eIDAS bridge for eSealing component with the different main modules involved: electronic sealing and validation.



3. Requisites

In order to integrate with eIDAS bridge and be able to use it to sign and verify credentials you will need to:

- Obtain a X509 certificate in a PKCS12 format. If you intend to sign credentials under eIDAS, please contact a TSP in order to issue a QEC for eSealing.
- An instance of the eIDAS Bridge running (check project README for installing and setup).
- A DID issued to the entity you represent, and access and control of its associated private key.
- A public URL where the QEC certificate can be stored and retrieved.

Then the following setup steps will need to be fulfilled.

4. Setup

If you are running the eIDAS Bridge locally, by default, the service will be located at `http://localhost:5002` and will expose the following endpoints:

```
/eidas/load-qec
/eidas/service-endpoint
/eidas/get-pubkey
/eidas/sign-credential
/eidas/verify-credential
```

4.1. Load QEC to perform sealing

First, you will have to load the certificate in eIDAS Bridge instance. To do so, do a `POST` call to `/eidas/load-qec` endpoint with the following JSON object containing the QEC:

```
{

  "did": "did:example:21tDAKCERh95uGgKbJNHYp",

  "p12data":
"308203c90201033082038f06092a864886f70d010701a08203800482037c3082037830820
27706092a864886f70d010706a0820268308202640201003082025d06092a864886f70d010
701301c060a2a864886f70d010c0106300e0408273ffca45dbf806b02020800808202307d4
1394e3b892f6a8e38c5373d1d21ae2130cdaf377d634e3eca63b66e71558c3e1f66cee435c
f0db87e4ff1aed5de200c68e1e35d90bca5634276bcbfc31af291a69ad1ac223fea6c4806f
0ceb89673c7ca0b4baae72ba5e3b78207fc1d253c959d3affedc703b97fa5bbabfd6017011
f70e657c788dea4a7d28fb4386d5bcd6cb6ba23304297ae5b8df499a4cb94ea3e04bcc727c
2c4c06201e7b64ffeda417eb7a88a02827fe91fae8cef3df69028dd4be5ea4224584384555
c4e3d7574c9e55de20be880c384477e4da82bb1b0734ccbb726b46a514d8c8c8bbebe375a6
7686a1bf500e106662ebbf598f86a24a2c94ce10853bec25a1466b6a3a28d49d1077e089ed
fd222c9aac861c55189ac917b9637485d6330a24ee4be1418e361438842800d8490cff89c3
2cc73ea7e6267994f550457553ac9039efa961344897ded92c53f47f7dba30e673b01932d8
8a53bc6c13ae393fa9a39dee53a5c888a92f954b47ca4f3ec542dc0a759f30f0cb39206099
f1c969cf42afc2e1b372ea2801bd2f84451a4861316451b259025ea4ff0230687e08afeccb
0b64608b179cac8920b44d6d0b2fda45b5be3cfe29c0097bc950f8aafbb8ca6aa5d35b3015
bc161e906408abcd85dba2726d6aa5f9e2b63fb9dc6bcd81de574e89e4c408099ee5612060
461aa374abda1e370325f9bd7b6dd98f16b78199ac8de5850a615d7b5c9b14eef625ab68e9
1ffa287378466c1eecd371d5a70c4cee31fda0e9f59a3ce0b7e4eaae691ee3cf013a13a1a0
fdfce253081fa06092a864886f70d010701a081ec0481e93081e63081e3060b2a864886f70
d010c0a0102a081ac3081a9301c060a2a864886f70d010c0103300e0408fae73712fbcb42e
90202080004818828de9f3fc3cc2366b93e12a1d7ef46eaa3d88876251c76f849b44b1ec67
185e29561832a7049401371193edbceb08620166ecec1e0636cd75f48c28cacdcc53cb538e
6fafcf8f54852488317d528cf3b3d7a1272187a957f7ab4aceb30e8c4b54fae3ea037433e2
cd2d33bbe51abdcb001d89373fc6875aebe972519b5f4bcb90583cc292fbc1665312530230
6092a864886f70d01091531160414 5d468eb5cd8c0db171cdf38758809f04ac95ebc830313
021300906052b0e03021a05000414136d9fe56d935eb5478171f41a4ef290de62b0360408f
9101c1c3a66cab702020800",

  "password": "passphrase"

}
```

4.2. Store the QEC certificate in a publicly accessible URL

The QEC should be publicly available via HTTP so that any bridge or wallet can get it and validate the signatures done with the certificate's private key.

