University of Trento

SELF-SOVEREIGN IDENTITY
Custom DID method

Accademic Year 2021/2022

Supervisors:
Prof. Fabrizio Granelli
Dr. Mattia Zago

Graduate Student:
Marco Xausa

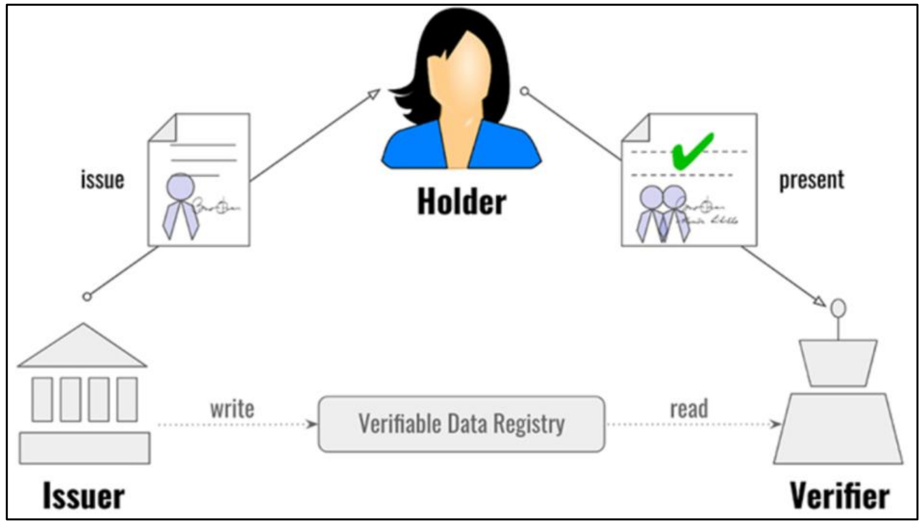
Summary

Introduction

The different versions of the Internet trusted different identity and access management standards. The first version, the Web1.0 was based on the **login and password** paradigm. With the developing of technologies and functionalities, a new pattern was introduced with the Web2.0, the **Single Sign On**. It allows the user to perform the authentication only once for several different services.
In any case, with the Web heading to its third version, new objectives have to be achieved. The **Self-Sovereign Identity** aims to reach such objectives, giving users more control of their data and bringing decentralization to identity and access management.

Decentralized Identity is a form of Self-Sovereign Identity that exploits the distributed ledger technology to ensure the chain of trustworthiness, allowing entities to provide information about themselves without any centralized request. It is composed by different layers:

- Applications and wallets
- Agent layer
- Verifiable Credentials (VC)
- Decentralized Identifiers (DID) and Verifiable Data Registries (VDR)



