aries-notebook

# SSI and Hyperledger Aries Cloud Agent[¶](#SSI-and-Hyperledger-Aries-Cloud-Agent)

## 1. Ecosystem overview[¶](#Xe383b24658054de516d7570c096020c142a36c0)

We present a brief overview of the technologies and protocols used and briefly describe the main terminology surrounding them in the following sections.

### 1.1 Self-sovereign identity (SSI)[¶](#Xa1a143d747d936f3e07bd733dc16be659175ad5)

Sources:

* <https://sovrin.org/wp-content/uploads/2017/06/The-Inevitable-Rise-of-Self-Sovereign-Identity.pdf>
* <https://en.wikipedia.org/wiki/Self-sovereign_identity>

To create the long-missing identity layer of the Internet, a new, trusted infrastructure is required which enables identity owners to share not only identity, but also verified attributes about people, organisations and things, with full permission and consent.

For identities to be truly self-sovereign, this infrastructure needs to reside in an environment of diffuse trust, not belonging to or controlled by any single organisation or even a small group of organisations. Distributed ledger technology (DLT) is the breakthrough that makes this possible. It enables multiple institutions, organisations and governments to work together for the first time by forming a decentralised network much like the Internet itself, where data is replicated in multiple locations to be resistant to faults and tampering. While distributed ledger technology has been around for some time, new DLTs such as Bitcoin and Ethereum have resulted in a greater realisation of its potential, particularly with respect to decentralisation and security.

When combined with distributed key management and peer-to-peer sharing of encrypted claims, DLT is what finally makes self-sovereign identity possible. Within this identity layer, mechanisms for discovery, routing of requests, exchanging of data and recording events can exist pervasively, with no single entity being in control.

The three pillars of the eco-system are: *Decentralized identifiers (DIDs)*, *Verifiable Credentials (VCs)* and the *Verifiable data registry (Blockchain)*. A verifiable credentials flow, which presents the role of each of the pillars is presented in the figure below. Further explanations are given in the following sections.

![image.png](data:image/png;base64;base64,)

Source: Figure based on <https://tykn.tech/verifiable-credentials/>

### 1.2 Decentralized Identifiers (DIDs)[¶](#X70fc40b2e9f06ec9f92ae8478075e9b25d3e3ff)

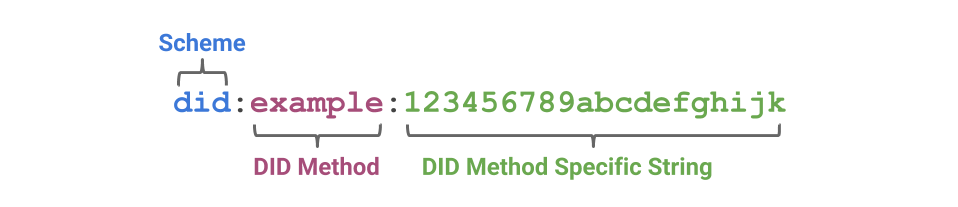
Sources:

* <https://www.w3.org/TR/did-core/>
* <https://w3c-ccg.github.io/did-primer/>

At a superficial level, a decentralized identifier (DID) is simply a new type of a globally unique identifier. But at a deeper level, DIDs are the core component of an entirely new layer of decentralized digital identity and public key infrastructure (PKI) for the Internet.

For self-sovereign identity, which can be defined as a lifetime portable digital identity that does not depend on any centralized authority, we need a new class of identifier that fulfills four requirements: persistence, global resolvability, cryptographic verifiability, and decentralization.

#### 1.2.1 DID format[¶](#X4218ae4e174216301ee368d55b4ad9a6e3debb7)



Source: W3C DID-primer

**DID methods** define how DIDs work with a specific blockchain, while a **DID method specific string** specifies the format of the method-specific identifier. The method specific identifier string **must** be unique in the namespace of that DID method.

#### 1.2.2 DID documents[¶](#X6882934c99aec00dba4b756686aa7e21c967c77)

The DID infrastructure can be thought of as a global key-value database, where the key is a DID, and the value corresponding to the DID is a **DID document**. The purpose of the DID document is to describe the public keys, authentication protocols and service endpoints necessary to bootstrap cryptographically-verifiable interactions with the identified entity. A *DID document* is a valid [JSON-LD](https://json-ld.org/) document.

### 1.3 Verifiable Credentials (VCs)[¶](#Xc8eee91c99914aa702f597c8eb12fb5c982f15a)

Sources:

* <https://www.w3.org/TR/vc-data-model>
* <https://en.wikipedia.org/wiki/Verifiable_credentials>
* <https://w3c.github.io/vc-use-cases>

It is currently difficult to transmit credentials such as driver's licenses, proofs of age, education qualifications, and healthcare data, via the Internet in a way that is verifiable yet protects individual privacy.

Verifiable Credentials are useful when a person needs to prove that they are:

* above a certain age,
* capable of driving a particular motor vehicle,
* require a particular medication,
* trained and certified as an electrician,
* professionally licensed to practice medicine,
* cleared to travel internationally etc.

Note that these use-cases are merely high-level examples. More use-cases, described more in-depth, can be found at [W3C VC-use-cases](https://w3c.github.io/vc-use-cases).

The *Verifiable Credentials* ecosystem is composed of four primary roles: 1) The **Issuer**, who issues verifiable credentials about a specific **Subject**.

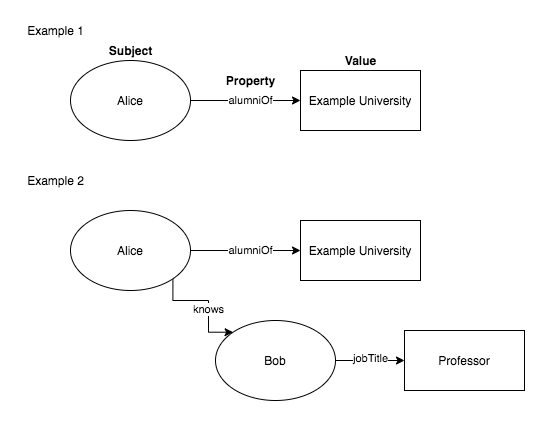
2) The **Holder** stores credentials on behalf of a **Subject**. **Holders** are typically also the **Subjects** of a credential.

3) The **Verifier** requests a **profile** of the **Subject**. A **profile** contains a specific set of credentials. The **Verifier** verifies that the credentials provided in the **profile** are fit-for-purpose.

4) The **Identity registry** is a mechanism, that is used to issue identifiers for **Subjects**.

A visual depiction of the ecosystem above and the corresponding information flow is presented below: ![image.png](data:image/png;base64;base64,) Source: Figure based on W3C vc-data-model

The ecosystem roles exchange data that enables the realization of the previously mentioned process. The data that is exchanged differs based on the roles participating, but is fundamentally composed of **Claims**,**Credentials**, and **Profiles/Presentations**.