For testing purposes only, or if you will only be validating signatures using the same certificate than the previously uploaded one, you can skip this step.

4.3. Update the `publicKey` element in the DID document

Then, you will need to update the DID Document of your DID in order to add the public key of your certificate and the address where the full QEC can be retrieved. This step is required in order to allow any entity to validate a signature signed with the QEC private key without any previous knowledge.

EBSI is proposing the following structure of the `publicKey` element that should hold the QEC key:

```
    "publicKey": [{
        "id":"did:ebsi:FlemGov#eidasKey",
        "type": "RsaVerificationKey2018",
        "expiration": "2024-11-28",
        "controller": "did:ebsi:qtsp",
        "publicKeyPem":"---PUBLIC KEY...",
        "LoA": "High",
        "service": {
            "id": "did:ebsi:FlemGov#verifiableID-eidasKey",
            "type": "VerifiableIDService",
            "serviceEndpoint":    "https://VID-id-hub.service-endpoint.gov"
// ID Hub is a QTSP provided or self-hosted service
        },
        "proof": {
            "type": "RsaSignature2018",
            "created":"2019-11-29",
            "verificationMethod":"did:ebsi:qtsp#eidasKey",
            "jws":"ab...321"
        }]
```

From all the properties, only those in bold are required for the eIDAS Bridge to validate the signatures:

- `publicKeyPem` should have the public key of the certificate, PEM encoded.
- `service`: should indicate where the QEC certificate can be obtained.

4.3.1. Getting the publicKeyPem element

eIDAS Bridge API exposes and endpoint to retrieve the public key of the QEC uploaded by sending a `POST` request to `/eidas/get-pubkey` endpoint with the following payload:

```
{
  "did": "did:example:21tDAKCERh95uGgKbJNHYp"
}
```

As a response you will get something like:

```
{
  "publicKeyPem": "-----BEGIN PUBLIC KEY-----
\nMFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAEswOm6PqrB3ddfKCWZPWMzSESDrd8xtcl\ndd8sCK
vtW1+UC6s0g79GxkAJznPZ6Vu4DI0CJkMvbe+pF5ykUz7D+g==\n-----END PUBLIC KEY---
--\n"
}
```

5. SIGN AND VERIFY

5.1. Sign

In order to eSeal credentials, do a `POST` call from the Wallet to `/eidas/sign-credential` endpoint with the following JSON object containing the credential payload that needs to be eSealed:

```
{
  "@context": {
    "schema": "http://schema.org/",
    "homepage": "schema:url"
  },
  "id": "http://example.edu/credentials/3732",
  "type": "[\"VerifiableCredential\", \"UniversityDegreeCredential\"]",
  "issuer": "did:example:21tDAKCERh95uGgKbJNHYp",
  "issuanceDate": "2010-01-01T19:23:24Z",
  "credentialSubject": {
    "did": "did:example:21tDAKCERh95uGgKbJNHYp",
    "degree": {
      "type": "BachelorDegree",
      "name": "Bachelor of Science and Arts"
    }
  }
}
```

If succeeded, the API will response with a 200 status code and a success message:

```
{
  "message": "success"
}
```

5.2. Verify

In order to verify a credential, just send it with a POST request to `/eidas/verify-credential.`

If the signature is correct, the API returns with a `200 code status` and the message "`VALID`":

```
{
  "message": "VALID"
}
```