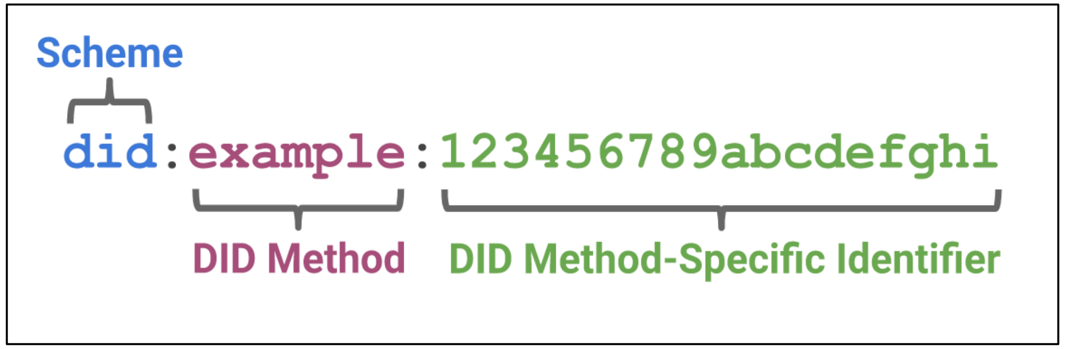
Context

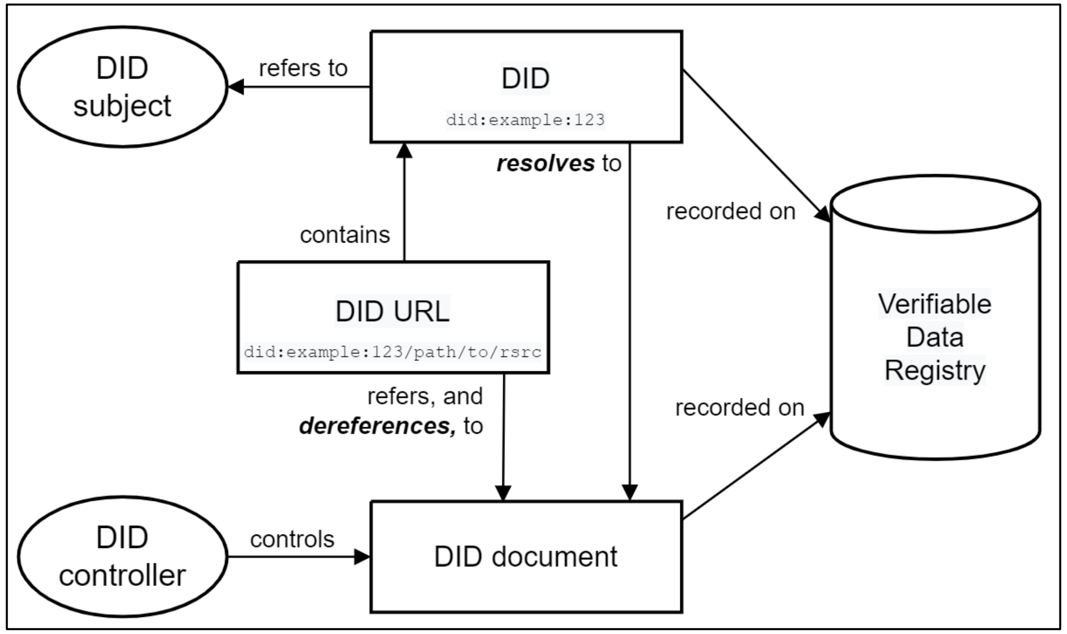
A DID is a unique string that functions as a decentralized address. It refers to a subject and can be controlled by one or more entities.

There are different components involved in the DID infrastructure :

- DID subject
- DID controller
- DID URL
- DID Document
- Verifiable Data Registry (VDR)

DID method : rules these interactions, describing which actions are allowed and how they are performed.


Scheme
did:example:123456789abcdefghi
DID Method
DID Method-Specific Identifier



did:monokee

Monokee is a web-centric and cloud-based Identity and Access Manager. The company’s product category refers to IDAAS (IDentity as a Service). The identity orchestration component has been extended to **support self-sovereign identity** in order to help companies in the transaction to a decentralized model.

monokee

Therefore, the main purpose of this DID method is to **group different representations** of the same issuer within the various ledgers to facilitate the managing of Verifiable Credentials from different environments.

The interaction with a VDR has been emulated by an **API generalization**. These APIs are meant to store data in a MongoDB database and are supposed to run in the localhost at the 8080 port. This generalization is expected to be further specialized.

Method syntax

A monokee DID can be crated using the following schema:

```
did = "did:monokee:" method-specific-identifier
```

Where the `method-specific-identifier` conforms to the Universal Unique Identifiers v4 with the following ABNF rule:

```
method-specific-identifier = time-low "-" time-mid "-"
                             time-high-and-version "-"
                             clock-seq-and-reserved
                             clock-seq-low "-" node
time-low                   = 4hexOctet
time-mid                   = 2hexOctet
time-high-and-version      = 2hexOctet
clock-seq-and-reserved     = hexOctet
clock-seq-low              = hexOctet
node                       = 6hexOctet
hexOctet                   = hexDigit hexDigit
hexDigit = "0" / "1" / "2" / "3" / "4" / "5" / "6" / "7" / "8" / "9" /
           "a" / "b" / "c" / "d" / "e" / "f"
```

CRUD - Create

To proceed with the creation of a DID the required components are:

- `method-specific-identifier`
- Ed25519/x25519 key pair

The DID can be derived using the syntax rule. The public keys are used to create the DID document following the structure on the image.

```
did:monokee:bdc78193-5a5c-4584-83cd-a08f523ebdb7
```

```
{
  "@context": [
    "https://w3id.org/did/v1",
    "https://w3id.org/security/suites/ed25519-2018/v1",
    "https://w3id.org/security/suites/x25519-2019/v1"
  ]
  "id": "did:monokee:bdc78193-5a5c-4584-83cd-a08f523ebdb7",
  "verificationMethod": [
    {
      "type": "Ed25519VerificationKey2018",
      "id": "did:monokee:bdc78193-5a5c-4584-83cd-a08f523ebdb7#key-1",
      "controller": "did:monokee:bdc78193-5a5c-4584-83cd-a08f523ebdb7",
      "publicKeyBase58": "<encoded_public_key>"
    },
    {
      "type": "X25519KeyAgreementKey2019",
      "id": "did:monokee:bdc78193-5a5c-4584-83cd-a08f523ebdb7#key-agreement-1",
      "controller": "did:monokee:bdc78193-5a5c-4584-83cd-a08f523ebdb7",
      "publicKeyBase58": "<encoded_key>"
    }
  ],
  "authentication": [
    "did:monokee:bdc78193-5a5c-4584-83cd-a08f523ebdb7#key-1"
  ],
  "assertionMethod": [
    "did:monokee:bdc78193-5a5c-4584-83cd-a08f523ebdb7#key-1"
  ],
  "keyAgreement": [
    "did:monokee:bdc78193-5a5c-4584-83cd-a08f523ebdb7#key-agreement-1"
  ],
  "service": [
    {
      "id": "did:monokee:bdc78193-5a5c-4584-83cd-a08f523ebdb7#sov-did-1",
      "type": "SovrinIssuer",
      "serviceEndpoint": "did:sov:WRfXPg8danfKVubE3eX8pw"
    }
  ]
}
```

CRUD - Create

The DID document can then be sent to server using an HTTP POST call with the following body structure:

```
{
  didDocument:<document>,
  signature:<base58_encoded_signature>
}
```

The signature must be created using the DID document and the private key related to the public key in the `#key-1` verification method.
The endpoint where the HTTP call has to be sent is:

```
http://localhost:8080/createdid
```

CRUD – Read & Dereference

The **resolution** of a DID can be performed sending an HTTP GET request to the following endpoint:

```
http://localhost:8080/resolvedid/<did>
```

The response will encompass either the document and the metadata:

```
{
  "@context": "https://w3id.org/did-resolution/v1",
  "didDocument": [...],
  "didResolutionMetadata": {
    "contentType": "application/did+ld+json"
  },
  "didDocumentMetadata": {
    "created": "2019-03-23T06:35:22Z",
    "updated": "2023-08-10T13:40:06Z"
  }
}
```

A DID URL can be used to retrieve one of the following:

- A verification method
- A service
- A DID document related to a did:sov

And is establised by concatenating a fragment with the relative identifier on the DID.

```
did:monokee:bdc78193-5a5c-4584-83cd-a08f523ebdb7#key-1
```

The **dereferenciation** process looks for the component in the DID document after resolving the DID. If it is a `did:sov`, it is resolved by and the respective document is returned.

CRUD - Update

The updating capabilities for this method are the following:

- Replace the `#key-1` verification method's public key by sending an HTTP UPDATE request with the following data:

```
{
  methodUrl:<methodUrl>,
  newKey:<newKey>,
  signature:<encodedSignature>
}
```

```
http://localhost:8080/updatekey
```

- Add an ed25519 verification method by sending an HTTP UPDATE request with the following data:

```
{
  "did":<did>,
  "verificationMethod":<verificationMethod>,
  "signature":<encodedSignature>
}
```

```
http://localhost:8080/addverificationmethod
```

CRUD - Deactivate

To deactivate a DID an HTTP DELETE request to the following endpoint must be sent:

```
http://localhost:8080/deactivate/<did>
```

The request has to report a signature produced from the «deactivate» message and the private key related to the `#key-1` verification method’s public key:

```
{
    signature:<encodedSignature>
}
```

The DID document is not ereased from the database, however, the metadata will identify it as deactivated.

Considerations

Up to now, the `did:monokee` method is a technology demonstrator proving the feasibility of cross-chain DID method linking, a building block necessary to enable full interoperability within different competing standards in the DID ecosystem.

All the core functionalities have been implemented. However, future works are required to refine the result. The APIs generalization permits a variety of possible future specializations, such as ledger-agnostic DIDs or direct peer-to-peer VDR interactions. Ideally, the final result should still APIs, though the latter should interact with a VDR.

A potential way to achieve so is to exploit Walt.id SSI kit which groups different components to create an infrastructure that delivers SSI services. The objective is to use one of its components as a bridge between the API and different VDRs

UNIVERSITY OF TRENTO

Thanks for the attention!