* A **Claim** is a statement about a subject, expressed as a subject-property-value relationship. This enables a wide variety of statements, e.g. whether or not someone is over the age of 21, or whether someone graduated from a particular university. Such a claim is presented in the *Example 1* in the figure below. These claims may be merged together to express a graph of information about a particular subject. The *Example 2* extends the data model by adding claims that state that Alice knows Bob and that Bob is a professor.  Source: based on W3C vc-data-model
* A **Credential** or **Verifiable Credential(VC)** is a set of multiple claims, combined into a single document, where the data is digitally signed. A **Credential** is usually sent from the *Issuer* to the *Holder*.
* ![](data:image/svg+xml; qs=0.85;base64,)
* Source: W3C vc-data-model

A more complete depiction of a **VC** is shown in the figure below:

![](data:image/svg+xml; qs=0.85;base64,)

Source: W3C vc-data-model

Normally, a **VC** is composed of at least two information graphs. The first graph expreses the verifiable credential itself (labeled as *Credential graph* in the figure), which contains the credential metadata and claims. The second graph (labeled as *Proof graph*) expresses the digital proof, which is usually a digital signature.

* When a *Verifier* asks for data from a *Holder*, the *Holder* typically bundles a set of credentials into a data structure called a **Profile** or a **Verifiable Presentation** and signs the data signiture:
* ![](data:image/svg+xml; qs=0.85;base64,)
* Source: W3C vc-data-model

### 1.4 Verifiable data registry[¶](#X5229350a2405bd09d7d8760f40f9e1900b51867)

Sources:

* <https://www.w3.org/TR/vc-data-model/>
* <https://en.wikipedia.org/wiki/Distributed_ledger>

A **Verifiable Data Registry** is a role a system performs by mediation between the creation and verification of identifiers, keys and other relevant data, such as verifiable credential schemas, revocation registries, issuer public keys etc., which are required to use verifiable credentials. The most commonly used verifiable data registry is a distributed ledger, built atop of the *Distributed Ledger Technology(DLT)*, commonly called *Blockchain Technology*. This approach is used in order to establish a decentralized database, which provides control over the evolution of data between entities through a peer-to-peer network. Multiple *consensus algorithms* are used to ensure replication of data across the nodes of the network. In other words, the distributed ledger or a blockchain enables everyone in the network to have the same source of thruth about which credentials are valid and who attested to the validity of the data inside the credential, without revealing the actual data.

The main types of data, that are written to the blockchain in the context of Verifiable Credentials include:

* Public DIDs
* Credential Definitions: Credential definitions represent the templates to be stored on the ledger for different (often tangible) proofs of identity or qualification issued by authorities. These include drivers licenses, passports, identification cards, credit cards, etc.
* Schemas: Schemas provide formal descriptions for the structure of a credential or a template, which outlines the verified data you can issue or verify from the users.

#### 1.4.1 Hyperledger Indy[¶](#X2ff5a48e39afa11e0a136e6ef86e803f5f0a682)

Source:

* <https://hyperledger-indy.readthedocs.io/en/latest/>
* <https://www.hyperledger.org/use/hyperledger-indy>

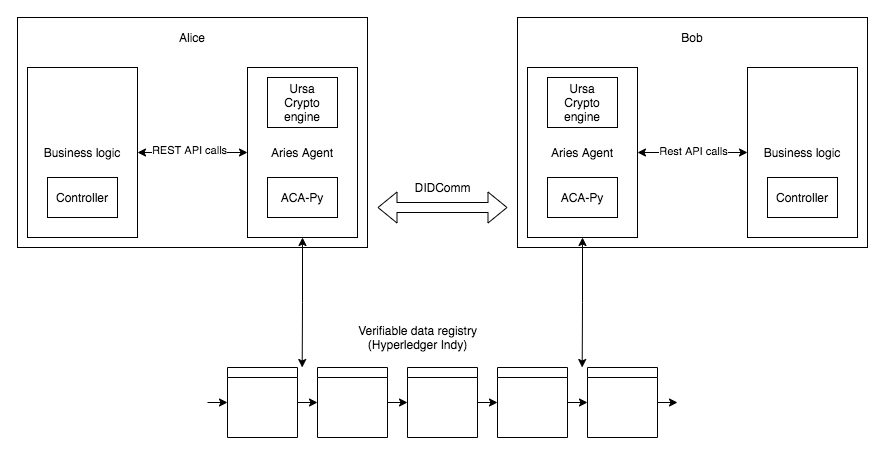
The Hyperledger Indy project strides to provide tools, libraries and reusable components for providing digital identities rooted on blockchains or other distributed ledgers. Two key parts of the project include the *Indy-node* and *Indy-sdk*. The *Indy-node* provides a way to run a distributed ledger, while the *Indy-sdk* allows the developer to interact with the ledger. In the following sections we use the **von-network** project, which provides us with a simple way to deploy multiple *Hyperledger-Indy* nodes, without having to manually set them up. Furthermore, the *von-network* project comes with a browser based ledger explorer UI, which allows for easier debugging and more transparancy.

### 1.5 Hyperledger Aries Cloud Agent[¶](#X40c23872d1dee071290eef1db4d499a1b2f08d9)

Sources:

* <https://github.com/hyperledger/aries-rfcs>

The [Hyperledger Aries Cloud Agent - Python](https://github.com/hyperledger/aries-cloudagent-python) is an open-source project, built on top of the concepts explained in the previous sections. It represents a foundation for building Verifiable Credentials (VC) ecosystems. It operates in the second and third layers of the [Trust over IP](https://trustoverip.org/) using [DIDComm messaging](https://github.com/hyperledger/aries-rfcs/tree/master/concepts/0005-didcomm) and [Hyperledger Aries protocols](https://www.hyperledger.org/use/aries).

An overview of the arcitecture is presented below: 

Alice and Bob represent the two entities (could be a person, a device, an organisation, etc.) trying to establish a connection and send messages to each other. An *Aries agent* is the implementation of a set of *aries-rfcs*, where all the necessary steps for establishing a connection, issuing a credential, multi-tenant ecosystems etc. are implemented. The cryptographic methods needed for the agents to work are implemented as a part of the *Hyperledger Ursa* project. The *contoller* is an implementation of the business logic to invoke certain aries protocols within the agent. In the presentation, shown in the next sections of this document, we play the role of the controller. By invoking different REST API endpoints of the agent, specifically the *ACA-py* process, we are interacting with the agent, which handles more complex processes such as writing to the ledger, managing the wallet and interacting with other agents.

## 2. Establishing the connection, credential issue protocol[¶](#X3e0fa035311f99961151822c4c2fc78ba6b5f83)

The following sections contain a demonstration on how to establish a connection between two agents using the [Aries Cloud Agent](https://github.com/hyperledger/aries-cloudagent-python) protocol. After the connection is established, we demonstrate the credential issuing protocol.

In order to store DIDs, credential definitions and schemas, the *Hyperledger Indy Ledger* is used. We could use an existing hosted ledger but in this notebook, we are using a locally deployed ledger. To achieve this, we use the [von-network](https://github.com/bcgov/von-network) (Verifiable Organizations Network) project, which automatically sets up and deploys multiple *Indy ledger* nodes using *Docker*.

