IDENTIFYING AND TRACKING PHYSICAL OBJECTS WITH HYPERLEDGER DECENTRALIZED APPLICATIONS

A Degree Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

David Chicano Valenzuela

In partial fulfillment

of the requirements for the degree in

Telecommunications Technologies and Services ENGINEERING

Advisors:

Matevž Pustišek (Univerza v Ljubljani)

Jose Luis Muñoz Tapia (UPC ETSETB)

Ljubljana , June 2022

Abstract

The passing of time has made clear the trend of decentralization. Distributed ledger technologies like Blockchain have brought a full range of new possibilities to improve, in this case, trust and identity management on the internet as Self Sovereign Identity (SSI) does. This thesis has analyzed the Blockchain impact on IoT with the Linux Foundation Project called Hyperledger, concretely, with its Identity Stack solution. Even though it is a really unmatured project, it has some useful tools to start understanding and testing the capabilities of this technology. To make things clear, the project consists of a practical scenario simulation of communication and verified credentials sending between a Raspberry Pi with an RFID sensor, which will be tracking an object's state of delivery, and a graphical Java Application. Everything through the Hyperledger SSI Stack, formed by a public Indy network instance and Aries agents.

Resum

El pas del temps ha deixat clara la tendència a la descentralització. Les Distributed Ledger Technologies, com el Blockchain han aportat un ventall complet de noves possibilitats per millorar, en aquest cas, la gestió de la confiança i la identitat a Internet, com ho fa la Self Sovereign Identity (SSI). Aquesta tesi ha analitzat l'impacte del Blockchain per a IoT amb el projecte Linux Foundation anomenat Hyperledger, concretament, amb la seva solució Identity Stack. Tot i que és un projecte realment sense madurar, té algunes eines útils per començar a entendre i provar les capacitats d'aquesta tecnologia. Per aclarir les coses, el projecte consisteix en una simulació d'escenari pràctic de communicació i d'enviament credencials verificades entre una Raspberry Pi amb un sensor RFID, que farà el seguiment de l'estat de lliurament d'un objecte, i una aplicació gràfica Java. Tot a través de l'Hyperledger SSI Stack, format per una instància pública d'Indy Ledger i agents Aries.

Resumen

El paso del tiempo ha dejado clara la tendencia a la descentralización. Las Distributed Ledger Technologies, como el Blockchain han aportado un abanico completo de nuevas posibilidades para mejorar, en este caso, la gestión de la confianza y la identidad en Internet, como lo hace la Self Sovereign Identity (SSI) . Esta tesis ha analizado el impacto del Blockchain para IoT con el proyecto Linux Foundation llamado Hyperledger, concretamente, con su solución Identity Stack. Aunque es un proyecto realmente sin madurar, tiene algunas herramientas útiles para empezar a entender y probar las capacidades de esta tecnología. Para aclarar las cosas, el proyecto consiste en una simulación de escenario práctico de comunicación y de envío credenciales verificadas entre una Raspberry Pi con un sensor RFID, que realizará el seguimiento del estado de entrega de un objeto, y una aplicación gráfica Java. Todo a través del Hyperledger SSI Stack, formado por una instancia pública de Indy Ledger y agentes Aries.

Revision history and approval record

Revision	Date	Purpose
0	12/04/2022	Document creation
1	18/06/2022	Document revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
David Chicano Valenzuela	david.chicano@estudiantat.upc.edu
Matevž Pustišek	Matevz.Pustisek@fe.uni-lj.si
Jose Luis Muñoz Tapia	jose.luis.munoz@upc.edu

Written by:		Reviewed and approved by:	
Date	18/06/2022	Date	20/06/2022
Name	David Chicano Valenzuela	Name	Jose Luis Muñoz Tapia
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

1. Introduction

1.1. Statement of purpose

The purpose of this project is to analyze decentralized applications for IoT based on Hyperledger technologies. In particular, it will explore and demonstrate options for identifying and tracking physical objects with Hyperledger decentralized applications.

1.2. Requirements and specifications

Project requirements:

- Presentation of the background research on tools and concepts:
- Explanation of the different identifying options/standards and their problematics.
- Explanation of the SSI solution.
- General explanation of the different blockchains and a specific one of the Hyperledger private network.
- Development of a simple decentralized application with Hyperledger.
- Tracking of objects with IoT based on the blockchain.
- Identification, authentication, and authorization of the IoT devices on the blockchain.

Project specifications:

- An implementation of the decentralized application with Hyperledger capable of:
    - IoT SSI management.
    - Object tracking based on the live status changing.

1.3. Work plan, Milestones and Gantt diagram

1.3.1. Workplan

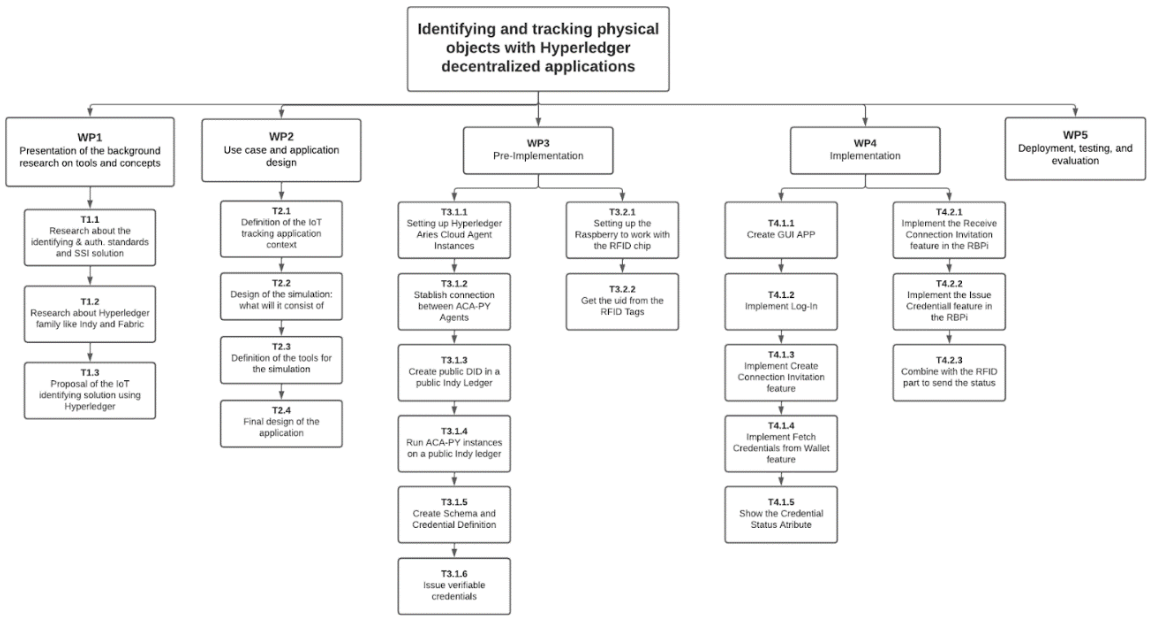

Identifying and tracking physical objects with Hyperledger decentralized applications
WP1 Presentation of the background research on tools and concepts
T1.1 Research about the identifying & auth. standards and SSI solution
T1.2 Research about Hyperledger family like Indy and Fabric
T1.3 Proposal of the IoT identifying solution using Hyperledger
WP2 Use case and application design
T2.1 Definition of the IoT tracking application context
T2.2 Design of the simulation: what will it consist of
T2.3 Definition of the tools for the simulation
T2.4 Final design of the application
WP3 Pre-Implementation
T3.1.1 Setting up Hyperledger Aries Cloud Agent Instances
T3.1.2 Stablish connection between ACA-PY Agents
T3.1.3 Create public DID in a public Indy Ledger
T3.1.4 Run ACA-PY instances on a public Indy ledger
T3.1.5 Create Schema and Credential Definition
T3.1.6 Issue verifiable credentials
T3.2.1 Setting up the Raspberry to work with the RFID chip
T3.2.2 Get the uid from the RFID Tags
WP4 Implementation
T4.1.1 Create GUI APP
T4.1.2 Implement Log-In
T4.1.3 Implement Create Connection Invitation feature
T4.1.4 Implement Fetch Credentials from Wallet feature
T4.1.5 Show the Credential Status Atribute
T4.2.1 Implement the Receive Connection Invitation feature in the RBPi
T4.2.2 Implement the Issue Credentiall feature in the RBPi
T4.2.3 Combine with the RFID part to send the status
WP5 Deployment, testing, and evaluation

Figure 1 Work Plan Diagram

Project: Presentation of the background research on tools and concepts	WP ref: 1
Major constituent: Software	Sheet 1 of 5
Short description: Getting knowledge about the problematics and possible solutions of the topic.	Start event: 14/02/2022 End event: 25/03/2022
T1.1 Research about the identifying & auth. standards and SSI solution. T1.2 Research about Hyperledger family like Indy and Fabric. T1.3 Proposal of the IoT identifying solution using Hyperldeger.	

Project: Use case and application design	WP ref: 2
Major constituent: Software	Sheet 2 of 5
Short description: Structuring how the project will be carried out and what will be about.	Start event: 25/03/2022 End event: 29/04/2022
T2.1 Definition of the IoT tracking application context. T2.2 Design of the simulation: what will it consist of. T2.3 Definition of the tools for the simulation. T2.4 Final design of the application.	

Project: Pre-Implementation	WP ref: 3
Major constituent: Software and programming	Sheet 3 of 5
Short description: Building the environment and setting everything up.	Start event: 25/03/2022 End event: 30/05/2022
T3.1.1 Setting up Hyperledger Aries Cloud Agent Instances T3.1.2 Stablish a connection between ACA-PY Agents T3.1.3 Create public DID in a public Indy Ledger T3.1.4 Run ACA-PY instances on a public Indy ledger T3.1.5 Create Schema and Credential Definition T3.1.6 Issue verifiable credentials T3.2.1 Setting up the Raspberry to work with the RFID chip T3.2.2 Get the uid from the RFID Tags	

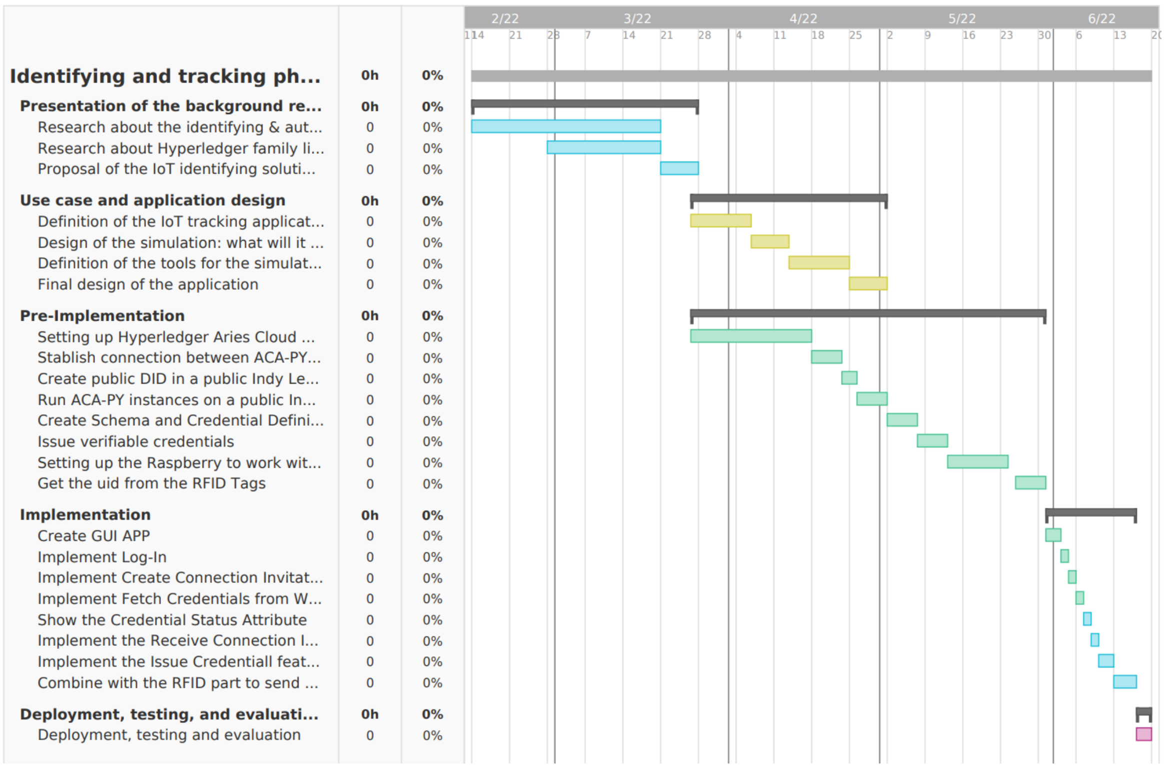
Project: Implementation	WP ref: 4
Major constituent: Software and programming	Sheet 4 of 5
Short description: Implementing the use of case.	Start event: 30/03/2022 End event: 15/05/2022
T4.1.1 Create GUI APP T4.1.2 Implement Log-In T4.1.3 Implement Create Connection Invitation feature T4.1.4 Implement Fetch Credentials from Wallet feature T4.1.5 Show the Credential Status Attribute T4.2.1 Implement the Receive Connection Invitation feature in the RBPi T4.2.2 Implement the Issue Credentiall feature in the RBPi T4.2.3 Combine with the RFID part to send the status	

Project: Deployment, testing, and evaluation	WP ref: 5
Major constituent: Evaluation	Sheet 5 of 5
Short description: Test the final version and get conclusions.	Start event: 15/05/2022 End event: 21/06/2022

1.3.2. Milestones

WP#	Task#	Short title	Milestone	Date
1	3	Presentation of the background research on tools and concepts	HOla	14/02/2022
2	4	Use case and application design	Definition of tools	25/03/2022
3	8	Pre-Implementation	Set-up of tools	25/03/2022
4	8	Implementation	Definitive simulation program	30/03/2022
5	1	Deployment, testing, and evaluation	Sending credentials from sensor to app	15/05/2022

1.3.3. Gantt diagram



Identifying and tracking ph...	**0h**	**0%**
Presentation of the background re...	**0h**	**0%**
Research about the identifying & aut...	0	0%
Research about Hyperledger family li...	0	0%
Proposal of the IoT identifying soluti...	0	0%
Use case and application design	**0h**	**0%**
Definition of the IoT tracking applicat...	0	0%
Design of the simulation: what will it ...	0	0%
Definition of the tools for the simulat...	0	0%
Final design of the application	0	0%
Pre-Implementation	**0h**	**0%**
Setting up Hyperledger Aries Cloud ...	0	0%
Stablish connection between ACA-PY...	0	0%
Create public DID in a public Indy Le...	0	0%
Run ACA-PY instances on a public In...	0	0%
Create Schema and Credential Defini...	0	0%
Issue verifiable credentials	0	0%
Setting up the Raspberry to work wit...	0	0%
Get the uid from the RFID Tags	0	0%
Implementation	**0h**	**0%**
Create GUI APP	0	0%
Implement Log-In	0	0%
Implement Create Connection Invitat...	0	0%
Implement Fetch Credentials from W...	0	0%
Show the Credential Status Attribute	0	0%
Implement the Receive Connection I...	0	0%
Implement the Issue Credentiall feat...	0	0%
Combine with the RFID part to send ...	0	0%
Deployment, testing, and evaluati...	**0h**	**0%**
Deployment, testing and evaluation	0	0%

Figure 2 Gantt Diagram of the project

1.4. <u>Description of the deviations from the initial plan and incidences that may have occurred</u>

First, the comprehension of the topics related to SSI was completely new for me and has been a hard task to understand and achieve the necessary knowledge as a developer.

I have been stuck in the definition of the tools for the simulation and pre-implementation part for so long. It has been tough for me because it is a very new project and there is not a lot of documentation, and the only one available is not clear enough. Also, there are many problems with the compatibility of the versions between libraries, utilities, and ledgers…

Finally, I have not been able to find a solution to the communication from the IoT sensor to the App because I would have to create some public endpoints to create an external connection to my computer, but I do not have permission to access the network router.

2. State of the art of the technology used or applied in this thesis:

With the accelerated rise of the technology industry, the need to increase, optimize, and reduce costs is clear. There is a growing demand for improved computing power, storage, infrastructure, protocols, and code.

With the advent of personal computers and private networks, similar computational capabilities were now housed both on the clients, as well as the servers. This, in part, gave rise to the "client-server" architecture, which supported the development of relational database systems. Massive data sets, which are housed on mainframes, could move onto a distributed architecture. This data could replicate from server to server, and subsets of the data could be accessed and processed on clients, and then, synced back to the server.