A more in-depth description of the connection establishment and credential issuing processes can be found at:

* [Connection protocol (RFC)](https://github.com/hyperledger/aries-rfcs/tree/master/features/0160-connection-protocol)
* [Issue credential protocol (RFC)](https://github.com/hyperledger/aries-rfcs/tree/master/features/0453-issue-credential-v2#choreography-diagram)

Prerequisites:

* *docker*, configured and running
* the current user added to the 'docker' group\ sudo usermod -aG docker [non-root user]
* *aries-cloudagent* installed\ pip3 install aries-cloudagent
* PlantUML installed\ pip3 install iplantuml

In [1]:

import iplantuml

### 2.1 Von-network setup[¶](#X0fba12fe116e2796f27d796782ac1ccd1866d5d)

In [2]:

!git clone https://github.com/bcgov/von-network

Cloning into 'von-network'...  
remote: Enumerating objects: 1786, done.  
remote: Counting objects: 100% (180/180), done.  
remote: Compressing objects: 100% (113/113), done.  
remote: Total 1786 (delta 77), reused 133 (delta 52), pack-reused 1606  
Receiving objects: 100% (1786/1786), 534.47 KiB | 1.56 MiB/s, done.  
Resolving deltas: 100% (1015/1015), done.

1) Run VON-Network, which in turn sets up 4 Hyperledger Indy nodes, that act as a ledger. We store the schema and the credential definition on the ledger. When running with ./manage start --logs, the web UI runs on localhost:9000, genesis file resides in http://localhost:9000/genesis.

In [5]:

%cd von-network  
!bash manage up

[Errno 2] No such file or directory: 'von-network'  
/home/klemen/aries/von-network  
Starting von\_node3\_1 ...   
Starting von\_node1\_1 ...   
Starting von\_node2\_1 ...   
Starting von\_node4\_1 ...   
Starting von\_webserver\_1 ...   
ting von\_node1\_1 ... doneWant to see the scrolling container logs? Run "./manage logs"

### 2.2 Create a DID and publish it to the ledger[¶](#X492c76f42187b55700993fdb9bff23b0d705767)

Register a DID on the ledger. Alice will act as our issuer.

In [7]:

import requests  
import json  
ledger\_url = "http://localhost:9000"  
  
def send\_request(url, data={}, req\_type="post"):  
 if req\_type == "get":  
 headers = {'content-type': 'application/json'}  
 return requests.get(url, data=json.dumps(data), headers=headers, verify=True, allow\_redirects=False)  
   
 headers = {'content-type': 'application/json'}  
 return requests.post(url, data=json.dumps(data), headers=headers, verify=True, allow\_redirects=False)  
  
  
def register\_DID\_on\_ledger(seed, role, alias):  
 d = {   
 "seed": seed,  
 "role": role,  
 "alias": alias   
 }  
 return send\_request(ledger\_url + "/register", d)  
  
  
  
resp = register\_DID\_on\_ledger("Alice000000000000000000000000001", "TRUST\_ANCHOR", "Alice")  
  
alice\_did = resp.json()["did"]  
print(f"Alice's DID: {alice\_did}")  
alice\_verkey = resp.json()["verkey"]  
print(f"Response: \n{resp.text}")

Alice's DID: PLEVLDPJQMJvPLyX3LgB6S  
Response:   
{  
 "did": "PLEVLDPJQMJvPLyX3LgB6S",  
 "seed": "Alice000000000000000000000000001",  
 "verkey": "DAwrZwgMwkTVHUQ8ZYAmuvzwprDmX8vFNXzFioxrWpCA"  
}

3) Start Alice: Run in a separate terminal, since jupyter can't handle multiple blocking processes.

aca-py start \  
--label Alice \  
-it http 0.0.0.0 8000 \  
-ot http \  
--admin 0.0.0.0 11000 \  
--admin-insecure-mode \  
--genesis-url http://localhost:9000/genesis \  
--seed Alice000000000000000000000000001 \  
--endpoint http://localhost:8000/ \  
--debug-connections \  
--auto-provision \  
--wallet-type indy \  
--wallet-name Alice \  
--wallet-key secret

* --label Alice This is the label or name that you give to your instance. It is the name that for example a Wallet App will see when you try to make a connection, or when you receive a credential.
* -it http 0.0.0.0 8000 and -ot http are the inbound and outbound transport methods that ACA-py uses to communicate to other ACA-py instances. Remember port 8000 here, you need it for endpoint.
* --admin 0.0.0.0 11000 and --admin-insecure-mode are the parameters that configure how your controller application can communicate with ACA-py. In this case, the Admin Endpoints are available on port 11000, and insecure, meaning there is no authentication required. Go ahead, open localhost:11000. You should see the Swagger docs, and you should see the provided label, in this case Alice. These are the endpoints your controller application will interact with
* --genesis-url http://localhost:9000/genesis This is the URL to the genesis file. When you create a schema and credential definition, you create transactions in the Indy ledger. To be able to create these transactions, ACA-py needs to know about the genesis transaction, this is common in blockchains and distributed ledgers.
* --seed Alice000000000000000000000000001 This is the seed value we used to register the DID. This seed value proves that you are the owner of the public DID.
* --endpoint http://localhost:8000/ This is the URL that ACA-py will send to ledger, to register where the ACA-py instance for your DID can be reached. If you have started aca-py without errors, you should be able to find this endpoint in your von-network webserver. Check localhost:9000/browse/domain.
* --debug-connections This parameter makes sure that more information about connections is being printed when we start making a connection between Alice and Bob in the next section.
* --auto-provision This parameter makes sure that ACA-py is going to create a wallet for you when it doesn’t exist. Usually you should create a wallet only once using the aca-py provision command, but that is out of scope for this blog post.
* --wallet-type indy, --wallet-name Alice and --wallet-key secret are the parameters that are used to create the wallet. In this setup, the wallet is stored in files on your system. You can find the wallets in ~/.indy\_client/wallet/. The key is required to write and read to the wallet

If you start Alice successfully, you can stop it, start it again with the same parameters, and it will just continue where it left off. If you start Alice with a different wallet-name, a new wallet will be created, so you won’t have access to all the previous data stored in the wallet. If you change the seed value here without registering it in the ledger first, it wont work.

4) Start **Bob**, who will be our holder: We can start Bob the same way we did with Alice. However, Bob is going to be our holder. Bob is not going to create a schema or a credential definition, so he does not require a public DID. In fact, if you use a wallet app, you will not have a public DID at all. Remember it costs money to register a DID? This means only the issuer is paying for the registration of its DID, not the holder.

Bob does not need a public DID, so we are not going to register a DID on the ledger. This means our command-parameters change a bit. Open another terminal and run:

Run in a separate terminal, jupyter can't handle multiple blocking processes.

aca-py start \  
 --label Bob \  
 -it http 0.0.0.0 8001 \  
 -ot http \  
 --admin 0.0.0.0 11001 \  
 --admin-insecure-mode \  
 --endpoint http://localhost:8001/ \  
 --genesis-url http://localhost:9000/genesis \  
 --debug-connections \  
 --auto-provision \  
 --wallet-local-did \  
 --wallet-type indy \  
 --wallet-name Bob \  
 --wallet-key secret

There is one new parameter: --wallet-local-did. Bob doesn’t have a public DID, but he does need a local DID. The local DID will be used for Alice to create a credential for, more on that later.

The Bob ACA-py instance also has an Admin API, you access it on localhost:11001.

You should now have two ACA-py instances running next to each other. One for Alice (the issuer), and one for Bob (the holder). With the two agents running, it is time to play the controller for both of them.

### 2.3 Connecting Alice and Bob[¶](#X914824ed4b2124cb5cdee1db1d4d6c711d040ed)

A secure connection between the two parties is established through the next sequence of messages:

In [8]:

%%plantuml  
  
@startuml  
actor "Alice(issuer)" as A  
actor "Bob(holder)" as B  
database "Ledger" as L  
  
autonumber  
A -> L : Register the DID on the Ledger  
A -> A : Create the invitation  
A --> B : Alice sends the invitation to Bob, out of band  
B -> B : Bob receives the invitation  
B -> A: Bob accepts the invitation (Alice automatically receives the connection request)  
A -> A: Alice accepts the connection request  
@enduml

Out[8]:

Alice(issuer)Alice(issuer)Bob(holder)Bob(holder)LedgerLedger1Register the DID on the Ledger2Create the invitation3Alice sends the invitation to Bob, out of band4Bob receives the invitation5Bob accepts the invitation (Alice automatically receives the connection request)6Alice accepts the connection request

In [9]:

def pp\_json(json\_thing, sort=True, indents=2):  
 if type(json\_thing) is str:  
 out = json.dumps(json.loads(json\_thing), sort\_keys=sort, indent=indents)  
 else:  
 out = json.dumps(json\_thing, sort\_keys=sort, indent=indents)  
 return out  
  
alice\_endpoint = "http://localhost:11000"  
bob\_endpoint = "http://localhost:11001"  
  
def create\_invitation():  
 d = {  
 "handshake\_protocols": ["did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/didexchange/1.0"],  
 "use\_public\_did": False  
 }  
 return send\_request(alice\_endpoint + "/out-of-band/create-invitation", d)  
  
  
resp = create\_invitation()  
invitation = resp.json()["invitation"]  
print(f"Invitation: \n{pp\_json(invitation)}")

Invitation:   
{  
 "@id": "bc7b2e7c-0bdb-4200-8a83-3a1f938e9637",  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/out-of-band/1.0/invitation",  
 "handshake\_protocols": [  
 "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/didexchange/1.0"  
 ],  
 "label": "Alice",  
 "service": [  
 {  
 "id": "#inline",  
 "recipientKeys": [  
 "did:key:z6MksGWQZxiw8u9WZNug79Z6Tw5EDXQVPJw2K5yTEAgPq1ti"  
 ],  
 "serviceEndpoint": "http://localhost:8000/",  
 "type": "did-communication"  
 }  
 ]  
}

The invitation can be passed to **Bob** as a link, or a QR code. We have it stored in invitation variable. A non-repudiable signature should be added to the invitation. Such a solution is proposed in [Aries RFC 0066: Non-Repudiable Signature for Cryptographic Envelope](https://github.com/hyperledger/aries-rfcs/blob/master/features/0066-non-repudiable-cryptographic-envelope/README.md).

In [10]:

def receive\_invitation(invitation):  
 return send\_request(bob\_endpoint + "/out-of-band/receive-invitation", invitation)  
  
resp = receive\_invitation(invitation)  
   
bob\_connection\_id = resp.json()["connection\_id"]  
print(f"Response: \n{pp\_json(resp.json())}")  
print(f"Bobs connection id: {bob\_connection\_id}")

Response:   
{  
 "accept": "manual",  
 "connection\_id": "5719b699-08a4-4e21-ba65-6ec9fd4eb3f5",  
 "created\_at": "2021-07-16 13:03:34.593283Z",  
 "invitation\_key": "DpFMyiUVoMf3St4yRabFcqXEPx8dyRgfd54XPtiNuo7L",  
 "invitation\_mode": "once",  
 "invitation\_msg\_id": "bc7b2e7c-0bdb-4200-8a83-3a1f938e9637",  
 "rfc23\_state": "invitation-received",  
 "routing\_state": "none",  
 "state": "invitation",  
 "their\_label": "Alice",  
 "their\_role": "inviter",  
 "updated\_at": "2021-07-16 13:03:34.593283Z"  
}  
Bobs connection id: 5719b699-08a4-4e21-ba65-6ec9fd4eb3f5

Now, **Bob** accepts the invitation.

In [11]:

def accept\_invitation():  
 return send\_request(f"{bob\_endpoint}/didexchange/{bob\_connection\_id}/accept-invitation", {})  
  
resp = accept\_invitation()  
print(f"Response: \n{pp\_json(resp.json())}")

Response:   
{  
 "accept": "manual",  
 "connection\_id": "5719b699-08a4-4e21-ba65-6ec9fd4eb3f5",  
 "created\_at": "2021-07-16 13:03:34.593283Z",  
 "invitation\_key": "DpFMyiUVoMf3St4yRabFcqXEPx8dyRgfd54XPtiNuo7L",  
 "invitation\_mode": "once",  
 "invitation\_msg\_id": "bc7b2e7c-0bdb-4200-8a83-3a1f938e9637",  
 "my\_did": "6mFMT5uNXX7GXbgACCy4Mb",  
 "request\_id": "85502b5c-0c42-407b-90ba-2abb9d5d2dac",  
 "rfc23\_state": "request-sent",  
 "routing\_state": "none",  
 "state": "request",  
 "their\_label": "Alice",  
 "their\_role": "inviter",  
 "updated\_at": "2021-07-16 13:03:39.222208Z"  
}

You can see that the invitation was accepted in the **Bob**'s and **Alice**'s logs. When **Bob** accepts the invitation, he sends a connection request to **Alice**. This happens automatically.

Even though **Alice** started this interaction by creating an invitation, it is actually **Bob** that sends a connection request to **Alice**, therefore **Bob** is called the requester. **Alice** is called the responder.

Alice now accepts the connection request with her connection\_id. Notice that this ID is different for **Alice** and **Bob**. Firstly, lets get **Alice**'s connection\_id.

In [12]:

def get\_connection\_id():  
 return send\_request(alice\_endpoint + "/connections", req\_type="get")  
resp = get\_connection\_id().json()  
  
# Get alice's id  
for el in resp["results"]:  
 if "their\_label" in el.keys() and el["their\_label"] == "Bob" and el["rfc23\_state"] == "request-received":  
 alice\_connection\_id = el["connection\_id"]  
 break  
  
  
print(f"Response: \n{pp\_json(resp)}")  
print(f"Alice's connection id: {alice\_connection\_id}")

Response:   
{  
 "results": [  
 {  
 "accept": "manual",  
 "connection\_id": "5a7c104b-2691-4ad6-a394-b8e276eb8606",  
 "created\_at": "2021-07-16 13:03:24.484962Z",  
 "invitation\_key": "DpFMyiUVoMf3St4yRabFcqXEPx8dyRgfd54XPtiNuo7L",  
 "invitation\_mode": "once",  
 "invitation\_msg\_id": "bc7b2e7c-0bdb-4200-8a83-3a1f938e9637",  
 "request\_id": "85502b5c-0c42-407b-90ba-2abb9d5d2dac",  
 "rfc23\_state": "request-received",  
 "routing\_state": "none",  
 "state": "request",  
 "their\_did": "6mFMT5uNXX7GXbgACCy4Mb",  
 "their\_label": "Bob",  
 "their\_role": "invitee",  
 "updated\_at": "2021-07-16 13:03:39.472770Z"  
 }  
 ]  
}  
Alice's connection id: 5a7c104b-2691-4ad6-a394-b8e276eb8606

**Alice** can now accept the connection request.

In [13]:

def accept\_connection():  
 return send\_request(f"{alice\_endpoint}/didexchange/{alice\_connection\_id}/accept-request")  
  
resp = accept\_connection()  
print(f"Response: {pp\_json(resp.json())}")

Response: {  
 "accept": "manual",  
 "connection\_id": "5a7c104b-2691-4ad6-a394-b8e276eb8606",  
 "created\_at": "2021-07-16 13:03:24.484962Z",  
 "invitation\_key": "DpFMyiUVoMf3St4yRabFcqXEPx8dyRgfd54XPtiNuo7L",  
 "invitation\_mode": "once",  
 "invitation\_msg\_id": "bc7b2e7c-0bdb-4200-8a83-3a1f938e9637",  
 "my\_did": "S74xe7bcqexiauknzc3rde",  
 "request\_id": "85502b5c-0c42-407b-90ba-2abb9d5d2dac",  
 "rfc23\_state": "response-sent",  
 "routing\_state": "none",  
 "state": "response",  
 "their\_did": "6mFMT5uNXX7GXbgACCy4Mb",  
 "their\_label": "Bob",  
 "their\_role": "invitee",  
 "updated\_at": "2021-07-16 13:03:46.562227Z"  
}

The connection is established.