Over time, Internet and cloud computing architectures enabled global access from a variety of computing devices; whereas mainframes were largely designed to address the needs of large corporations and governments. Even though this "cloud architecture" is decentralized in terms of hardware, it has given rise to application-level centralization (e.g. Facebook, Twitter, Google, etc.).

The transition from centralized to decentralized computing, storage, and processing is already happening. These architectures and systems are aiming to give explicit control of digital assets to end-users and remove the need to trust any third-party servers and infrastructure.

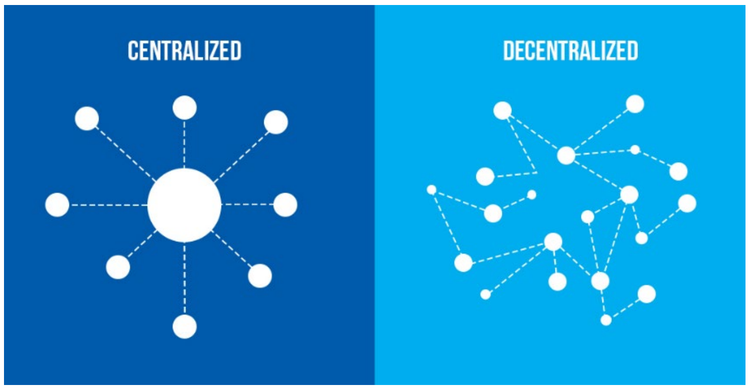


Figure 3 Centralized vs Decentralized

2.1. Distributed Ledger Technologies

A distributed ledger is a type of data structure that resides across multiple computer devices, generally spread across locations or regions.

While distributed ledger technology (DLT) existed before Bitcoin, the Bitcoin blockchain marked a before and after. DLT is all about the idea of a "decentralized" network against the conventional "centralized" mechanism, and it is deemed to have far-reaching implications on sectors and entities that have long relied upon a trusted third party.

In summary, distributed ledger technology generally consists of three basic components:

- A data model that captures the current state of the ledger
- A language of transactions that changes the ledger state
- A protocol used to build consensus among participants around which transactions will be accepted, and in what order, by the ledger.

Blockchain is a specific form or subset of distributed ledger technologies, which constructs a chronological chain of blocks, hence the name 'block-chain'.

2.2. Blockchain

A blockchain is a peer-to-peer distributed ledger forged by consensus, combined with a system for "smart contracts" and other assistive technologies. Together these can be used to build a new generation of transactional applications that establishes trust, accountability, and transparency at their core, while streamlining business processes and legal constraints.

Smart contracts are simply computer programs that execute predefined actions when certain conditions within the system are met.

A block refers to a set of transactions that are bundled together and added to the chain at the same time.

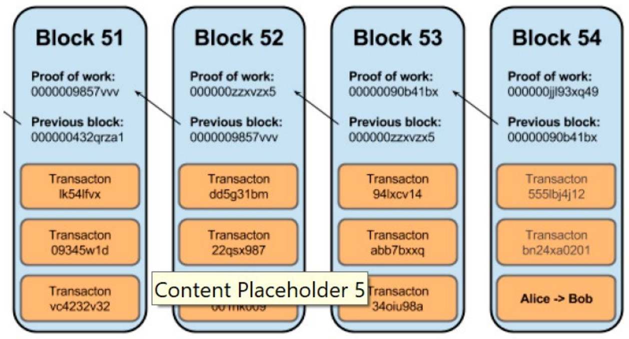


Figure 4 Block Diagram

Each block is timestamped, with each new block referring to the previous block. Combined with cryptographic hashes, this timestamped chain of blocks provides an immutable record of all transactions in the network, from the very first (or genesis) block.

Consensus refers to a system of ensuring that parties agree to a certain state of the system as the true state. Is a process whereby the computers that are part of the network synchronize the data on the blockchain. Consensus in the network refers to the process of achieving agreement among the network participants as to the correct state of data on the system. Consensus leads to all nodes sharing the exact same data. A consensus algorithm, hence, does two things: it ensures that the data on the ledger is the same for all the nodes in the network, and, in turn, prevents malicious actors from manipulating the data. The consensus algorithm varies with different blockchain implementations.

There are several consensus mechanisms or algorithms like Proof of Work, Proof of Stake, Proof of Elapsed Time, Simplified Byzantine Fault Tolerance, Practical Byzantine Fault Tolerance (PBFT).

Also, the immutability of the data which sits on the blockchain is perhaps the most powerful and convincing reason to deploy blockchain-based solutions for a variety of socio-economic processes which are currently recorded on centralized servers. Once a transaction is written onto the blockchain, no one can change it, or, at least, it would be extremely difficult to change it.

Transactions are the record of an event, cryptographically secured with a digital signature, that is verified, ordered, and bundled together into blocks, from the transactions in the blockchain. In the Bitcoin blockchain, transactions involve the transfer of bitcoins, while in other blockchains, transactions may involve the transfer of any asset or a record of some service being rendered. Furthermore, a smart contract within the blockchain may allow the automatic execution of transactions upon meeting predefined criteria.

2.2.1. Types of Blockchains

A blockchain can be both permissionless (like Bitcoin or Ethereum) or permissioned (like the different Hyperledger blockchain frameworks). A permissionless blockchain is also known as a public blockchain, because anyone can join the network. A permissioned blockchain, or private blockchain, requires pre-verification of the participating parties within the network, and these parties are usually known to each other.

2.3. Self Sovereign Identity (SSI)

SSI is the idea that you control your own data, you control when and how it is provided to others, and when it is shared, it is done so in a trusted way. With SSI, there is no central authority holding your data that passes it on to others upon request. And because of the underlying cryptography and blockchain technology, SSI means that you can present claims about your identity and others can verify it with cryptographic certainty.

It's difficult to establish trust in the identity contact and what they say. The basic mechanism for knowing who you are on the Internet is the user ID and password combination.

The use of common identifiers on so many different sites creates what is known as a correlation problem. Correlation in this context means associating without consent information about a single identity across multiple systems. The proliferation of this kind of correlation on the Internet, driven primarily by advertising, has resulted in a massive loss of privacy for Internet users (basically, everyone).

Identity-related data is currently of particularly high value and so large data repositories of identity data are favorite targets for hackers. This includes not only user ID and password data to enable unauthorized access to accounts, but also all of the other information we use to "prove" our identity online such as name, email address, government ID and so on.

In SSI, entities are identified by decentralized identifiers (DID). The trust and management of DIDs are assured by blockchain technology, and DIDs can be decoupled from centralized registries, identity providers, and certificate authorities.

The same framework can be adapted for non-person subjects and thus for IoT.

2.3.1. Verifiable Credentials & DID

A credential is (formally) an attestation of qualification, competence, or authority issued to an entity by a third party with a relevant or de facto authority or assumed competence to do so.

For identity, verifiable credentials are digital, cryptographically-protected data from authorities that you can use to prove you are you.

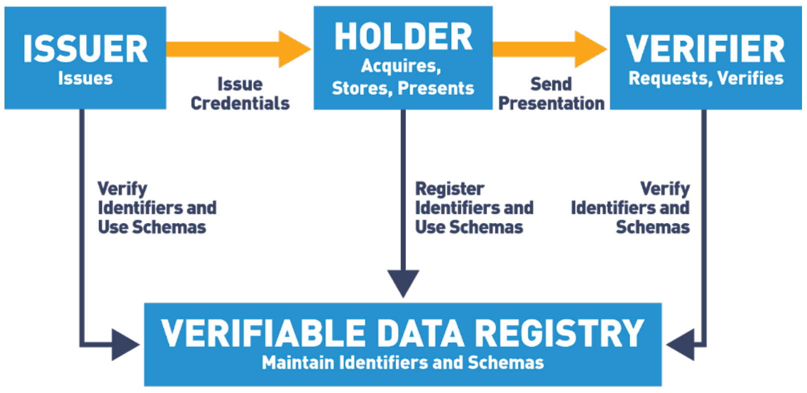

ISSUER
Issues
Issue Credentials
HOLDER
Acquires, Stores, Presents
Send Presentation
VERIFIER
Requests, Verifies
Verify Identifiers and Use Schemas
Register Identifiers and Use Schemas
Verify Identifiers and Schemas
VERIFIABLE DATA REGISTRY
Maintain Identifiers and Schemas

Figure 5 SSI flow diagram

That means that the problem of whether the verifier trusts the holder more or less goes away. The holder cannot forge the cryptography protecting the verifiable credentials and so the data acquired can be accepted without further concern.

However, there does remain a challenge with a verifiable credential—and it is actually the same thing with a paper credential: Do you trust the issuer and the process by which the issuer decided to issue the credential?

A credential is made up of a set of individual claims. For example, in the case of a driver's license, the claims would be a person's name, address, height, weight, hair colour, driver's license number and so on.

In all uses of a verifiable credential, what is issued is a credential. However, in some implementations, when a credential is presented the credential is proven, while in others (including in Hyperledger Indy), the claims within the credential are proven individually.

Another pair of terms that might seem to be used interchangeably are proof and presentation. Proof is evidence of the claim (for example, a birth certificate, passport or driver's license, or in the case of a business, incorporation papers).

Instead of typing in your name, address and government ID, you provide a presentation of that information from verifiable credentials issued to you by an authority trusted by the verifier.

- Your verifiable credentials are issued to you, stored in your digital wallet, and you decide when and where you want to use them.
- Verifiable presentation data is proven without needing to call back to the issuer.
- Verifiable credentials go beyond identity to enable the digitization of almost any paper-based verification process.

While verifiable credentials are an important component of SSI, the decentralized identifier (DID) is a key enabler for verifiable credentials. DIDs are a new type of identifier that is in the process of becoming a World Wide Web Consortium (W3C) standard. As we discussed, the verifiable credential model requires a decentralized ecosystem to work. Such an ecosystem is brought about with DIDs and agents.

DIDs are a special kind of identifier that are created by their owner, independent of any central authority. Per the DID specification, a DID looks like the following and is similar to an HTTP address but used differently.

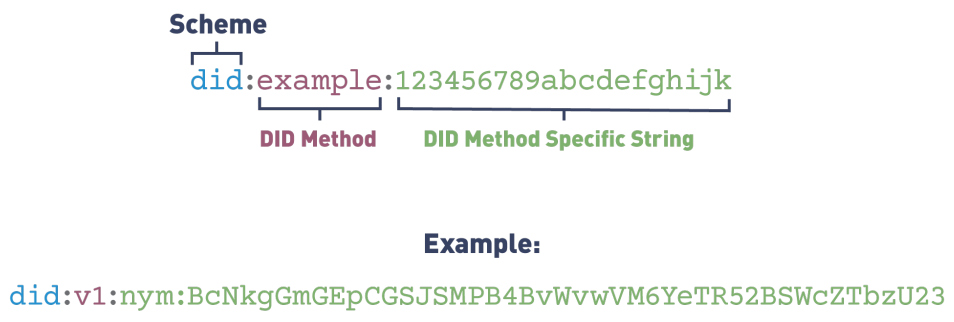

Scheme
did:example:123456789abcdefghijk
DID Method
DID Method Specific String
Example:
did:v1:nym:BcNkgGmGEpCGSJSMPB4BvWvwVM6YeTR52BSWcZTbzU23

Figure 6 DID

DIDs are:

- A new type of uniform resource location (URL).
- Created by anyone at any time.
- Globally unique.
- Highly available.
- Cryptographically verifiable.

Anything can be a DID subject: person, group, organization, a material thing, digital thing, logical thing, etc.

When we pass a valid DID to a piece of software called a DID Resolver, it works like a browser given a URL, resolving the identifier (URL or DID) and returning a document.

A DID can be created by anyone, not just a central service such as a Certificate Authority (CA). People, organizations and things can all create, publish and share DIDs. Further, at any time, the controller of the DID (usually the creator) can update the DID.

Like the URLs that we are familiar with, DIDs can be resolved (often resolved by reading from a blockchain). When we pass a valid DID to a piece of software called a DID Resolver, it works like a browser given a URL, resolving the DID and returning a document. However, instead of returning a web page, a DID Resolver returns a DID Document (DIDDoc), a JSON document whose format is defined in the DID specification.

A DIDDoc contains (usually) public keys whose private keys are held by the entity that controls the DID, and (usually) service endpoints that enable communication with that entity.

This means that with a DID, you can:

- Resolve a DID to get a DIDDoc.
- Within the DIDDoc, find a public key and an endpoint for the entity that controls the DID.
- Use the endpoint to send a message to that entity.
- In the message, ask them for proof they have the private key related to the public key.
- Since often a public key in the DIDDoc is used to encrypt the message sent, just being able to respond to the message is “proof” of control of the DID. Without the private key, they could not have decrypted the message.
- What the sent message might ask is that the controller of the DID provide a verifiable credential presentation.

- Receive back the proof.
- Verify that proof.

Just as today when you first register with a site, you create an account userID and password; with DIDs, you will create and share a new DID (and DIDDoc) with the site. When you return to the site, providing the DID and proving that you control the DID are sufficient for the service to know who you are. Hey, a way to eliminate user IDs and passwords!

A public DID is one that is intended to be widely available—visible to anyone that is presented with the DID. Since anyone can both resolve the DID and contact the owner, if the DID is used many times, it is correlatable. A company or government, for example, will often use public DIDs. A prime example of a public DID is the DID of a verifiable credential issuer. Since the holder of the credential may present the credential to anyone, the identity (via the public DID) of the issuer must be part of what the verifier learns from the presentation. That way, the verifier can investigate (as necessary) to decide if they trust the issuer. Public DIDs are usually put on blockchains so that they can be globally resolved.

- Public DIDs are published on the blockchain.

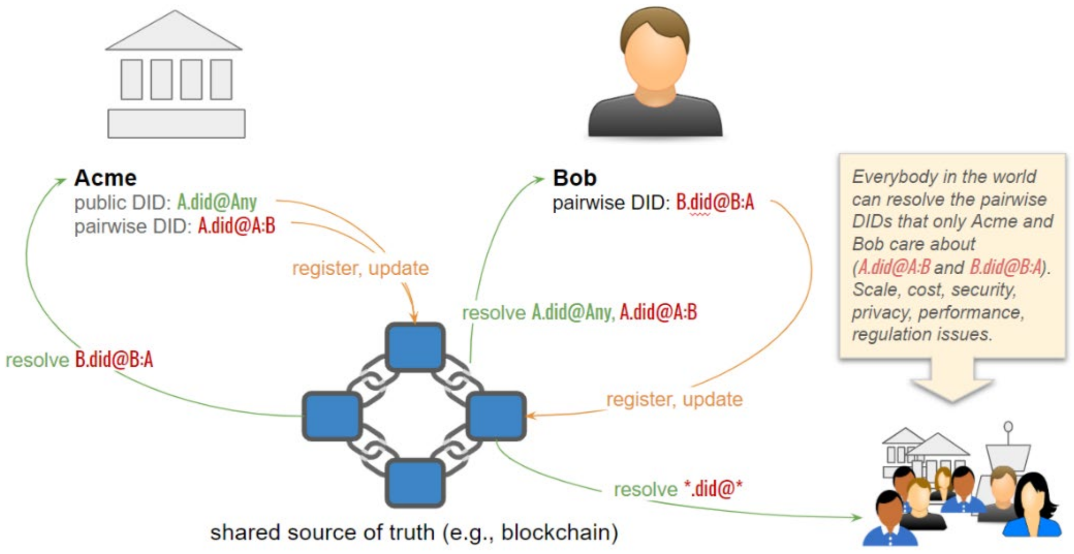

Acme
public DID: A.did@Any
pairwise DID: A.did@A:B
Bob
pairwise DID: B.did@B:A
Everybody in the world can resolve the pairwise DIDs that only Acme and Bob care about (A.did@A:B and B.did@B:A). Scale, cost, security, privacy, performance, regulation issues.
register, update
resolve A.did@Any, A.did@A:B
resolve B.did@B:A
register, update
resolve *.did@*
shared source of truth (e.g., blockchain)

Figure 7 Public DID work structure

On the other hand, an important use case of DIDs is that they enable two (or several) parties to connect with one another. In this case, everyone in the world doesn't have to be able to resolve the DIDs for the parties—just the parties themselves—and private DIDs (often called pairwise DIDs) are used. … No one other than the parties involved can see or resolve the DIDs.

- Private DIDs are not published on the blockchain.