### 2.4 Test the connection[¶](#X2323ee42210474de14c431a6eeaa911499661d2)

We can now send messages between the agents.

In [14]:

def send\_message(message, sender\_endpoint, sender\_conn\_id):  
 d = {  
 "content" : message  
 }  
 return send\_request(f"{sender\_endpoint}/connections/{sender\_conn\_id}/send-message", d)  
  
resp = send\_message("This is Alice", alice\_endpoint, alice\_connection\_id)  
print(resp)  
send\_message("This is Bob", bob\_endpoint, bob\_connection\_id)  
print(resp)

<Response [200]>  
<Response [200]>

### 2.5 Creating a schema and credential definition[¶](#X171dcd13af47eea71027380f4ee4cb1d958147a)

**Alice** will be issuing a credential to **Bob**. Before she can do that she needs to create a schema and a credential definition.

Creating a schema:

In [15]:

def create\_schema():  
 d = {  
 "attributes": [  
 "name",  
 "age"  
 ],  
 "schema\_name": "testtest",  
 "schema\_version": "1.0"  
 }  
 return send\_request(f"{alice\_endpoint}/schemas", d)  
  
resp = create\_schema()  
schema\_id = resp.json()["schema\_id"]  
print(f"Schema\_id: {schema\_id}")  
print(f"Response: \n{pp\_json(resp.json())}")

Schema\_id: PLEVLDPJQMJvPLyX3LgB6S:2:testtest:1.0  
Response:   
{  
 "schema": {  
 "attrNames": [  
 "age",  
 "name"  
 ],  
 "id": "PLEVLDPJQMJvPLyX3LgB6S:2:testtest:1.0",  
 "name": "testtest",  
 "seqNo": 8,  
 "ver": "1.0",  
 "version": "1.0"  
 },  
 "schema\_id": "PLEVLDPJQMJvPLyX3LgB6S:2:testtest:1.0"  
}

Now let’s create a credential definition based upon the just created schema.

In [16]:

def create\_credential\_definition(schema\_id):  
 d = {  
 "schema\_id": schema\_id,  
 "tag": "testtest"  
 }  
 return send\_request(f"{alice\_endpoint}/credential-definitions", d)  
  
resp = create\_credential\_definition(schema\_id)  
print(f"Response: \n{pp\_json(resp.json())}")

Response:   
{  
 "credential\_definition\_id": "PLEVLDPJQMJvPLyX3LgB6S:3:CL:8:testtest"  
}

### 2.6 Issuing a credential[¶](#X1f8d6fc0d77d191afeda6139a1a401bb5c407b1)

As previously stated, **Alice** will function as our credential *issuer*, while **Bob** has a role of the *holder*. The credential issuing process can be done in three ways, based on who is the initiator of the issuing process: 1) The *holder* initializes the issuing process by sending a proposal to the *issuer*. This proposal contains information about what you would like to receive from the *issuer*. Based on that, the *issuer* sends an offer to the *holder*.

2) The *issuer* sends the offer to the *holder* directly, so no proposal is needed.

3) The *holder* can directly send a request to the *issuer*, skipping the proposal and offer.

The main flow of issuing credentials is: 1) Holder sends a proposal to the issuer (issuer receives proposal)

2) Issuer sends an offer to the holder based on the proposal (holder receives offer)

3) Holder sends a request to the issuer (issuer receives request)

4) Issuer sends credential to holder (holder receives credentials)

5) Holder stores credential (holder sends acknowledge to issuer)

6) Issuer receives acknowledge

The whole process in action is shown below.

In [17]:

%%plantuml  
  
@startuml  
actor "Alice(issuer)" as A  
actor "Bob(holder)" as B  
database "Ledger" as L  
  
A -> A : Create a schema and a credential definition  
A -> L: Send credential definition to ledger  
  
autonumber   
B -> A : Send proposal  
A --> B : Send offer based on proposal  
B -> A : Send request  
A --> B : Send credential  
B --> A : Store credential and send acknowledgement  
@enduml

Out[17]:

Alice(issuer)Alice(issuer)Bob(holder)Bob(holder)LedgerLedgerCreate a schema and a credential definitionSend credential definition to ledger1Send proposal2Send offer based on proposal3Send request4Send credential5Store credential and send acknowledgement

### 1) Bob sends a proposal to Alice[¶](#Xbcda7c3769f338aa67dfa45c1bb463b3d78108b)

The connection\_id is used.

In [18]:

def send\_proposal():  
 d = {   
 "comment": "I need a credential. 1",  
 "connection\_id": bob\_connection\_id,  
 "credential\_preview": {  
 "@type": "issue-credential/2.0/credential-preview",  
 "attributes": [  
 {  
 "mime-type": "plain/text",  
 "name": "name",   
 "value": "Bob"  
 },  
 {  
 "mime-type": "plain/text",  
 "name": "age",   
 "value": "30"  
 }  
 ]  
 },  
 "filter": {  
 "dif": {},  
 "indy": {}  
 }  
 }  
 return send\_request(f"{bob\_endpoint}/issue-credential-2.0/send-proposal", d)  
  
resp = send\_proposal().json()  
bob\_cred\_ex\_id = resp["cred\_ex\_id"]  
  
print(f"Response: \n{pp\_json(resp)}")  
print(f"cred\_ex\_id: {bob\_cred\_ex\_id}")

Response:   
{  
 "auto\_issue": false,  
 "auto\_offer": false,  
 "auto\_remove": true,  
 "conn\_id": "5719b699-08a4-4e21-ba65-6ec9fd4eb3f5",  
 "created\_at": "2021-07-16 13:04:05.063164Z",  
 "cred\_ex\_id": "2bd25ad9-b650-40d6-855b-7be8bea8f773",  
 "cred\_preview": {  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/credential-preview",  
 "attributes": [  
 {  
 "mime-type": "plain/text",  
 "name": "name",  
 "value": "Bob"  
 },  
 {  
 "mime-type": "plain/text",  
 "name": "age",  
 "value": "30"  
 }  
 ]  
 },  
 "cred\_proposal": {  
 "@id": "7498225e-10ef-4999-ab12-eb7eeae18ffb",  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/propose-credential",  
 "comment": "I need a credential. 1",  
 "credential\_preview": {  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/credential-preview",  
 "attributes": [  
 {  
 "mime-type": "plain/text",  
 "name": "name",  
 "value": "Bob"  
 },  
 {  
 "mime-type": "plain/text",  
 "name": "age",  
 "value": "30"  
 }  
 ]  
 },  
 "filters~attach": [  
 {  
 "@id": "0",  
 "data": {  
 "base64": "e30="  
 },  
 "mime-type": "application/json"  
 },  
 {  
 "@id": "1",  
 "data": {  
 "base64": "e30="  
 },  
 "mime-type": "application/json"  
 }  
 ],  
 "formats": [  
 {  
 "attach\_id": "0",  
 "format": "dif/credential-manifest@v1.0"  
 },  
 {  
 "attach\_id": "1",  
 "format": "hlindy-zkp-v1.0"  
 }  
 ]  
 },  
 "initiator": "self",  
 "role": "holder",  
 "state": "proposal-sent",  
 "thread\_id": "7498225e-10ef-4999-ab12-eb7eeae18ffb",  
 "updated\_at": "2021-07-16 13:04:05.063164Z"  
}  
cred\_ex\_id: 2bd25ad9-b650-40d6-855b-7be8bea8f773

The result you get back is a Credential Exchange Record. It is a record that contains the state of the credential process. These records are stored in ACA-py and can be retrieved using the /issue-credentials-2.0/records/{id} endpoint where the id is the cred\_ex\_id in the result.

### 2) Alice responds with an offer[¶](#Xd191b8d84d1f7ca7201757ca3a32103cbef6641)

In [19]:

def get\_alice\_cred\_id():  
 resp = send\_request(f"{alice\_endpoint}/issue-credential-2.0/records", data={}, req\_type="get").json()["results"]  
 for el in resp:  
 if el["cred\_ex\_record"]["cred\_proposal"]["comment"] == "I need a credential. 1": # TODO: the corresponding proposal should be found differently  
 return el["cred\_ex\_record"]["cred\_ex\_id"]  
  
def respond\_with\_offer(alice\_cred\_id):  
 return send\_request(f"{alice\_endpoint}/issue-credential-2.0/records/{alice\_cred\_id}/send-offer")  
  
alice\_cred\_id = get\_alice\_cred\_id()  
  
  
#alice\_cred\_id = alice\_cred\_id.json()["results"][0]["credential\_proposal\_dict"]["@id"]  
resp = respond\_with\_offer(alice\_cred\_id)  
print(pp\_json(resp.json()))

{  
 "auto\_remove": true,  
 "conn\_id": "5a7c104b-2691-4ad6-a394-b8e276eb8606",  
 "created\_at": "2021-07-16 13:04:05.114797Z",  
 "cred\_ex\_id": "7f34acb4-07aa-4c1f-8daa-eb93aafd1233",  
 "cred\_offer": {  
 "@id": "6bfd4fe7-ea43-40a6-a37d-5bce84e25d15",  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/offer-credential",  
 "credential\_preview": {  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/credential-preview",  
 "attributes": [  
 {  
 "mime-type": "plain/text",  
 "name": "name",  
 "value": "Bob"  
 },  
 {  
 "mime-type": "plain/text",  
 "name": "age",  
 "value": "30"  
 }  
 ]  
 },  
 "formats": [  
 {  
 "attach\_id": "0",  
 "format": "hlindy-zkp-v1.0"  
 }  
 ],  
 "offers~attach": [  
 {  
 "@id": "0",  
 "data": {  
 "base64": ""  
 },  
 "mime-type": "application/json"  
 }  
 ],  
 "~thread": {  
 "thid": "7498225e-10ef-4999-ab12-eb7eeae18ffb"  
 }  
 },  
 "cred\_preview": {  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/credential-preview",  
 "attributes": [  
 {  
 "mime-type": "plain/text",  
 "name": "name",  
 "value": "Bob"  
 },  
 {  
 "mime-type": "plain/text",  
 "name": "age",  
 "value": "30"  
 }  
 ]  
 },  
 "cred\_proposal": {  
 "@id": "7498225e-10ef-4999-ab12-eb7eeae18ffb",  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/propose-credential",  
 "comment": "I need a credential. 1",  
 "credential\_preview": {  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/credential-preview",  
 "attributes": [  
 {  
 "mime-type": "plain/text",  
 "name": "name",  
 "value": "Bob"  
 },  
 {  
 "mime-type": "plain/text",  
 "name": "age",  
 "value": "30"  
 }  
 ]  
 },  
 "filters~attach": [  
 {  
 "@id": "0",  
 "data": {  
 "base64": "e30="  
 },  
 "mime-type": "application/json"  
 },  
 {  
 "@id": "1",  
 "data": {  
 "base64": "e30="  
 },  
 "mime-type": "application/json"  
 }  
 ],  
 "formats": [  
 {  
 "attach\_id": "0",  
 "format": "dif/credential-manifest@v1.0"  
 },  
 {  
 "attach\_id": "1",  
 "format": "hlindy-zkp-v1.0"  
 }  
 ]  
 },  
 "initiator": "external",  
 "role": "issuer",  
 "state": "offer-sent",  
 "thread\_id": "7498225e-10ef-4999-ab12-eb7eeae18ffb",  
 "trace": false,  
 "updated\_at": "2021-07-16 13:04:12.502448Z"  
}

### 3) Bob requests the credential[¶](#X51f1a3ecf7fd5b6fbb5565c8831928b13758bca)

In [20]:

def request\_credential():  
 return send\_request(f"{bob\_endpoint}/issue-credential-2.0/records/{bob\_cred\_ex\_id}/send-request", data={}, req\_type="post")  
  
resp = request\_credential()  
print(pp\_json(resp.json()))

{  
 "auto\_issue": false,  
 "auto\_offer": false,  
 "auto\_remove": true,  
 "conn\_id": "5719b699-08a4-4e21-ba65-6ec9fd4eb3f5",  
 "created\_at": "2021-07-16 13:04:05.063164Z",  
 "cred\_ex\_id": "2bd25ad9-b650-40d6-855b-7be8bea8f773",  
 "cred\_offer": {  
 "@id": "6bfd4fe7-ea43-40a6-a37d-5bce84e25d15",  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/offer-credential",  
 "credential\_preview": {  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/credential-preview",  
 "attributes": [  
 {  
 "mime-type": "plain/text",  
 "name": "name",  
 "value": "Bob"  
 },  
 {  
 "mime-type": "plain/text",  
 "name": "age",  
 "value": "30"  
 }  
 ]  
 },  
 "formats": [  
 {  
 "attach\_id": "0",  
 "format": "hlindy-zkp-v1.0"  
 }  
 ],  
 "offers~attach": [  
 {  
 "@id": "0",  
 "data": {  
 "base64": ""  
 },  
 "mime-type": "application/json"  
 }  
 ],  
 "~thread": {  
 "thid": "7498225e-10ef-4999-ab12-eb7eeae18ffb"  
 }  
 },  
 "cred\_preview": {  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/credential-preview",  
 "attributes": [  
 {  
 "mime-type": "plain/text",  
 "name": "name",  
 "value": "Bob"  
 },  
 {  
 "mime-type": "plain/text",  
 "name": "age",  
 "value": "30"  
 }  
 ]  
 },  
 "cred\_proposal": {  
 "@id": "318b0838-a6dd-4f16-b07b-a55138d72572",  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/propose-credential",  
 "credential\_preview": {  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/credential-preview",  
 "attributes": [  
 {  
 "mime-type": "plain/text",  
 "name": "name",  
 "value": "Bob"  
 },  
 {  
 "mime-type": "plain/text",  
 "name": "age",  
 "value": "30"  
 }  
 ]  
 },  
 "filters~attach": [  
 {  
 "@id": "0",  
 "data": {  
 "base64": "eyJzY2hlbWFfaWQiOiAiUExFVkxEUEpRTUp2UEx5WDNMZ0I2UzoyOnRlc3R0ZXN0OjEuMCIsICJjcmVkX2RlZl9pZCI6ICJQTEVWTERQSlFNSnZQTHlYM0xnQjZTOjM6Q0w6ODp0ZXN0dGVzdCJ9"  
 },  
 "mime-type": "application/json"  
 }  
 ],  
 "formats": [  
 {  
 "attach\_id": "0",  
 "format": "hlindy-zkp-v1.0"  
 }  
 ]  
 },  
 "initiator": "self",  
 "role": "holder",  
 "state": "request-sent",  
 "thread\_id": "7498225e-10ef-4999-ab12-eb7eeae18ffb",  
 "updated\_at": "2021-07-16 13:04:15.717561Z"  
}