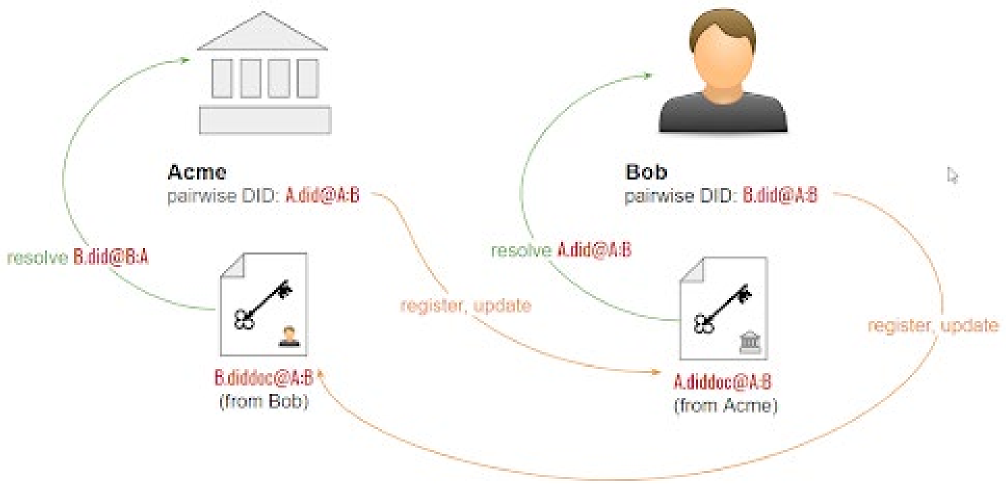


Figure 8 Private DID work structure

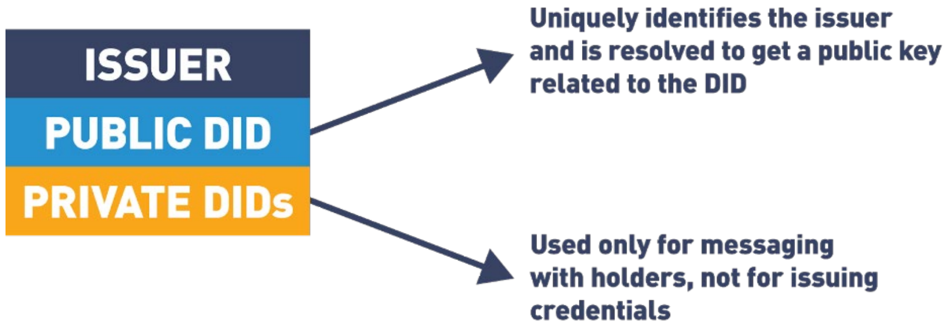

ISSUER
PUBLIC DID
PRIVATE DIDs
Uniquely identifies the issuer and is resolved to get a public key related to the DID
Used only for messaging with holders, not for issuing credentials

Figure 9 Public vs Private DID

2.3.2. Zero-Knowledge Proof

A zero-knowledge proof (ZKP) is about proving attributes about an entity (a person, organization, or thing) without exposing a correlatable identifier about that entity.

Formally, it's about presenting claims from verifiable credentials without exposing the key (and hence a unique identifier) of the proving party to the verifier. A ZKP still exposes the data asked for (which could uniquely identify the prover), but does so in a way that proves the prover possesses the issued verifiable credential while preventing multiple verifiers from correlating the prover's identity. This approach is called Anonymous Credentials (AnonCreds).

In the non-ZKP model, the DID of the holder/prover is included in the credential and hence any verifiers of the credential know a common, globally unique identifier for the holder/prover, which is a privacy concern as it is a point of correlation.

2.3.3. Trust Over IP (ToIP)

Along with SSI, another term you will see is trust over IP (ToIP). ToIP is a set of protocols being developed to enable a layer of trust on the Internet, for example, protocols embodied in Hyperledger Indy, Aries and Ursa. It includes self-sovereign identity in that it covers identity, but goes beyond that to cover any type of authentic data. Authentic data in this context is data that is not necessarily a credential (attributes about an entity) but is managed as a verifiable credential and offers the same guarantees when proven.

ToIP is defined by the “Trust over IP Technology Stack,” as represented in this image from Drummond Reed:

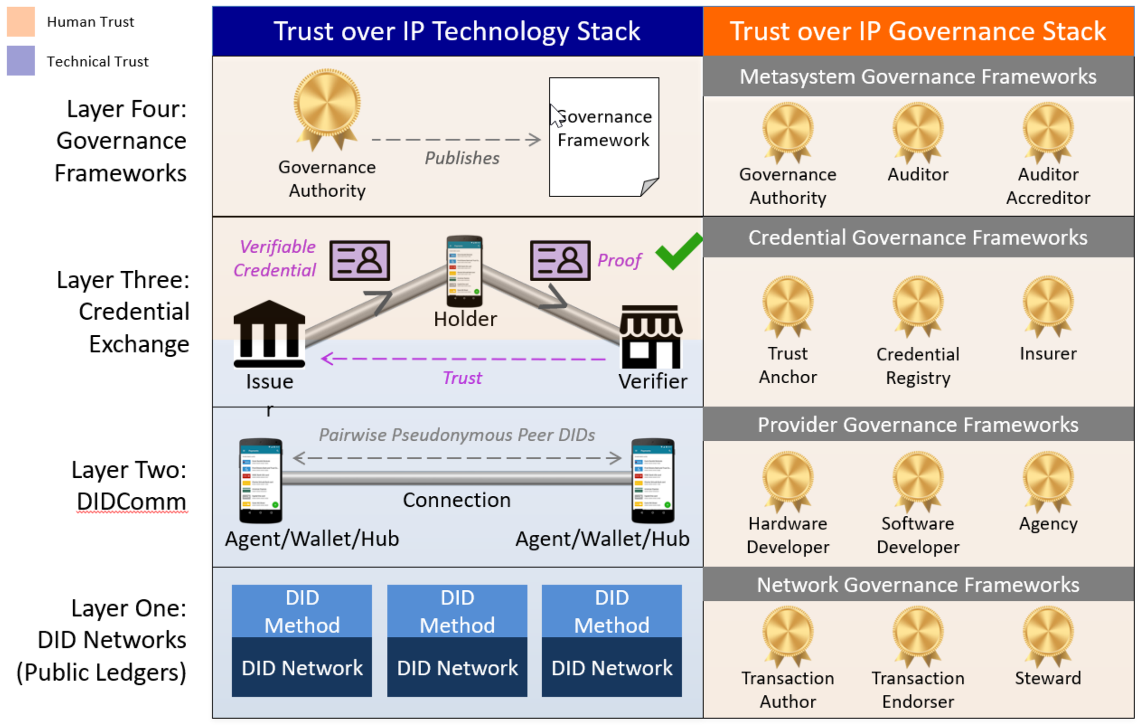


Figure 10 Trust over IP Technology Stack

2.3.4. How SSI Address Internet Challenges

Internet Challenges Today	How Verifiable Credentials, SSI and ToIP Address These Challenges
User IDs/passwords are the norms, but they are a pain to use, and as a result, are susceptible to attack. They are the best we have right now, but not a solid basis for trust. Further, IDs work only one way—users don't get an ID from a service they use.	Websites can use DIDs and, as needed, verifiable credentials to get enough information about users to establish sessions. We’ll see later that using DIDs and verifiable credentials are a lot easier and safer for users than passwords.
Other personal information and identifiers we have that we could otherwise use to prove our identity are not trusted because it's impossible to tell if the data was actually	Presentations of claims from verifiable credentials and knowing who issued the credentials mean that we can trust the claims to be correct. This is based on the four

issued to the person providing it. The many breaches of private identifiers make them impossible to completely trust, and verifying that information adds (sometimes significant) costs.	attributes we learn when verifying a presentation.
Since the identity attributes we could use are not trusted (they are not things only we know), we often have to resort to in-person delivery of paper documents to prove things about ourselves, a further cost for all participants.	We can use verifiable credentials online instead of having to use paper documents in-person.
Reviewers of the paper documents we present must become experts in the state of the art in forging and falsifying paper documents, a difficult role in which to be placed.	The trust of the claims is in cryptography and knowing about the issuer. People don’t have to be experts at detecting forgeries.
The identifiers we use are correlated across sites, allowing inferences to be made about us, and exposing information we don't intend to be shared across sites. This is annoying at the least, and can have catastrophic results in the worst case.	By using private DIDs and verifiable credentials based on ZKPs, we can radically reduce the online correlation that is happening today.
Centralized repositories of identifiers and data about the people associated with those identifiers are targeted by hackers because the data has high value. This exacerbates the problem of not being able to trust "personal" data presented online (see above).	If high-value data is only accepted when presented as a claim from a verifiable credential, there is no value in having data from breaches.
Centralized identifiers can be abused by those who control those identifiers. For example, they can be taken away from a subject without due process.	You create your own DIDs and those identifiers cannot be taken away by a centralized authority. You control your DIDs, no one else.

2.4. Hyperledger

The Hyperledger Foundation, funded in 2015, is the open, global ecosystem for enterprise blockchain technologies. As part of the Linux Foundation, it is a neutral home for developers to collaborate, contribute, and maintain open-source software.

The Hyperledger Foundation hosts many open source projects that serve as the building blocks for enterprise blockchain deployments. These projects are conceived and built by the Hyperledger developer community as freely available, enterprise-grade software that vendors, end users, service providers, start-ups, academics, and others can use to build and deploy blockchain networks and even commercial solutions.

Because they are developed and governed as open source technologies, Hyperledger projects are all community-led. This means that the code is written collaboratively and available for review and use by anyone. Decisions about development roadmaps and priorities are made openly and cooperatively.

Open-source software development is a transparent process, which is particularly fitting for blockchain technologies. It brings together organizations and individuals with different requirements and drives them to work together to develop common solutions that can be the foundation for mutual success—another good parallel with blockchain.

The role of the Hyperledger Foundation is to ensure the health and transparency of the community and all its projects, including managing the development cycle, software licensing, security audits, and provenance tracking for every line of code.

2.4.1. Hyperledger Identity "Stack"

Hyperledger presents new approaches enabled by blockchain technology that are the building blocks of decentralized identity—approaches that make SSI possible. Hyperledger Indy, Aries and Ursa are implementations of those building blocks.

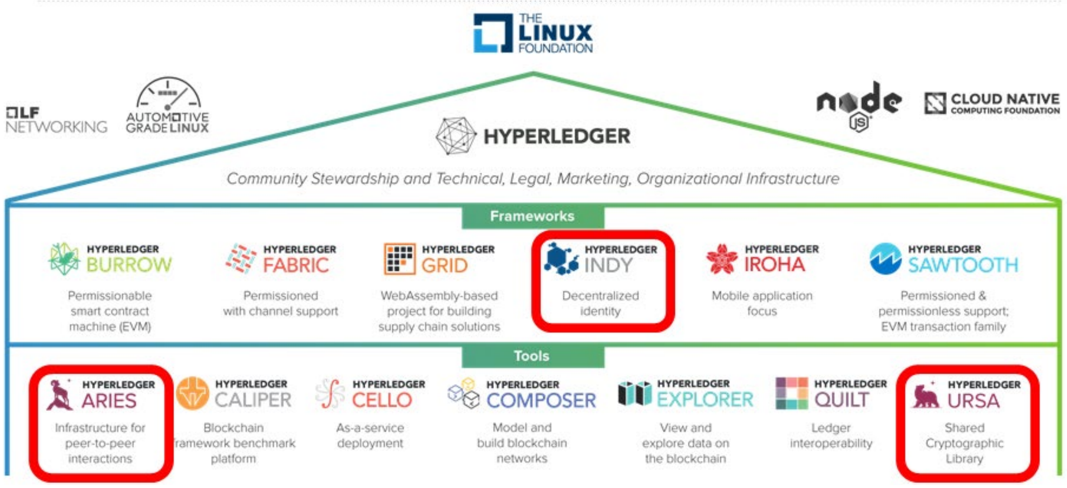

THE LINUX FOUNDATION
LF NETWORKING
AUTOMOTIVE GRADE LINUX
node JS
CLOUD NATIVE COMPUTING FOUNDATION
HYPERLEDGER
Community Stewardship and Technical, Legal, Marketing, Organizational Infrastructure
Frameworks
HYPERLEDGER BURROW
Permissionable smart contract machine (EVM)
HYPERLEDGER FABRIC
Permissioned with channel support
HYPERLEDGER GRID
WebAssembly-based project for building supply chain solutions
HYPERLEDGER INDY
Decentralized identity
HYPERLEDGER IROHA
Mobile application focus
HYPERLEDGER SAWTOOTH
Permissioned & permissionless support; EVM transaction family
Tools
HYPERLEDGER ARIES
Infrastructure for peer-to-peer interactions
HYPERLEDGER CALIPER
Blockchain framework benchmark platform
HYPERLEDGER CELLO
As-a-service deployment
HYPERLEDGER COMPOSER
Model and build blockchain networks
HYPERLEDGER EXPLORER
View and explore data on the blockchain
HYPERLEDGER QUILT
Ledger interoperability
HYPERLEDGER URSA
Shared Cryptographic Library

Figure 11 Linux Foundation Leadership

Hyperledger Indy was Hyperledger's first "identity-focused" blockchain framework, joining Hyperledger in 2017. The code for Indy was contributed by the Sovrin Foundation.

As Indy evolved within Hyperledger, there was a realization that the cryptography in Indy could be used in several Hyperledger projects, and even outside of Hyperledger. In 2018, the decision was made to migrate the indy-crypto code repository out of Indy and into its own project: Hyperledger Ursa.

Although Indy is an excellent implementation of SSI capabilities, it is not the only implementation. In the long term, there will likely be multiple implementations that are used by different people, organizations and communities. What if we had agents that could use DIDs and verifiable credentials from multiple ecosystems? Aries was proposed on that basis, and accepted as a Hyperledger project in March 2019.

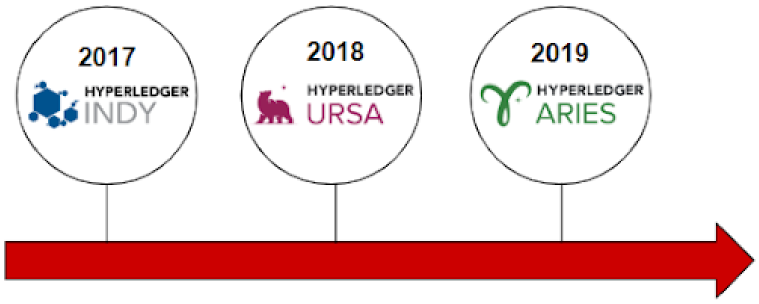


Figure 12 Identity Stack Creation Timeline

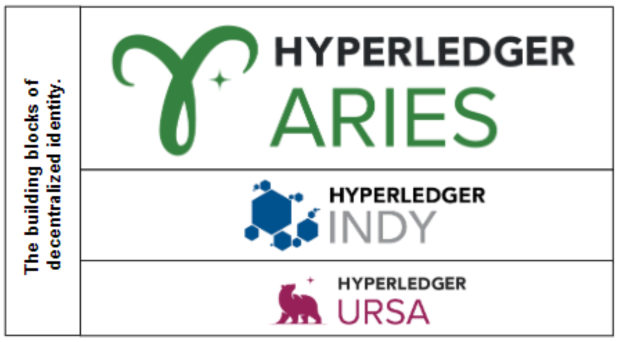


Figure 13 Identity Stack

2.4.1.1. Hyperledger Indy

Indy includes verifiable credentials based on zero-knowledge proof (ZKP) technology, decentralized identifiers, a software development kit (SDK) for building agents and an implementation of a public, permissioned distributed ledger.

Figure 14 Indy Logo

What Goes on the Blockchain?

Public DIDs: When the issuer wants to issue a credential, they MUST have a DID on the blockchain that allows verifiers to find out who they are, and in the process, why the credentials they issue can be trusted.

Schemas: Next, the issuer MAY put a schema on the blockchain that says in effect, "When I issue a credential, it's going to have these fields in it." An issuer might put their own schema on the ledger, or they can use one that someone else put on the ledger. In the current iteration of Indy, the schema is just a list of attribute names. As this course is being written, work is ongoing in the Indy project to add what is known as "rich schema." Once that is in place, a schema will include additional information about the attributes, including a data type and how the information is encoded in the credential.