### 4) Alice issues the credential[¶](#Xf6300455ff7d3832e9dd111b89b50cac7b4a5f9)

In [21]:

def issue\_credential():  
 d = {"comment": "This is the credential."}  
 return send\_request(f"{alice\_endpoint}/issue-credential-2.0/records/{alice\_cred\_id}/issue", data=d, req\_type="post")  
  
resp = issue\_credential()  
print(pp\_json(resp.json()))

{  
 "cred\_ex\_record": {  
 "auto\_remove": true,  
 "conn\_id": "5a7c104b-2691-4ad6-a394-b8e276eb8606",  
 "created\_at": "2021-07-16 13:04:05.114797Z",  
 "cred\_ex\_id": "7f34acb4-07aa-4c1f-8daa-eb93aafd1233",  
 "cred\_issue": {  
 "@id": "4bfef527-b9de-4983-8618-b32d35486012",  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/issue-credential",  
 "comment": "This is the credential.",  
 "credentials~attach": [  
 {  
 "@id": "0",  
 "data": {  
 "base64": ""  
 },  
 "mime-type": "application/json"  
 }  
 ],  
 "formats": [  
 {  
 "attach\_id": "0",  
 "format": "hlindy-zkp-v1.0"  
 }  
 ]  
 },  
 "cred\_offer": {  
 "@id": "6bfd4fe7-ea43-40a6-a37d-5bce84e25d15",  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/offer-credential",  
 "credential\_preview": {  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/credential-preview",  
 "attributes": [  
 {  
 "mime-type": "plain/text",  
 "name": "name",  
 "value": "Bob"  
 },  
 {  
 "mime-type": "plain/text",  
 "name": "age",  
 "value": "30"  
 }  
 ]  
 },  
 "formats": [  
 {  
 "attach\_id": "0",  
 "format": "hlindy-zkp-v1.0"  
 }  
 ],  
 "offers~attach": [  
 {  
 "@id": "0",  
 "data": {  
 "base64": ""  
 },  
 "mime-type": "application/json"  
 }  
 ],  
 "~thread": {  
 "thid": "7498225e-10ef-4999-ab12-eb7eeae18ffb"  
 }  
 },  
 "cred\_preview": {  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/credential-preview",  
 "attributes": [  
 {  
 "mime-type": "plain/text",  
 "name": "name",  
 "value": "Bob"  
 },  
 {  
 "mime-type": "plain/text",  
 "name": "age",  
 "value": "30"  
 }  
 ]  
 },  
 "cred\_proposal": {  
 "@id": "7498225e-10ef-4999-ab12-eb7eeae18ffb",  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/propose-credential",  
 "comment": "I need a credential. 1",  
 "credential\_preview": {  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/credential-preview",  
 "attributes": [  
 {  
 "mime-type": "plain/text",  
 "name": "name",  
 "value": "Bob"  
 },  
 {  
 "mime-type": "plain/text",  
 "name": "age",  
 "value": "30"  
 }  
 ]  
 },  
 "filters~attach": [  
 {  
 "@id": "0",  
 "data": {  
 "base64": "e30="  
 },  
 "mime-type": "application/json"  
 },  
 {  
 "@id": "1",  
 "data": {  
 "base64": "e30="  
 },  
 "mime-type": "application/json"  
 }  
 ],  
 "formats": [  
 {  
 "attach\_id": "0",  
 "format": "dif/credential-manifest@v1.0"  
 },  
 {  
 "attach\_id": "1",  
 "format": "hlindy-zkp-v1.0"  
 }  
 ]  
 },  
 "cred\_request": {  
 "@id": "7c0d8bbf-dc14-4935-94e4-80d417a21cda",  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/request-credential",  
 "formats": [  
 {  
 "attach\_id": "0",  
 "format": "hlindy-zkp-v1.0"  
 }  
 ],  
 "requests~attach": [  
 {  
 "@id": "0",  
 "data": {  
 "base64": ""  
 },  
 "mime-type": "application/json"  
 }  
 ],  
 "~thread": {  
 "thid": "7498225e-10ef-4999-ab12-eb7eeae18ffb"  
 }  
 },  
 "initiator": "external",  
 "role": "issuer",  
 "state": "credential-issued",  
 "thread\_id": "7498225e-10ef-4999-ab12-eb7eeae18ffb",  
 "trace": false,  
 "updated\_at": "2021-07-16 13:04:18.831329Z"  
 },  
 "dif": null,  
 "indy": {  
 "created\_at": "2021-07-16 13:04:18.838469Z",  
 "cred\_ex\_id": "7f34acb4-07aa-4c1f-8daa-eb93aafd1233",  
 "cred\_ex\_indy\_id": "b017bfee-9064-4a2e-9cd2-317559edc66b",  
 "updated\_at": "2021-07-16 13:04:18.838469Z"  
 }  
}

### 5) Bob stores the received credential[¶](#Xb675e131edd985570c2e4d1366e9d2916be1be1)

In [22]:

def store\_credential():  
 return send\_request(f"{bob\_endpoint}/issue-credential-2.0/records/{bob\_cred\_ex\_id}/store")  
  
resp = store\_credential()  
print(pp\_json(resp.json()))