Credential Definitions: Before an issuer can issue a credential using a schema, they MUST put a credential definition on the blockchain. A credential definition says, "I'm an issuer using this DID, I'm going to use that schema for my credentials, and here are the public keys (one per claim) that I'm going to use to sign the claims when I issue a credential." In all, an issuer must have a DID, a schema and a credential definition on the blockchain before issuing credentials.

Revocation Registry: The issuer MAY want to be able to revoke credentials and if they do, they must also write a revocation registry to the ledger before issuing credentials. The revocation registry links back to the credential definition, and allows the issuer to unilaterally revoke credentials, independent of the holders. No one can look at the revocation registry to see if a specific credential has been revoked. But, with the magic of a zero-knowledge proof (ZKP), a holder can prove (and a verifier can verify) that the holder's credential has not been revoked—provided of course it has not been revoked. This action is taken by the prover when creating a presentation.

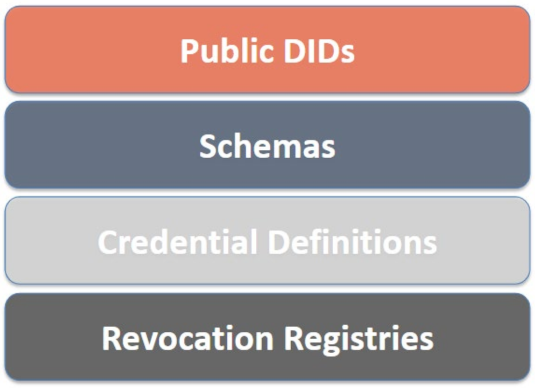


Figure 15 What goes on Indy Blockchain

There are some public implementations of the Indy network available. One of them is the Sovrin Indy instance. The Sovrin Foundation is a global non-profit that has organized the deployment of Hyperledger Indy code on a (growing) number of nodes to create a running public permissioned Indy blockchain instance.

2.4.1.2. Hyperledger Aries

Aries is a toolkit designed for initiatives and solutions focused on creating, transmitting, storing and using verifiable digital credentials. At its core are protocols enabling connectivity between agents using secure messaging to exchange information. Aries is all about peer-to-peer interactions between agents controlled by different entities—people, organizations and things. Using the standardized messaging channel, verifiable credentials can be exchanged based on DIDs rooted in different ledgers (based on Indy or other technology) using a range of verifiable credentials implementations.

Figure 16 Aries Logo

Verifiable credentials can be exchanged based on DIDs rooted in different ledgers (based on Indy or other technology), and it is “verifiable credential-agnostic”—support different verifiable credential implementations within a single agent.

Agents

An Aries agent is a piece of software that enables an entity (a person, organization or thing) to assume one or more of the roles within the verifiable credential model—an issuer, holder or verifier—and allows that entity to interact with others that also have verifiable credential roles. Agents may do (many!) other things, but it’s their ability to handle verifiable credentials that is their defining characteristic.

- Private, pairwise DIDs (secure their peer-to-peer communications)
- Different pairwise DID for every relationship, each with keys and endpoints that you control, communication between agents is end-to-end encrypted, secure and safe.
- Verifiable credentials and proofs are easily exchanged between agents, you can be certain who is on the other end of the connection.

Device Agents

Internet of Things (IoT) devices might embed agents capable of issuing verifiable credentials based on data from the device’s sensors. The devices themselves might be certified by a standards organization, and by immediately putting the sensor data into a verifiable credential, the data can be tied to the device and proven not to be tampered with as it is used. For example, sensors in your car can generate verifiable credentials to track mileage and maintenance on the vehicle so that the data can be proven when the car is sold. Or, inspections on gas pumps could be handled by the pump issuing verifiable credentials that could be monitored and only inspected when generating faulty data.

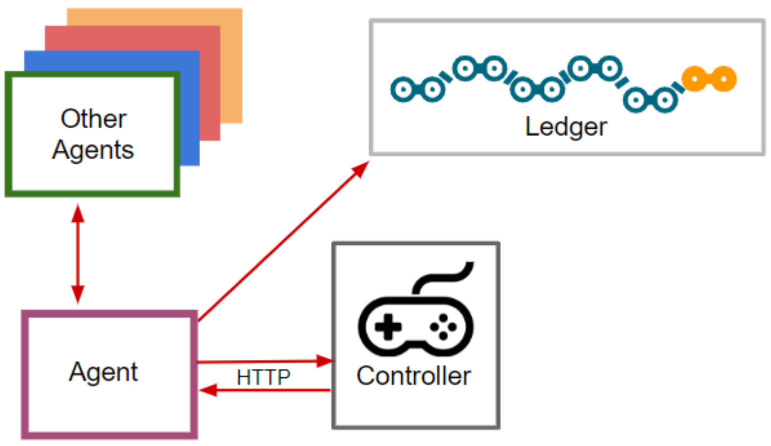


Figure 17 Aries Agents communication

3. Methodology / project development:

In this section will be detailed step by step the procedures carried out for the realization of the project. It will detail how the interaction between a holder (Application) and an issuer (IoT device) is made using ACA through a public Indy ledger. It will be taking into account all the necessary parts, which include the creation of a graphical application the configuration of the Raspberry Pi with the RFID sensor, the use of ACA project and the configuration of the Indy ledger.

It is worth it to say that Hyperledger Aries is a project in progress, it does not even have a release version yet. Everything that you will see below about ACA-py can be perfectly outdated in newer versions than this one that we will be using. Also you may face many problems with compatibilities between dependencies. So what you will see here it is exactly what worked for me.

3.1. Scenario

The scenario will be a delivery. Imagine you have purchased a product and you want to see the current delivery status.

This will be tracked by the IoT device. At the very beginning of the delivery, the product will be attached to an IoT device with an RFID sensor. We will assume that the deliverer has an RFID tag and the final user has another. When the deliverer gets the product, passes his RFID tag near the sensor and immediately the IoT device updates the status to "in delivery". As soon as the product reaches the final user he would do the same and the status would be updated to "delivered".

In order to watch them in real-time the status, we will have a graphical Application. This App is not only meant for data tracking, it will have another feature, the create invitation for the connection between the App and the sensor, which we will discuss later.

This would be the high-level overview of the experiment, there are a lot of other things happening under this.

Since the main goal of this experiment is to be based on one of the practical solutions available for SSI (in this case Hyperledger Identity Stack), we will use a developer tool from Hyperledger Aries, which is ACA-py. There will be one instance running per each part, one for the sensor and one for the App. The concept with this would be to delegate the connection and data transmission between them and the blockchain to their Agents. This can be done because the ACA-py is used as an API, which each of its endpoints is designed to do a specific part of the interaction.

In order to have a clear understanding of the experiment, it is necessary to define what we want to happen.

The following diagram shows the experiment, all the components involved, and their connections:

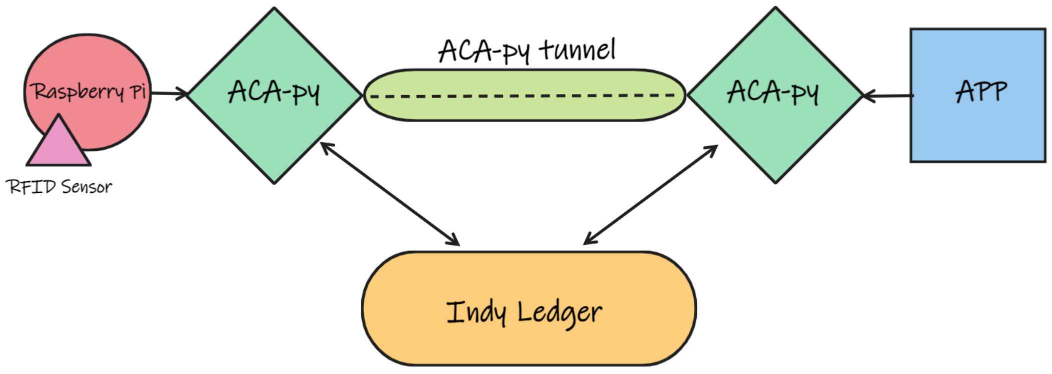


Figure 18 Components diagram

3.1.1. Components

We will have a closer look at the components; a short functional explanation of them.

As you could see in the diagram, the IoT part is simulated with a Raspberry Pi and an RFID sensor. In this case, we will be using a Raspberry Pi 4 Model B and a PN532 NFC RFID MODULE V3.

Its function will be to read the UID from the two different tags and make the proper calls to the Agent API to send de new status.

Immediately we have the ACA-py instance for the sensor. It will be configurated with a public API address, the IoT is not running on the same machine and we need to make the calls through the internet.

Its function will be to communicate with the ledger and the App Agent to send the new status.

Quite similar to the other part, we have the ACA-py instance for the App. It is not necessary to configure a public API address because it will be running on the same machine as the App, so we can use the localhost address.

Its function will be to communicate with the ledger and the sensor Agent to store the new status.

Then we have the App, which will be a simple graphical interface. It will have a log-in menu at the start, where you should introduce the App wallet password. If it is correct you will be sent to the menu, where you can create an invitation connection for the sensor and also see the credentials stored in the wallet, those which have the status written.

Its function will be to create the connection invitation and fetch the credentials in the wallet.

Finally, we have the Indy ledger, in this case, we will be using one of the free public instances currently available on the internet, the BCovrin Dev ledger. It has a really simple interface to see the nodes running, register DIDs, and see the ledger state.

Its function will be to store the sensor DID, its schemas, and credential definitions.

3.1.2. Step by Step: Achieve Sending Credentials

The sending and receiving of the delivery status updates will be done by verifiable credentials. This credential contains an attribute field where we will write the new status value. Then it will be sent from the sensor to the App. In order to make this work, it is necessary to understand every step before and during the sending.

Just to clarify, this procedure is adapted to the Aries protocols, it may change in other SSI projects/solutions. Also, the verifier part is not implemented.

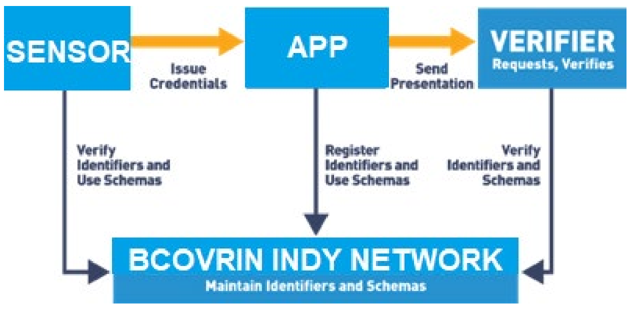

SENSOR
Issue Credentials
APP
Send Presentation
VERIFIER
Requests, Verifies
Verify Identifiers and Use Schemas
Register Identifiers and Use Schemas
Verify Identifiers and Schemas
BCOVRIN INDY NETWORK
Maintain Identifiers and Schemas

Figure 19 Project SSI workflow diagram

3.1.2.1. Concept

First we will have a look to the whole process inside the ACA logic.

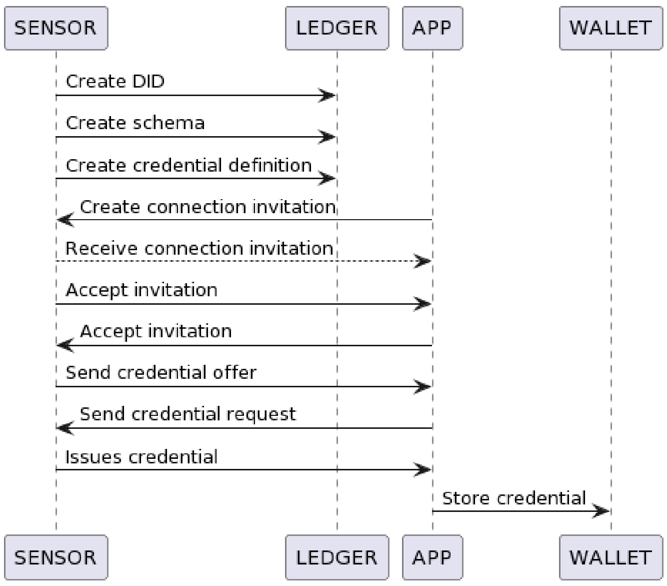


Figure 20 Concept UML diagram

We will differentiate three parts.

The first one would be everything the ledger needs to have before to send a credential. Which is to have registered a public DID from the sender/issuer, a schema and a credential definition related to that schema.

Once you have a DID registered you can create the connection, this steps are:

1. App creates a connection invitation.
2. Sensor gets the invitation and does the receive connection invitation.
3. Sensor accepts the invitation.
4. App accepts invitation.

Now the connection should be created and we can move on to the credential dance. For this it is necessary to have the schema and credential definition already created, in the ledger and the wallet.

1. Sensor sends a credential offer.
2. App sends a credential request.
3. Sensor creates and issues the credential.
4. App stores it in the wallet.

With this we could fetch the credential from the App wallet in order to read it.

3.1.2.2. Detailed Concept

Once the general concept is clear, we will see what is really happening behind all the components. This diagram includes the interaction with the ACA-py Agents as well.

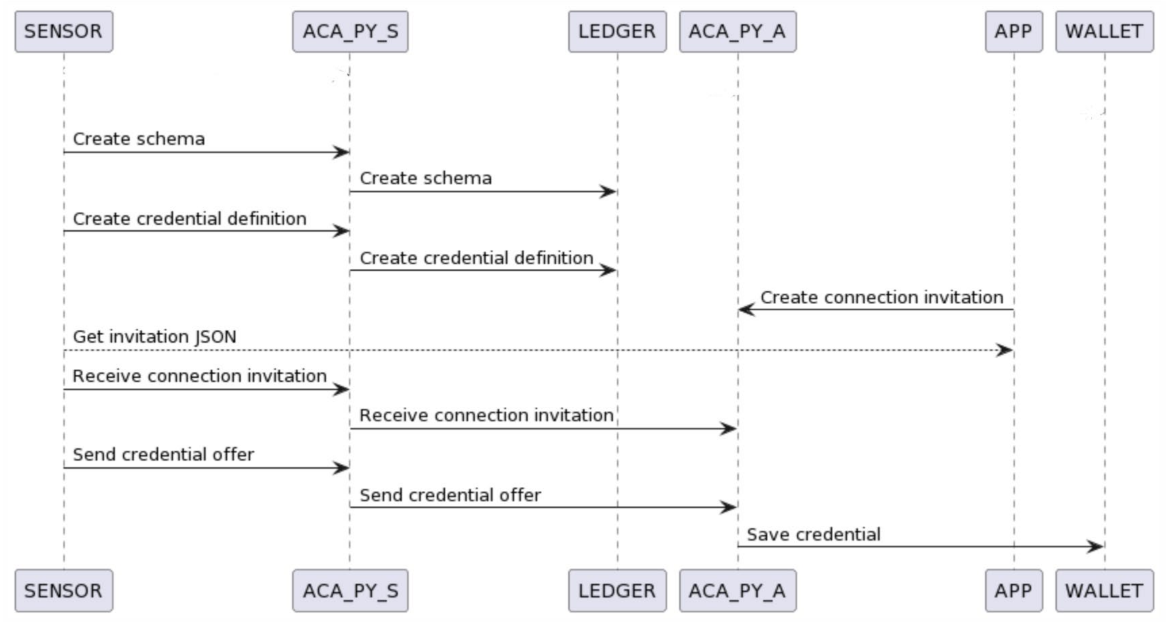

SENSOR
ACA_PY_S
LEDGER
ACA_PY_A
APP
WALLET
Create schema
Create schema
Create credential definition
Create credential definition
Create connection invitation
Get invitation JSON
Receive connection invitation
Receive connection invitation
Send credential offer
Send credential offer
Save credential

Figure 21 Detailed Concept UML diagram

As we can see, there are some differences with the last diagram:

- In the simulation we will skip some steps of the connection and the credential dance by enabling some auto-flags when running the ACA-py instances. In a real scenario this may not be a proper way because it may be a good behaviour to ensure what kind of interactions are you accepting. But for this simple case it will save some code and time.
- The DID registration is made manually from the BCovrin web page interface that is why is not shown here. Conversely, the schema and credential definition are made using the ACA-py.
- In order to get the connection invitation created by the App, the sensor will directly access to a public server where the App stores it. This step is meant to be a key for security, in a real case this would be a QR scanning, which makes sure that the issuer and the holder meets and know each other.

3.2. Public Indy Ledger

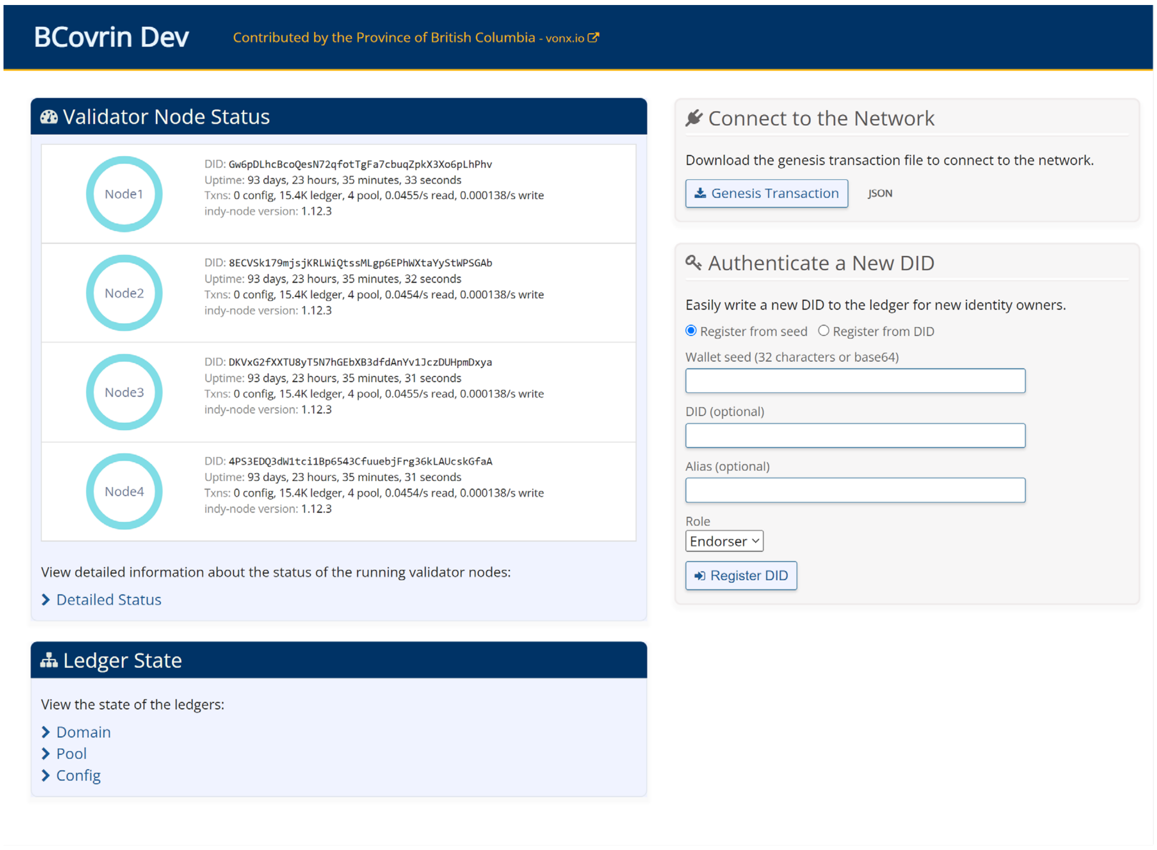
3.2.1. Choosing between Public Indy Ledgers

In the “Introduction to Hyperledger Sovereign Identity Blockchain Solutions: Indy, Aries & Ursa” course from the Linux Foundation they mention and explain the Sovrin public ledger. So I first chose the Sovrin BuilderNet since is the free developer ledger. This test ledger is up-to-date with the new versions of Indy, which means that the TAA is currently working. This made a complete mess with the interactions between ACA-py and BuilderNet. ACA-py has no clear documentation for how to deal with TAA with this version of Indy.

But luckily there is another free public ledger that seems to not have been implemented yet the TAA, the BCovrin.

3.2.2. BCovrin Blockchain

The Government of British Columbia has three Indy networks, for test, development and production. We are going to use the development one.



BCovrin Dev Contributed by the Province of British Columbia - vonx.io

Validator Node Status

Node1
DID: Gw6pDLhcBcoQesN72qfotTgFa7cbuqZpkX3Xo6pLhPhv
Uptime: 93 days, 23 hours, 35 minutes, 33 seconds
Txns: 0 config, 15.4K ledger, 4 pool, 0.0455/s read, 0.000138/s write
indy-node version: 1.12.3

Node2
DID: 8ECVSk179mjsjKRLWiQtssMLgp6EPhWXtaYyStWPSGAb
Uptime: 93 days, 23 hours, 35 minutes, 32 seconds
Txns: 0 config, 15.4K ledger, 4 pool, 0.0454/s read, 0.000138/s write
indy-node version: 1.12.3

Node3
DID: DKVxG2fXXTU8yT5N7hGEbXB3dfdAnYv1JczDUHpmDxya
Uptime: 93 days, 23 hours, 35 minutes, 31 seconds
Txns: 0 config, 15.4K ledger, 4 pool, 0.0455/s read, 0.000138/s write
indy-node version: 1.12.3

Node4
DID: 4PS3EDQ3dW1tci1Bp6543CfuuebjFrg36kLAUcskGfaA
Uptime: 93 days, 23 hours, 35 minutes, 31 seconds
Txns: 0 config, 15.4K ledger, 4 pool, 0.0454/s read, 0.000138/s write
indy-node version: 1.12.3

View detailed information about the status of the running validator nodes:
> Detailed Status

Ledger State

View the state of the ledgers:
> Domain
> Pool
> Config

Connect to the Network

Download the genesis transaction file to connect to the network.
Genesis Transaction JSON

Authenticate a New DID

Easily write a new DID to the ledger for new identity owners.
[✓ Register from seed] Register from DID
Wallet seed (32 characters or base64)
DID (optional)
Alias (optional)
Role
Endorser
Register DID

Figure 22 BCovrin Blockchain web

3.2.2.1. Register public DID

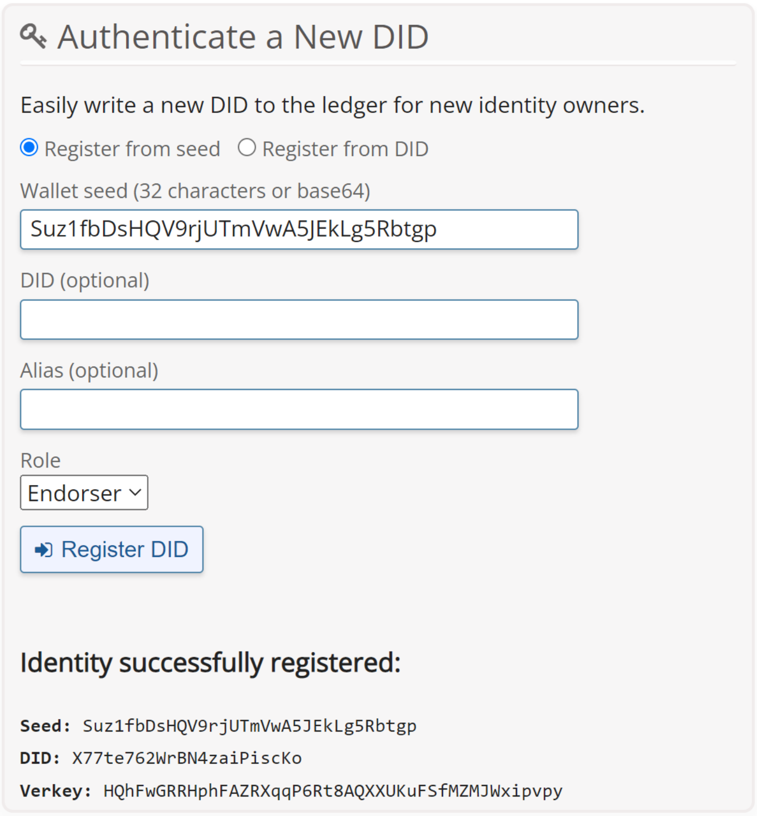
As we have explained before, the registration of the public DID for the sensor is directly made on the BCovrin page. We have on the right side the “Authenticate a New DID” box, which lets you register a new DID from seed or a DID. We will use the seed option. This is important because we can later use this seed to configure the ACA-py.

In this case, we have generated a random string of 32 characters using an online random text generator.

Once you wrote the seed and clicked on “Register DID” a new box should appear saying “Identity successfully registered” with three values:

- The seed used.
- The DID generated from this seed.
- The verification key for this DID.

We will save the seed and the DID for later use.



Authenticate a New DID

Easily write a new DID to the ledger for new identity owners.

[✓ Register from seed] Register from DID

Wallet seed (32 characters or base64): Suz1fbDsHQV9rjUTmVwA5JEkLg5Rbtgp

DID (optional):

Alias (optional):

Role: Endorser

Register DID

Identity successfully registered:

```
Seed: Suz1fbDsHQV9rjUTmVwA5JEkLg5Rbtgp
DID: X77te762WrBN4zaiPiscKo
Verkey: HQhFwGRRHphFAZRXqqP6Rt8AQXXUKuFSfMZMJWxipvpy
```

Figure 23 Register a DID in BCovrin web

3.3. Aries Cloud Agent (ACA-PY)

Here we are going to see how to run two local instances, how to create a connection between them and send credentials.

3.3.1. Set-Up

There are different ways of running ACA-py. You can run it as a stand-alone application, and you can run it in a docker container.

I did both but I finally used the docker option. I had some strange problems with some of the steps when I first tried to do it on the stand-alone application.

For this project we will be using Ubuntu 20.04 LTS.

3.3.1.1. Stand-alone

First you need to install libindy.

(The major artifact of the SDK is a C-callable library that provides the basic building blocks for the creation of applications on the top of Hyperledger Indy) We had some problems installing "libindy". Some dependencies were needed, namely the "libsodium18" bookstore that only exists for Ubuntu 16. When I tried to install on Ubuntu versions 16 and 18, these dependencies installed, but there were more dependency errors.

So we tried the Ubuntu 20 version:

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 68DB5E88
$ sudo add-apt-repository "deb https://repo.sovrin.org/sdk/deb bionic master"
$ sudo apt-get update
$ sudo apt-get install -y libindy
```

Now that libindy is installed, let's continue with ACA-py:

```
$ pip3 install python3-indy
$ pip3 install aries-cloudagent
```

To check if ACA-PY is installed, just run the code aca-py --version and version 0.6.0 should appear.

If you do not have pip for Python 3, you can install it like:

```
$ sudo apt-get -y install python3-pip
```

You can run the –help like:

```
$ aca-py --help
```

3.3.1.2. Docker

If you do not want to install all of this on your machine, you can also run ACA-py in a docker container. There are ready-made images available at hub.docker.com/r/bcgovimages/aries-cloudagent which contain ACA-py and all the necessary dependencies. You can run the --help like:

```
$ docker run --net=host bcgovimages/aries-cloudagent:py36-1.16-0_0.6.0 --help
```

As you can see the commands added to the end are passed directly to ACA-py.

3.3.1.3. Differences

Depending on which way you use there are two point to take into account to make thinks work as well.

1) **Genesis file** when provisioning/running. If you cannot access the genesis file of the ledger with an URL, the option would be to pass it as a file downloaded in your system. But if you pick the docker option will not be able to do it just with the file path because it will try to find it inside the docker image, not in your system. One solution to this would be to create your local server, store the genesis file to one accessible endpoint and pass the genesis local URL.
2) **Wallets**. When provisioning, a wallet will be created. With the ACA-py stand-alone the wallets will be stored in the *~/.indy_client/wallet/* of your system. That will be also happening when running the docker option, but no from your system, it will be stored inside the image. The moment you stop the docker instance, the wallet will not exist anymore. It means it will be creating the wallet from scratch every time. This could be a problem for you because you will not be able to see the credentials from other runnings, and also you will not be able to use schemas already defined in the ledger by you in other executions. You will have to be creating different schemas and credential definitions.

3.3.2. Configurating & Running Agent Instances

Provisioning [only necessary when using stand-alone ACA-py]

ACA-py has a provision argument. We will use the provision mode of ACA-py to create a wallet configured for the ledger with the seed value that we have already registered in BCovrin browser.

This step is only for the

For the sensor:

```
$ sudo docker run --net=host bcgovimages/aries-cloudagent:py36-1.16-0_0.6.0
provision \
  --endpoint http://localhost:8000/ \
  --genesis-url http://dev.bcovrin.vonx.io/genesis \
  --wallet-type indy \
  --wallet-name SensorWallet \
  --wallet-key secret \
  --seed Suz1fbDsHQV9rjUTmVwA5JEkLg5Rbtgp
```

```
>
Created new profile
Profile backend: indy
Profile name: SensorWallet
Created new public DID: X77te762WrBN4zaiPiscKo
Verkey: HQhFwGRRHphFAZRXqqP6Rt8AQXXUKuFSfMZMJWxipvpy
Ledger configured
```

- **--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.
- **--genesis-url http://dev.bcovrin.vonx.io/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.

- **--wallet-type indy**, **--wallet-name SensorWallet** 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.
- **--seed** This is the seed that we have saved from the one we registered the public DID in the BCovrin network. This seed value proves that you are the owner of the public DID.

We will do the same for the App:

```
$ sudo docker run --net=host bcgovimages/aries-cloudagent:py36-1.16-0_0.6.0
provision \
  --endpoint http://localhost:8001/ \
  --genesis-url http://dev.bcovrin.vonx.io/genesis \
  --wallet-type indy \
  --wallet-name AppWallet \
  --wallet-key secret \
  --wallet-local-did
```

```
>
Created new profile
Profile backend: indy
Profile name: AppWallet
Ledger configured
```

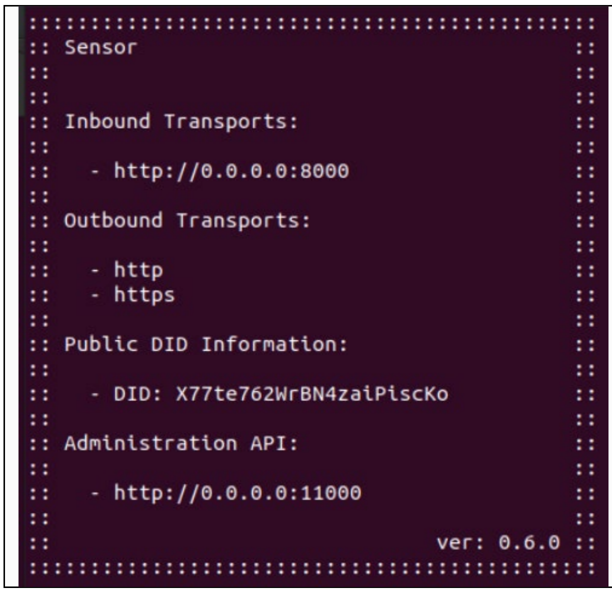
There is one new parameter: **--wallet-local-did**. The App doesn't have a public DID, but it does need a local DID. The local DID will be used for the sensor to create a credential for, more on that later.

Starting Agents

To start the ACA-py instance for the sensor:

```
sudo docker run --net=host bcgovimages/aries-cloudagent:py36-1.16-0_0.6.0
start \
--label Sensor \
-it http 0.0.0.0 8000 \
-ot http \
--admin 0.0.0.0 11000 \
--admin-insecure-mode \
--genesis-url http://dev.bcovrin.vonx.io/genesis \
--endpoint http://localhost:8000/ \
--debug-connections \
--wallet-type indy \
--wallet-name SensorWallet \
--wallet-key secret \
--seed Suz1fbDsHQV9rjUTmVwA5JEkLg5Rbtgp \
--auto-accept-invites
```

```
>
```



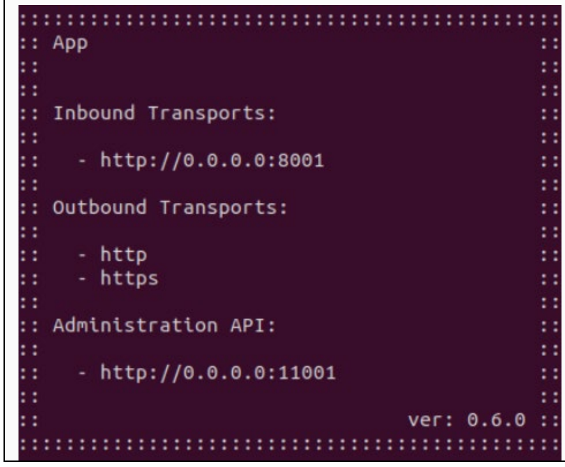
```
::::::::::::::::::::::::::::::::::::::::::::::::
:: Sensor                                     ::
::                                            ::
::                                            ::
:: Inbound Transports:                        ::
::                                            ::
::   - http://0.0.0.0:8000                    ::
::                                            ::
:: Outbound Transports:                       ::
::                                            ::
::   - http                                   ::
::   - https                                  ::
::                                            ::
:: Public DID Information:                    ::
::                                            ::
::   - DID: X77te762WrBN4zaiPiscKo            ::
::                                            ::
:: Administration API:                        ::
::                                            ::
::   - http://0.0.0.0:11000                   ::
::                                            ::
::                                ver: 0.6.0  ::
::::::::::::::::::::::::::::::::::::::::::::::::
```

- **--label Sensor** 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 sensor. These are the endpoints your controller application will interact with.
- **--debug-connections** This parameter makes sure that more information about connections is being printed when we start making a connection between sensor and the App in the next section.
- **--auto-accept-invites** This parameter automates the acceptance of the connection invitations.

To start the ACA-py instance for the App:

```
sudo  docker  run  --net=host  bcgovimages/aries-cloudagent:py36-1.16-0_0.6.0
start \
--label App \
-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://dev.bcovrin.vonx.io/genesis \
--debug-connections \
--wallet-local-did \
--wallet-type indy \
--wallet-name AppWallet \
--wallet-key secret \
--auto-accept-requests \
--auto-respond-credential-offer \
--auto-store-credential
```

>



```
::::::::::::::::::::::::::::::::::::::::::::::
:: App                                      ::
::                                          ::
::                                          ::
:: Inbound Transports:                      ::
::                                          ::
::   - http://0.0.0.0:8001                  ::
::                                          ::
:: Outbound Transports:                     ::
::                                          ::
::   - http                                 ::
::   - https                                ::
::                                          ::
:: Administration API:                      ::
::                                          ::
::   - http://0.0.0.0:11001                 ::
::                                          ::
::                              ver: 0.6.0  ::
::::::::::::::::::::::::::::::::::::::::::::::
```

- **--auto-accept-requests** This parameter automates the acceptance of the connection requests.
- **--auto-respond-credential-offer** This parameter automates the answer of the credential offer.
- **--auto-store-credential** This parameter automates the storing of the credential in the wallet as soon as the credential dance is finished.

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

3.3.3. Connection between Agents

The connection should follow this structure but as we said before, we will skip some steps with the auto flags enabled at the start of the run. This will be the acceptation steps.

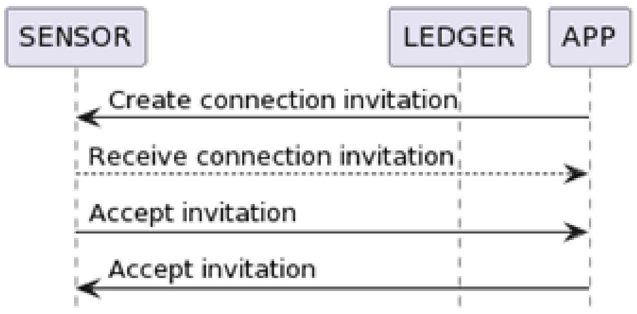


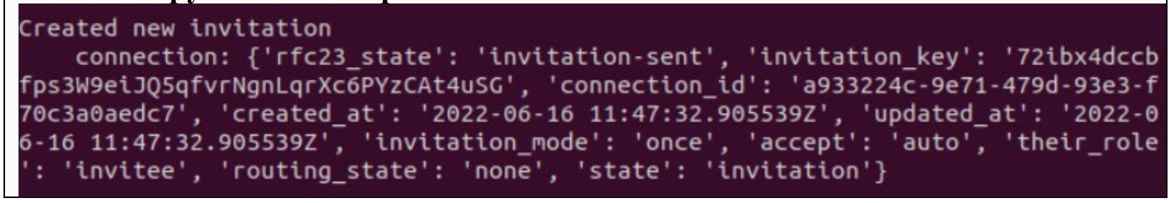
Figure 24 Agents Connection UML diagram

App creates an invitation

```
$ curl -X POST "http://localhost:11001/connections/create-invitation" \
-H 'Content-Type: application/json' \
-d '{}'
```

```
>
{
	"connection_id": "a933224c-9e71-479d-93e3-f70c3a0aedc7",
	"invitation": {
		"@type":
"did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/connections/1.0/invitation",
		"@id": "5f96e514-2a51-4ee0-9f28-5ea51a35cd8e",
		"label": "App",
		"recipientKeys":
["72ibx4dccbfps3W9eiJQ5qfvrNgnLqrXc6PYzCAt4uSG"],
		"serviceEndpoint": "http://localhost:8001/"
	},
	"invitation_url":
"http://localhost:8001/?c_i=eyJAdHlwZSI6ICJkaWQ6c292OkJ6Q2JzTlloTXJqSGlxWkRUV
UFTSGc7c3BlYy9jb25uZWN0aW9ucy8xLjAvaW52aXRhdGlvbiIsICJAaWQiOiAiNWY5NmU1MTQtMm
E1MS00ZWUwLTlmMjgtNWVhNTFhMzVjZDhlIiwgImxhYmVsIjogIkFwcCIsICJyZWNpcGllbnRLZXl
zIjogWyI3MmlieDRkY2NiZnBzM1c5ZWlKUTVxZnZyTmduTHFyWGM2UFl6Q0F0NHVTRyJdLCAic2Vy
dmljZUVuZHBvaW50IjogImh0dHA6Ly9sb2NhbGhvc3Q6ODAwMS8ifQ=="
}
```

APP ACA-py terminal output



```
Created new invitation
    connection: {'rfc23_state': 'invitation-sent', 'invitation_key': '72ibx4dccb
fps3W9eiJQ5qfvrNgnLqrXc6PYzCAt4uSG', 'connection_id': 'a933224c-9e71-479d-93e3-f
70c3a0aedc7', 'created_at': '2022-06-16 11:47:32.905539Z', 'updated_at': '2022-0
6-16 11:47:32.905539Z', 'invitation_mode': 'once', 'accept': 'auto', 'their_role
': 'invitee', 'routing_state': 'none', 'state': 'invitation'}
```

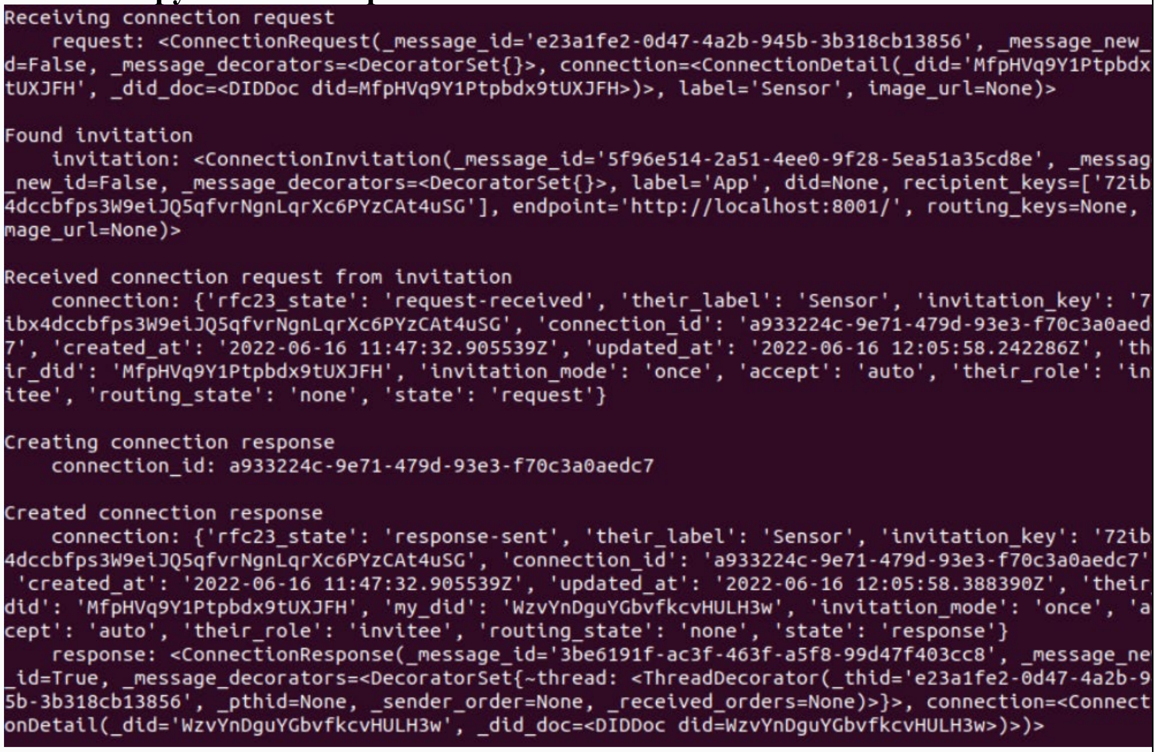
Sensor receives the invitation

We have to copy and paste in the body the value of the invitation key of the previous step.

```
$ curl -X POST "http://localhost:11000/connections/receive-invitation" \
-H 'Content-Type: application/json' \
-d '{
	"@type":
"did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/connections/1.0/invitation",
	"@id": "5f96e514-2a51-4ee0-9f28-5ea51a35cd8e",
	"label": "App",
	"recipientKeys": ["72ibx4dccbfps3W9eiJQ5qfvrNgnLqrXc6PYzCAt4uSG"],
	"serviceEndpoint": "http://localhost:8001/"
}'
```

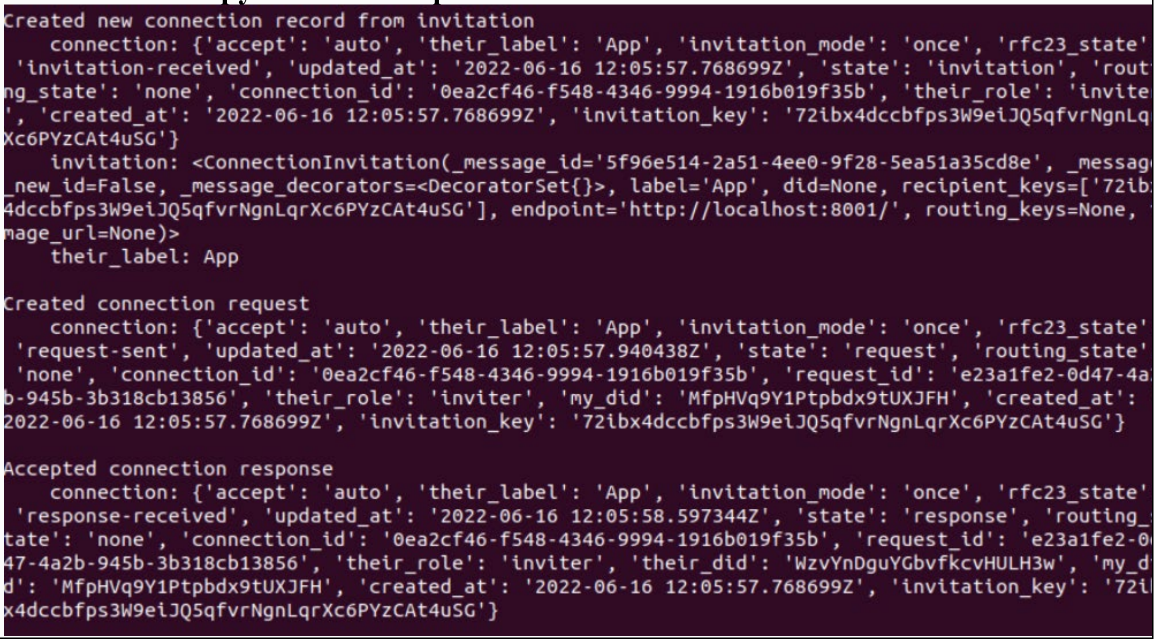
```
>
{
	"accept": "auto",
	"their_label": "App",
	"invitation_mode": "once",
	"rfc23_state": "request-sent",
	"updated_at": "2022-06-16 12:05:57.940438Z",
	"state": "request",
	"routing_state": "none",
        "connection_id": "0ea2cf46-f548-4346-9994-1916b019f35b",
        "request_id": "e23a1fe2-0d47-4a2b-945b-3b318cb13856",
        "their_role": "inviter",
        "my_did": "MfpHVq9Y1Ptpbdx9tUXJFH",
        "created_at": "2022-06-16 12:05:57.768699Z",
        "invitation_key": "72ibx4dccbfps3W9eiJQ5qfvrNgnLqrXc6PYzCAt4uSG"
}
```

APP ACA-py terminal output



```
Receiving connection request
    request: <ConnectionRequest(_message_id='e23a1fe2-0d47-4a2b-945b-3b318cb13856', _message_new_
d=False, _message_decorators=<DecoratorSet{}>, connection=<ConnectionDetail(_did='MfpHVq9Y1Ptpbdx
tUXJFH', _did_doc=<DIDDoc did=MfpHVq9Y1Ptpbdx9tUXJFH>)>, label='Sensor', image_url=None)>

Found invitation
    invitation: <ConnectionInvitation(_message_id='5f96e514-2a51-4ee0-9f28-5ea51a35cd8e', _messag
_new_id=False, _message_decorators=<DecoratorSet{}>, label='App', did=None, recipient_keys=['72ib
4dccbfps3W9eiJQ5qfvrNgnLqrXc6PYzCAt4uSG'], endpoint='http://localhost:8001/', routing_keys=None,
mage_url=None)>

Received connection request from invitation
    connection: {'rfc23_state': 'request-received', 'their_label': 'Sensor', 'invitation_key': '7
ibx4dccbfps3W9eiJQ5qfvrNgnLqrXc6PYzCAt4uSG', 'connection_id': 'a933224c-9e71-479d-93e3-f70c3a0aed
7', 'created_at': '2022-06-16 11:47:32.905539Z', 'updated_at': '2022-06-16 12:05:58.242286Z', 'th
ir_did': 'MfpHVq9Y1Ptpbdx9tUXJFH', 'invitation_mode': 'once', 'accept': 'auto', 'their_role': 'in
itee', 'routing_state': 'none', 'state': 'request'}

Creating connection response
    connection_id: a933224c-9e71-479d-93e3-f70c3a0aedc7

Created connection response
    connection: {'rfc23_state': 'response-sent', 'their_label': 'Sensor', 'invitation_key': '72ib
4dccbfps3W9eiJQ5qfvrNgnLqrXc6PYzCAt4uSG', 'connection_id': 'a933224c-9e71-479d-93e3-f70c3a0aedc7'
 'created_at': '2022-06-16 11:47:32.905539Z', 'updated_at': '2022-06-16 12:05:58.388390Z', 'their
did': 'MfpHVq9Y1Ptpbdx9tUXJFH', 'my_did': 'WzvYnDguYGbvfkcvHULH3w', 'invitation_mode': 'once', 'a
cept': 'auto', 'their_role': 'invitee', 'routing_state': 'none', 'state': 'response'}
    response: <ConnectionResponse(_message_id='3be6191f-ac3f-463f-a5f8-99d47f403cc8', _message_ne
_id=True, _message_decorators=<DecoratorSet{~thread: <ThreadDecorator(_thid='e23a1fe2-0d47-4a2b-9
5b-3b318cb13856', _pthid=None, _sender_order=None, _received_orders=None)>}>, connection=<Connect
onDetail(_did='WzvYnDguYGbvfkcvHULH3w', _did_doc=<DIDDoc did=WzvYnDguYGbvfkcvHULH3w>)>)>
```

SENSOR ACA-py terminal output



```
Created new connection record from invitation
    connection: {'accept': 'auto', 'their_label': 'App', 'invitation_mode': 'once', 'rfc23_state'
 'invitation-received', 'updated_at': '2022-06-16 12:05:57.768699Z', 'state': 'invitation', 'rout
ng_state': 'none', 'connection_id': '0ea2cf46-f548-4346-9994-1916b019f35b', 'their_role': 'invite
', 'created_at': '2022-06-16 12:05:57.768699Z', 'invitation_key': '72ibx4dccbfps3W9eiJQ5qfvrNgnLq
Xc6PYzCAt4uSG'}
    invitation: <ConnectionInvitation(_message_id='5f96e514-2a51-4ee0-9f28-5ea51a35cd8e', _messag
_new_id=False, _message_decorators=<DecoratorSet{}>, label='App', did=None, recipient_keys=['72ib
4dccbfps3W9eiJQ5qfvrNgnLqrXc6PYzCAt4uSG'], endpoint='http://localhost:8001/', routing_keys=None,
mage_url=None)>
    their_label: App