{  
 "cred\_ex\_record": {  
 "auto\_issue": false,  
 "auto\_offer": false,  
 "auto\_remove": true,  
 "conn\_id": "5719b699-08a4-4e21-ba65-6ec9fd4eb3f5",  
 "created\_at": "2021-07-16 13:04:05.063164Z",  
 "cred\_ex\_id": "2bd25ad9-b650-40d6-855b-7be8bea8f773",  
 "cred\_id\_stored": "0258b60d-bcd9-469e-9d52-a5c24b260bf6",  
 "cred\_issue": {  
 "@id": "4bfef527-b9de-4983-8618-b32d35486012",  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/issue-credential",  
 "comment": "This is the credential.",  
 "credentials~attach": [  
 {  
 "@id": "0",  
 "data": {  
 "base64": ""  
 },  
 "mime-type": "application/json"  
 }  
 ],  
 "formats": [  
 {  
 "attach\_id": "0",  
 "format": "hlindy-zkp-v1.0"  
 }  
 ],  
 "~thread": {  
 "thid": "7498225e-10ef-4999-ab12-eb7eeae18ffb"  
 }  
 },  
 "cred\_offer": {  
 "@id": "6bfd4fe7-ea43-40a6-a37d-5bce84e25d15",  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/offer-credential",  
 "credential\_preview": {  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/credential-preview",  
 "attributes": [  
 {  
 "mime-type": "plain/text",  
 "name": "name",  
 "value": "Bob"  
 },  
 {  
 "mime-type": "plain/text",  
 "name": "age",  
 "value": "30"  
 }  
 ]  
 },  
 "formats": [  
 {  
 "attach\_id": "0",  
 "format": "hlindy-zkp-v1.0"  
 }  
 ],  
 "offers~attach": [  
 {  
 "@id": "0",  
 "data": {  
 "base64": ""  
 },  
 "mime-type": "application/json"  
 }  
 ],  
 "~thread": {  
 "thid": "7498225e-10ef-4999-ab12-eb7eeae18ffb"  
 }  
 },  
 "cred\_preview": {  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/credential-preview",  
 "attributes": [  
 {  
 "mime-type": "plain/text",  
 "name": "name",  
 "value": "Bob"  
 },  
 {  
 "mime-type": "plain/text",  
 "name": "age",  
 "value": "30"  
 }  
 ]  
 },  
 "cred\_proposal": {  
 "@id": "318b0838-a6dd-4f16-b07b-a55138d72572",  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/propose-credential",  
 "credential\_preview": {  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/credential-preview",  
 "attributes": [  
 {  
 "mime-type": "plain/text",  
 "name": "name",  
 "value": "Bob"  
 },  
 {  
 "mime-type": "plain/text",  
 "name": "age",  
 "value": "30"  
 }  
 ]  
 },  
 "filters~attach": [  
 {  
 "@id": "0",  
 "data": {  
 "base64": "eyJzY2hlbWFfaWQiOiAiUExFVkxEUEpRTUp2UEx5WDNMZ0I2UzoyOnRlc3R0ZXN0OjEuMCIsICJjcmVkX2RlZl9pZCI6ICJQTEVWTERQSlFNSnZQTHlYM0xnQjZTOjM6Q0w6ODp0ZXN0dGVzdCJ9"  
 },  
 "mime-type": "application/json"  
 }  
 ],  
 "formats": [  
 {  
 "attach\_id": "0",  
 "format": "hlindy-zkp-v1.0"  
 }  
 ]  
 },  
 "initiator": "self",  
 "role": "holder",  
 "state": "done",  
 "thread\_id": "7498225e-10ef-4999-ab12-eb7eeae18ffb",  
 "updated\_at": "2021-07-16 13:04:23.383150Z"  
 },  
 "dif": null,  
 "indy": null  
}

The holder can retrieve the stored credential by calling /credentials endpoint:

In [23]:

def retrieve\_credential():  
 return send\_request(f"{bob\_endpoint}/credentials", req\_type="get", data={})  
  
resp = retrieve\_credential()  
print(pp\_json(resp.json()))

{  
 "results": [  
 {  
 "attrs": {  
 "age": "30",  
 "name": "Bob"  
 },  
 "cred\_def\_id": "PLEVLDPJQMJvPLyX3LgB6S:3:CL:8:testtest",  
 "cred\_rev\_id": null,  
 "referent": "0258b60d-bcd9-469e-9d52-a5c24b260bf6",  
 "rev\_reg\_id": null,  
 "schema\_id": "PLEVLDPJQMJvPLyX3LgB6S:2:testtest:1.0"  
 }  
 ]  
}

### 2.7 Skipping the proposal, the *issuer* sends the offer to the *holder* directly[¶](#X5d0ef314555603a4ef0ffe6abc9a01fddbef76b)

The *issuer* can directly send the offer to the *holder*, without the need for the proposal. This is done through the /issue-credential-2.0/send-offer endpoint.

In [24]:

def send\_offer():  
 d = {  
 "comment": "I can send you this credential",  
 "connection\_id": f"{alice\_connection\_id}",   
 "credential\_preview": {  
 "@type": "issue-credential/2.0/credential-preview",   
 "attributes": [  
 {  
 "name": "name",   
 "value": "Bob"  
 },{  
 "name": "age",  
 "value": "30"  
 }]  
 },  
 "filter": {  
 "dif": {},   
 "indy": {}  
 }  
 }  
 return send\_request(f"{alice\_endpoint}/issue-credential-2.0/send-offer", data=d, req\_type="post")  
   
resp = send\_offer()  
print(pp\_json(resp.json()))

{  
 "auto\_offer": false,  
 "conn\_id": "5a7c104b-2691-4ad6-a394-b8e276eb8606",  
 "created\_at": "2021-07-16 13:04:31.573511Z",  
 "cred\_ex\_id": "563dae93-5c21-4fbd-9ccb-90267d4e0a08",  
 "cred\_offer": {  
 "@id": "74702722-4f0e-498e-813d-49b98916f617",  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/offer-credential",  
 "comment": "I can send you this credential",  
 "credential\_preview": {  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/credential-preview",  
 "attributes": [  
 {  
 "name": "name",  
 "value": "Bob"  
 },  
 {  
 "name": "age",  
 "value": "30"  
 }  
 ]  
 },  
 "formats": [  
 {  
 "attach\_id": "0",  
 "format": "hlindy-zkp-v1.0"  
 }  
 ],  
 "offers~attach": [  
 {  
 "@id": "0",  
 "data": {  
 "base64": ""  
 },  
 "mime-type": "application/json"  
 }  
 ],  
 "~thread": {}  
 },  
 "cred\_preview": {  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/credential-preview",  
 "attributes": [  
 {  
 "name": "name",  
 "value": "Bob"  
 },  
 {  
 "name": "age",  
 "value": "30"  
 }  
 ]  
 },  
 "cred\_proposal": {  
 "@id": "76b51210-10b6-4b68-8f5a-d4c33553b18d",  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/propose-credential",  
 "comment": "I can send you this credential",  
 "credential\_preview": {  
 "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-credential/2.0/credential-preview",  
 "attributes": [  
 {  
 "name": "name",  
 "value": "Bob"  
 },  
 {  
 "name": "age",  
 "value": "30"  
 }  
 ]  
 },  
 "filters~attach": [  
 {  
 "@id": "dif",  
 "data": {  
 "base64": "e30="  
 },  
 "mime-type": "application/json"  
 },  
 {  
 "@id": "indy",  
 "data": {  
 "base64": "e30="  
 },  
 "mime-type": "application/json"  
 }  
 ],  
 "formats": [  
 {  
 "attach\_id": "dif",  
 "format": "dif/credential-manifest@v1.0"  
 },  
 {  
 "attach\_id": "indy",  
 "format": "hlindy-zkp-v1.0"  
 }  
 ]  
 },  
 "initiator": "self",  
 "role": "issuer",  
 "state": "offer-sent",  
 "thread\_id": "74702722-4f0e-498e-813d-49b98916f617",  
 "updated\_at": "2021-07-16 13:04:31.573511Z"  
}

After this offer, the flow continues with the holder responding with a request, as described above.