Created connection request
    connection: {'accept': 'auto', 'their_label': 'App', 'invitation_mode': 'once', 'rfc23_state'
 'request-sent', 'updated_at': '2022-06-16 12:05:57.940438Z', 'state': 'request', 'routing_state'
 'none', 'connection_id': '0ea2cf46-f548-4346-9994-1916b019f35b', 'request_id': 'e23a1fe2-0d47-4a
b-945b-3b318cb13856', 'their_role': 'inviter', 'my_did': 'MfpHVq9Y1Ptpbdx9tUXJFH', 'created_at':
2022-06-16 12:05:57.768699Z', 'invitation_key': '72ibx4dccbfps3W9eiJQ5qfvrNgnLqrXc6PYzCAt4uSG'}

Accepted connection response
    connection: {'accept': 'auto', 'their_label': 'App', 'invitation_mode': 'once', 'rfc23_state'
 'response-received', 'updated_at': '2022-06-16 12:05:58.597344Z', 'state': 'response', 'routing_
tate': 'none', 'connection_id': '0ea2cf46-f548-4346-9994-1916b019f35b', 'request_id': 'e23a1fe2-0
47-4a2b-945b-3b318cb13856', 'their_role': 'inviter', 'their_did': 'WzvYnDguYGbvfkcvHULH3w', 'my_d
d': 'MfpHVq9Y1Ptpbdx9tUXJFH', 'created_at': '2022-06-16 12:05:57.768699Z', 'invitation_key': '72i
x4dccbfps3W9eiJQ5qfvrNgnLqrXc6PYzCAt4uSG'}
```

From the sensor output or result, we will save the ***connection_id*** for the next steps.

Now the connection between the agents has been made.

3.3.4. Credential Dance

The credential dance should follow this structure but, as well as the connection, we will skip some steps with the auto flags enabled at the start of the run. This will result in just one call, the send credential offer.

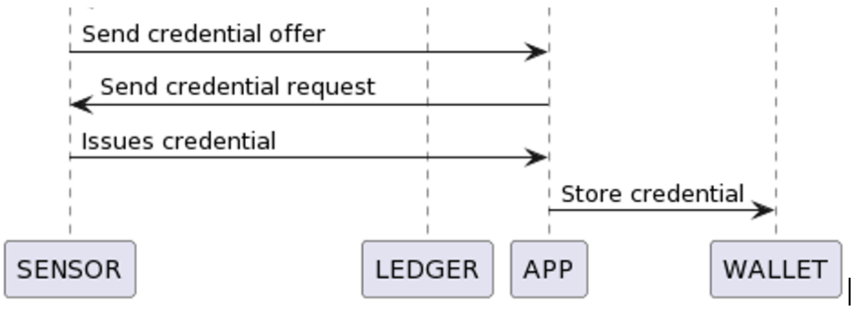


Figure 25 Credential Dance UML diagram

But first we need a schema and a credential definition. If you are using the ACA-py standalone this should be done just once, the schema and credential definition will be stored in your wallet and in the ledger, so you will be able to use them in other executions. Unfortunately, using the docker solution as we are doing here, we will have to create a new schema and credential definition each execution.

Create schema (ISSUER/SENSOR)

The important concept here is the ***attributes***, we have to define the name of the attributes that we will be sending on the next credentials.

```
$ curl -X POST http://localhost:11000/schemas \
  -H 'Content-Type: application/json' \
  -d '{
    "attributes": [
      "state"
    ],
    "schema_name": "SSI-delivery-status",
    "schema_version": "1.0"
}'
```

```
>
{
      "schema_id": "X77te762WrBN4zaiPiscKo:2:SSI-delivery-status:1.0",
      "schema": {
            "ver": "1.0",
            "id": "X77te762WrBN4zaiPiscKo:2:SSI-delivery-status:1.0",
            "name": "SSI-delivery-status",
            "version": "1.0",
            "attrNames": ["state"],
            "seqNo": 15390
      }
}
```

We cannot create a schema that already exists in the ledger. So if we want to use the same schema name with the docker option, we can just change the schema version.

Create credential definition (ISSUER/SENSOR)

Now using the ***schema_id*** generated from the creation of the schema, we can create a credential definition:

```
$ curl -X POST http://localhost:11000/credential-definitions \
  -H 'Content-Type: application/json' \
  -d '{
    "schema_id": "X77te762WrBN4zaiPiscKo:2:SSI-delivery-status:1.0",
    "tag": "default"
  }'
```

```
>
{
	"credential_definition_id":
"X77te762WrBN4zaiPiscKo:3:CL:15390:default"
}
```

Send credential offer (ISSUER/SENSOR)

Now using the ***connection_id*** generated from the creation of the sensor connection, and setting the proper ***attributes*** values, we can create a create a credential offer:

```
$ curl -X POST http://localhost:11000/issue-credential-2.0/send \
 -H "Content-Type: application/json" -d '{
  "auto_remove": true,
  "comment": "Sensor",
  "connection_id": "0ea2cf46-f548-4346-9994-1916b019f35b",
  "credential_preview": {
    "@type": "issue-credential/2.0/credential-preview",
    "attributes": [
      {
        "mime-type": "plain/text",
        "name": "state",
        "value": "delivered"
      }
    ]
  },
  "filter": {
    "dif": {},
    "indy": {}
  },
  "trace": false
}'
```

```
>
{
	"conn_id": "0ea2cf46-f548-4346-9994-1916b019f35b",
	"cred_offer": {
		"@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-
credential/2.0/offer-credential",
		"@id": "d32dd494-3e1d-4cd0-8c21-9aaf67c7fbe5",
		"~thread": {},
		"comment": "create automated v2.0 credential exchange record",
		"credential_preview": {
			"@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-
credential/2.0/credential-preview",
			"attributes": [{
				"name": "state",
                        "mime-type": "plain/text",
                        "value": "delivered"
                }]
        },
        "formats": [{
                "attach_id": "0",
                "format": "hlindy-zkp-v1.0"
        }],
        "offers~attach": [{
                "@id": "0",
                "mime-type": "application/json",
                "data": {
                        "base64": …
                }
        }]
    },
    "initiator": "self",
    "created_at": "2022-06-16 13:37:54.369602Z",
    "auto_remove": true,
    "cred_proposal": {
        "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-
credential/2.0/propose-credential",
        "@id": "f517b177-fc72-4786-8cb8-1712c157c741",
        "comment": "Sensor",
        "credential_preview": {
                "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-
credential/2.0/credential-preview",
                "attributes": [{
                        "name": "state",
                        "mime-type": "plain/text",
                        "value": "delivered"
                }]
        },
        "filters~attach": [{
                "@id": "dif",
                "mime-type": "application/json",
                "data": {
                        "base64": "e30="
                }
        }, {
                "@id": "indy",
                "mime-type": "application/json",
                "data": {
                        "base64": "e30="
                }
        }],
        "formats": [{
                "attach_id": "dif",
                "format": "dif/credential-manifest@v1.0"
        }, {
                "attach_id": "indy",
                "format": "hlindy-zkp-v1.0"
        }]
    },
    "trace": false,
    "auto_offer": false,
    "updated_at": "2022-06-16 13:37:54.369602Z",
    "cred_preview": {
```

```
                "@type": "did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/issue-
credential/2.0/credential-preview",
                "attributes": [{
                        "name": "state",
                        "mime-type": "plain/text",
                        "value": "delivered"
                }]
        },
        "thread_id": "d32dd494-3e1d-4cd0-8c21-9aaf67c7fbe5",
        "cred_ex_id": "9caf6260-a801-4072-bb91-175497d338ab",
        "state": "offer-sent",
        "role": "issuer",
        "auto_issue": true
}
```

3.4. IoT tracking with Raspberry Pi

We will be using a Raspberry Pi 4 Model B.

3.4.1. RFID Sensor

And a PN532 NFC RFID MODULE V3 as the sensor and two mifare cards.

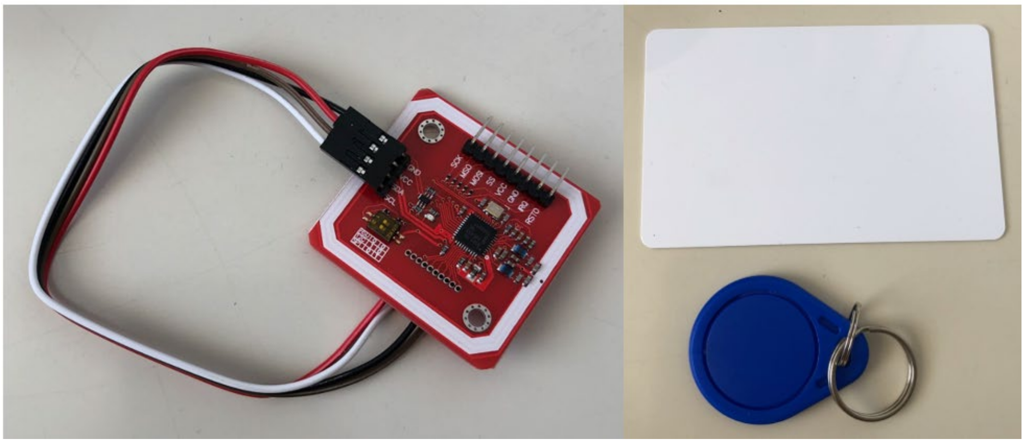


Figure 26 PN532 chip & NFC tags

The connection schema is:

VCC -- 5vDC 04

GND -- GND 06

SCL -- TXD0 08

SDA -- RXD0 10

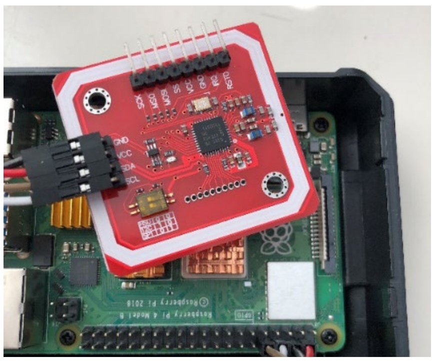


Figure 27 PN532 connection to Raspberry Pi 4 Model B

3.4.2. Raspberry Pi Set-Up

In order to use the NFC chip we have to properly configure the Raspberry Pi.

First we have to activate the serial connection:

```
$ sudo raspi-config
```

Then go to Advanced Options/Serial and set No to "Would you like a login shell to be accessible over serial?", valid and reboot.

Sometimes UART is disabled in kernel. To enable it you need to check in /boot/config.txt the line enable_uart

If it's set to 0 like follow:

```
enable_uart=0
```

You need to change to 1 as follow

```
enable_uart=1
```

And reboot.

With Raspberry PI version 3, things have changed, the real UART (Serial) is now affected to Bluetooth hardware module and the old one is now managed by software. Old serial on PI3 is /dev/ttyS0 and no more /dev/ttyAMA0 because this one is connected to Bluetooth.

It seems in the latest Jessie version there is a /dev/serial0 so you should replace /dev/ttyAMA0 with /dev/serial0 to use as before.

But, as it's software managed, this serial interface has a number of drawbacks.

- There is no support for parity
- Speed is limited (it's software managed)
- If your CPU is under heavy load it could corrupt the data especially at high speeds.

So if you really need reliable Serial on /dev/ttyAM0 (as before) I strongly suggest to use the hardware one.

To disable bluethooth and set /dev/ttyAM0 to real UART (as before), edit the file /boot/config.txt, add the following line at the end and reboot:

```
dtoverlay=pi3-disable-bt
```

3.5. Final Implementation

For this final implementation part, we will explain every part as the project should be, but unfortunately I have not been able to implement a solution to the communication from the Raspberry Pi to the App because I do not have permission to access the network router, which results in a rejected call as I try to make a http request to the supposed ACA-py public ip.

The implementation done for the App part is made to manually simulate the sensor.

We will also assume in the explanations that we have already configured and started the Raspberry Pi, sensor and ACA-py instances.

3.5.1. Application (HOLDER)

In this application we will combine a graphical interface with the functionalities and concepts of the SSI context, using the ACA-py endpoints available.

3.5.1.1. GUI App

For the graphical part of the application we will use the Java Swing Library. I will not explain how does Swing work because is not the goal of this project. There are a lot of documentation, tutorials and courses on the internet.

It consists in:

- Log in panel with an entry for the wallet key.
- Action Menu panel with a button to create the connection invitation, a button to fetch the credentials stored in the wallet and a text area where the credentials will be shown.

3.5.1.2. Features

To add a little layer of security, a log in will first show up at running the application. It will ask you to insert the wallet key. In our case is “secret”, but if you do not exactly write it an “Incorrect key” message will pop up.

From behind it just compares the string entered with the key already defined in the code.

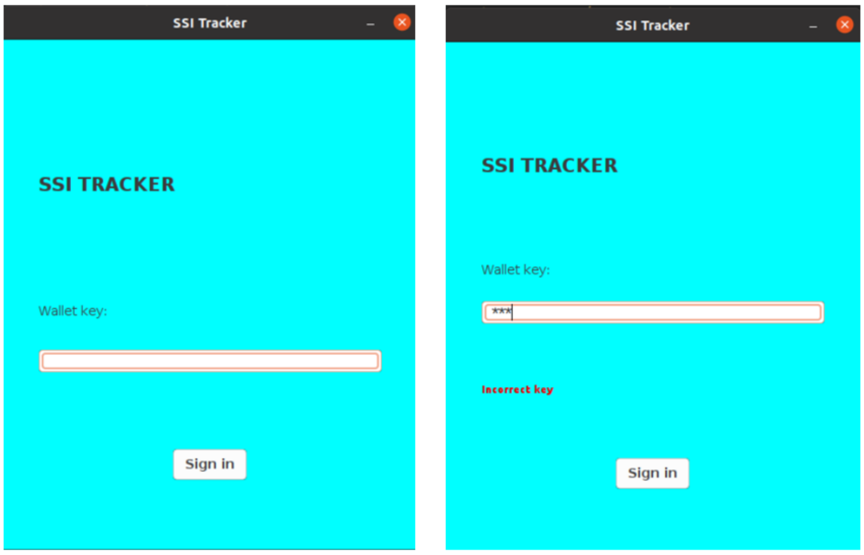


Figure 28 App Log In Menu & Error Handling

As soon as you insert the correct key, you will be sent to the action menu.

When pressing the "Create Connection Invitation" it should call to the ***/connections/create-invitation*** endpoint of the app ACA-py instance, then save the ***invitation*** field content of the output to the server accessible for the sensor.

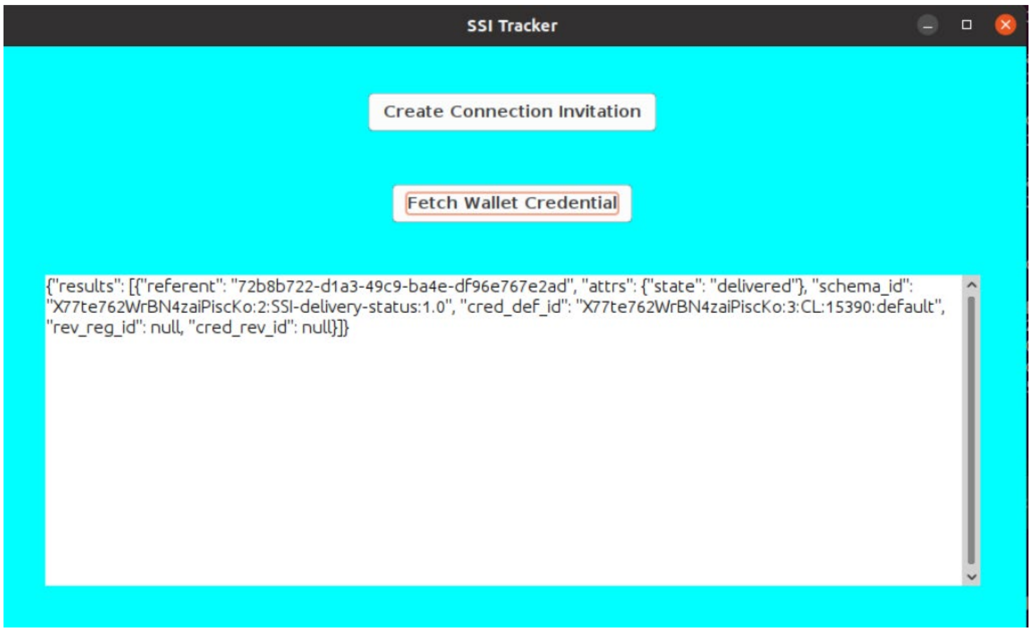
But in this case as this part of the server is not implemented, it will just print it in the terminal of the execution in order to copy it and use it in the manually simulation of the sensor.

```
{
       "@type":
"did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/connections/1.0/invitation",
       "@id": "5f96e514-2a51-4ee0-9f28-5ea51a35cd8e",
       "label": "App",
       "recipientKeys": ["72ibx4dccbfps3W9eiJQ5qfvrNgnLqrXc6PYzCAt4uSG"],
       "serviceEndpoint": "http://localhost:8001/"
}
```

Once the connection is made and the sensor sends a credential, we can press the "Fetch Wallet Credential" it should call to the ***/credentials*** endpoint of the app ACA-py instance, then print the content of the output to the text area.

You will be able to see the status value in one of the attributes from the credential.

Also, this method fetches all the available credentials stored in the wallet, so in the result may be multiple accumulated.



```
{"results": [{"referent": "72b8b722-d1a3-49c9-ba4e-df96e767e2ad", "attrs": {"state": "delivered"}, "schema_id":
"X77te762WrBN4zaiPiscKo:2:SSI-delivery-status:1.0", "cred_def_id": "X77te762WrBN4zaiPiscKo:3:CL:15390:default",
"rev_reg_id": null, "cred_rev_id": null}]}
```

Figure 29 App Action Menu; Credential Fetched

3.5.2. Sensor (ISSUER)

Now will combine the RFID UID reading of the NFC part with the functionalities and concepts of the SSI context, using the ACA-py endpoints available.

We will assume that a public DID, schema and credential definition has been already created, that they are in the ledger and the wallet has been properly configurated for the sensor ACA-py agent.

Everything will be written in Ruby language.

3.5.2.1. Ruby-NFC and dependencies set-up

To work with the PN532 sensor we will be using an NFC gem called ruby-nfc. This gem brings some NFC functionality for Ruby programming language. It allows you to:

- Read and modify contents of Mifare tags
- Send APDU commands to Android HCE or Blackberry VTE devices
- Communicate with dual-interface smart cards like Master Card PayPass or Visa payWave cards

Prerequisites

Install Ruby:

```
$ sudo apt install ruby-dev
```

Install dependencies and libnfc:

```
$ sudo apt-get install libusb-dev
$ sudo apt-get install autoconf automake git libtool libssl-dev pkg-config
$ sudo apt install libfreefare-dev
$ sudo apt install libnfc5 libnfc-bin libnfc-examples
```

Then we have add to the /etc/nfc/libnfc.conf file the following to specify which device we will use and what type of connection.

```
device.name = "pn532 UART"
device.connstring = "pn532_uart:/dev/ttyAMA0"
```

Installation

Now install the needed gems for the final script:

```
$ sudo gem install ruby-nfc
$ sudo gem install json
```

3.5.2.2. Features

The Ruby script has two different parts, one to create the connection with the App and the other to send credentials.

Connection

For this we will imagine that it is the first time we run it.

1) It will go to the public server from the App to get the JSON of the invitation.
2) It will call to the public ***/connections/receive_invitation*** endpoint of its ACA-py agent passing as body the invitation.
3) The connection should be created and we will store the ***connection_id*** of the output for the future.

Credential sending

Once the connection is done:

1) It will wait until reads an NFC Tag and save the UID value.
2) Depending on the UID value will set the status to "in_delivery" or "delivered".
3) It will create the credential offer body in JSON format, setting the ***status*** and ***connection_id***.
4) It will call to the public ***/issue-credential-2.0/send*** endpoint of its ACA-py agent with the body created.
5) The credential should be now stored on the App wallet.

4. **Budget**

The budget for this project only includes the number of hours you have dedicated to the thesis, evaluated at cost of junior engineer, since everything used for the project development or courses taken were free.

	Total Hours	Price (Brute salary)	Social Cost	Total Cost
Junior Software engineer	20h x week x 16 weeks= 320 h	9€ x hour	1,30 x worker	3744€

5. Conclusions and future development:

In conclusion, this essay has covered the concept that refers to sending IoT data using SSI Blockchain technology, from a more theorical part, which explained the main concepts of Blockchain and SSI, from a more practical part providing a guide to send verifiable credentials from an IoT device, a Raspberry Pi with an RFID sensor, using Hyperledger Aries Cloud Agent with an Hyperledger Indy public network.

We have been able to demonstrate in a real case how this technology could be used as a solution to identification of IoT devices. Not only that, we managed also to send data in the same SSI ecosystem using the verifiable credentials.

There are many different options for implementing an SSI system, for example using a Blockchain as Ethereum, but it should be written from scratch. Which means creating the proper smart contracts to manage the DIDs, the credentials, the verifier, etc,. This is truly a complex thing and not a really good option to deal with scalability, with environment adaptation, and you are tied to the Blockchain project properties changes.

This is why Hyperledger is a perfect solution to this. It brings a Blockchain adaptation for each kind of system. In this case the Identity Stack is a clear example of it.

With Indy we have the optimized solution for SSI network, and with Aries we have the solution to interact with the network.

Extrapolating to the IoT context, being certain of that your device message is really your device who has sent it, is a big step improving security. Not only that, a great amount of devices identifying is also well managed. Furthermore, since IoT devices could have a big amount of interactions in a small period of time, the solution of Aries to bring a secure channel in the connection between agents could have a great impact in the system efficiency.

Certainly, it will take more time to bring this to a real production state because this is a very new technology which is still in progress, but the potential is amazing.

In a future development, exploring the part of the verifier would be an important point to see. The verifier is the responsible of requesting proofs to the holder and verify that what it is said in the credential is true. Here, we are relying in the Aries connection and that we are the ones that have done everything. But in a real implementation you may not have this confidence with every part of the equation, here is when the verifier comes in.

Finally, it could be interesting to mention that this thesis will be submitted as a paper to The ERK 2022 conference organized by the IEEE Slovenia Section together with Faculty of Electrical Engineering University of Ljubljana and other Slovenian professional societies.

Bibliography:

[1] "BCovrin Dev Indy Network." Accessed June 6, 2022. http://dev.bcovrin.vonx.io/.

[2] Chechel, Max. About Ruby-Nfc. Ruby, 2021. https://github.com/hexdigest/ruby-nfc.

[3] "Enable Serial Port on Raspberry Pi – Charles's Blog." Accessed June 16, 2022. https://hallard.me/enable-serial-port-on-raspberry-pi/.

[4] "Get a Sovrin Network Endorser!" Accessed June 6, 2022. https://selfserve.sovrin.org/.

[5] GitHub. "How to Install ACA PY 0.6.0 · 5GZORRO/Identity Wiki." Accessed June 6, 2022. https://github.com/5GZORRO/identity.

[6] Hyperledger Aries Cloud Agent - Python. Python. 2019. Reprint, Hyperledger, 2022. https://github.com/hyperledger/aries-cloudagent-python/blob/00d97b3e0e6f713dfab383eb2e5e14e58472a47d/demo/AriesOpenAPIDemo.md.

[7] Hyperledger Aries Cloud Agent - Python. Python. 2019. Reprint, Hyperledger, 2022. https://github.com/hyperledger/aries-cloudagent-python/blob/00d97b3e0e6f713dfab383eb2e5e14e58472a47d/DevReadMe.md.

[8] Indy Node. Python. 2016. Reprint, Hyperledger, 2022. https://github.com/hyperledger/indy-node/blob/4947fc7b2e2a2f45bda76da01be2c0290e3141bc/docs/source/requests.md.

[9] Indy Node. Python. 2016. Reprint, Hyperledger, 2022. https://github.com/hyperledger/indy-node/blob/4947fc7b2e2a2f45bda76da01be2c0290e3141bc/design/txn_author_agreement.md.

[10] Indy SDK. Rust. 2017. Reprint, Hyperledger, 2022. https://github.com/hyperledger/indy-sdk/blob/1c7096dd95d0fd53881070f66907df4b9e61b874/docs/how-tos/transaction-author-agreement.md.

[11] "Indy Transaction Explorer." Accessed June 6, 2022. https://indyscan.io.

[12] Docker Documentation. "Install Docker Engine on Ubuntu," May 26, 2022. https://docs.docker.com/engine/install/ubuntu/.

[13] Jong, Laurence de. "Becoming a Hyperledger Aries Developer - Getting Started." Laurence de Jong, March 11, 2021. https://ldej.nl/post/becoming-a-hyperledger-aries-developer-getting-started/.

[14] "Connecting ACA-Py to Development Ledgers." Laurence de Jong, October 18, 2020. https://ldej.nl/post/connecting-acapy-to-development-ledgers/.

[15] GitHub. "Ledger and Wallet Config Updates; Add Support for Transaction Author Agreements by Andrewwhitehead · Pull Request #127 · Hyperledger/Aries-Cloudagent-Python." Accessed June 6, 2022. https://github.com/hyperledger/aries-cloudagent-python/pull/127.

[16] "PN532 NFC RFID Module V4 - ELECHOUSE." Accessed June 15, 2022. https://www.elechouse.com/product/pn532-nfc-rfid-module-v4/.

[17] Sovrin. "Preparing for the Sovrin Transaction Author Agreement," June 27, 2019. https://sovrin.org/preparing-for-the-sovrin-transaction-author-agreement/.

[18] "Introduction to Hyperledger Sovereign Identity Blockchain Solutions: Indy, Aries & Ursa." LearnThings.Online (blog), March 5, 2020. https://learnthings.online/other/2020/03/05/introduction-to-hyperledger-sovereign-identity-blockchain-solutions-indy-aries-ursa.

[19] "Setting up a PN532 NFC Module on a Raspberry Pi Using I2C." Accessed June 16, 2022. https://blog.stigok.com/2017/10/12/setting-up-a-pn532-nfc-module-on-a-raspberry-pi-using-i2c.html.

[20] GitHub. "Sovrin-Foundation/Sovrin at Stable." Accessed June 6, 2022. https://github.com/sovrin-foundation/sovrin.

[21] HL Education Git. JavaScript. 2017. Reprint, Hyperledger Project Archive, 2022. https://github.com/hyperledger-archives/education/blob/ce553cd3444a187e1c624ccc6ea6aae8b46ded26/LFS171x/docs/discovering-blockchain-technologies.md.

[22] "Hyperledger — Chapter 1 | Blockchain Foundation." Accessed May 8, 2022. https://www.linkedin.com/pulse/hyperledger-chapter-1-blockchain-foundation-moses-sam-paul-johnraj.

[23] "Introduction to Hyperledger Sovereign Identity Blockchain Solutions: Indy, Aries & Ursa." LearnThings.Online (blog), March 5, 2020. https://learnthings.online/other/2020/03/05/introduction-to-hyperledger-sovereign-identity-blockchain-solutions-indy-aries-ursa.

Appendices:

AcaPyService.java

```
import okhttp3.*;
import org.json.JSONObject;
import java.io.IOException;

public class AcaPyService {

    private static final String ACA_PY_ENDPOINT = "http://localhost:11001";
    private static final String KEY = "secret";
    public static final MediaType JSON
            = MediaType.get("application/json; charset=utf-8");

    public void start(String key) throws RuntimeException{

        if (!key.equals(KEY)) throw new RuntimeException();

    }

    // Calls to the proper ACA-PY endpoint to create invitation
    // and saves the result in the server
    public void createConnectionInvitation() throws IOException{

        OkHttpClient client = new OkHttpClient();

        RequestBody body = RequestBody.create("{}", JSON);
        Request request = new Request.Builder()
                .url(ACA_PY_ENDPOINT+"/connections/create-invitation")
                .post(body)
                .build();
        try (Response response = client.newCall(request).execute()) {
            String bodyString = response.body().string();
            JSONObject jsonObject = new JSONObject(bodyString);
            System.out.println(jsonObject.get("invitation"));
            //save that json into the server
        }
    }

    // Calls to the proper ACA-PY endpoint to fetch the wallet
    // credentials and return the last one
    public String fetchWalletCredential() throws Exception{

        OkHttpClient client = new OkHttpClient();

        Request request = new Request.Builder()
                .url(ACA_PY_ENDPOINT+"/credentials")
                .get()
                .build();
        try (Response response = client.newCall(request).execute()) {
            String bodyString = response.body().string();
            System.out.println(bodyString);
            return bodyString;
        }
    }
}
```

ActionPanel.java

```
import javax.swing.*;
import java.awt.*;

public class ActionPanel extends JPanel {
    // Define the layout configurations
    private GridBagConstraints constraints;

    // Define components
    private JButton createInvButton;
    private JButton fetchCredentialButton;
    private JTextArea credTextArea;
    private JLabel logoLabel;

    public ActionPanel() {
        super(new GridBagLayout());
        constraints = new GridBagConstraints();

        // Create components
        createInvButton = new JButton("Create Connection Invitation");
        fetchCredentialButton = new JButton("Fetch Wallet Credential");
        credTextArea = new JTextArea();

        // Creating and scaling the logo
        logoLabel = new JLabel();

        setupGUI();
    }

    public void setupGUI(){
        setBackground(Color.CYAN);

        constraints.gridx = 0;
        constraints.gridy = 0;
        constraints.weightx = 1.0;
        constraints.gridwidth = 2;
        constraints.fill = GridBagConstraints.NONE;
        constraints.anchor = GridBagConstraints.CENTER;
        constraints.insets = new Insets(40, 40, 40, 40);
        createInvButton.setFont(new java.awt.Font("Calibri", Font.BOLD,14));
        add(createInvButton, constraints);

        constraints.gridy = 1;
        constraints.gridwidth = 2;
        constraints.fill = GridBagConstraints.NONE;
        constraints.anchor = GridBagConstraints.CENTER;
        constraints.insets = new Insets(5, 40, 40, 40);
        fetchCredentialButton.setFont(new java.awt.Font("Calibri",
Font.BOLD,14));
        add(fetchCredentialButton, constraints);

        constraints.gridy = 2;
        constraints.gridheight = 5;
        constraints.gridwidth = 3;
        constraints.weightx = 1.0;
        constraints.weighty = 1.0;
        constraints.anchor = GridBagConstraints.CENTER;
        constraints.fill = GridBagConstraints.BOTH;
        constraints.insets = new Insets(5, 40, 40, 40);
        credTextArea.setEditable(false);
        credTextArea.setLineWrap(true);
        credTextArea.setWrapStyleWord(true);
        JScrollPane scrollPane = new JScrollPane(credTextArea);

scrollPane.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_A
LWAYS);

scrollPane.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLB
AR_NEVER);
        add(scrollPane, constraints);
    }
    public JButton getCreateInvButton() { return createInvButton; }
    public JButton getFetchCredentialButton() { return fetchCredentialButton;
}
    public JTextArea getCredTextArea() { return credTextArea; }
}
```

App.java

```
import javax.swing.*;

public class App {
    public static void main(String[] args) {

        try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception ex) {
            ex.printStackTrace();
        }

        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                Controller app = new Controller();
            }
        });
    }
}
```

Controller.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Properties;

public class Controller {
    // Define the window
    private JFrame frame;

    // Define the "container" panel of the "pages"
    private JPanel appPanel;

    // Define the different "pages"
    private LoginPanel loginPanel;
    private ActionPanel actionPanel;

    //Define ACA-PY service
    private AcaPyService acaPyService;

    // Define the top-level pages manager
    private CardLayout appCL;

    // Define the properties
    private Properties prop;

    public Controller() {

        // Create the CardLayout
        appCL = new CardLayout();

        // Create the first panel
        frame = new JFrame("SSI Tracker");
        appPanel = new JPanel(appCL);

        // Create the properties
        prop = new Properties();

        acaPyService = new AcaPyService();

        setupGUI();

        frame.pack();
        frame.setVisible(true);
    }

    public void setupGUI() {
        // Set up the the login menu
        setupLogin();

        // Set up the window.
        frame.add(appPanel);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        frame.setVisible(true);

    }

    public void setupLogin(){
        // Create login menu and configure it
        loginPanel = new LoginPanel();
        appPanel.add(loginPanel, "1");
        appCL.show(appPanel, "1");
        appPanel.validate();
        appPanel.repaint();
        frame.setResizable(false);
        Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
        dim = new Dimension(dim.width/4,dim.height/2);
        frame.setSize(dim);
        frame.setLocationRelativeTo(null);
        appPanel.setPreferredSize(dim);
        loginPanel.setPreferredSize(dim);

        JPasswordField keyField = loginPanel.getKeyField();
        JButton loginButton = loginPanel.getLoginButton();

        // When Login button pressed check the credentials to go next
        loginButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    // The getPassword from JPasswordField returns a char
array for security purpose
                    String key = new String(keyField.getPassword());

                    // Try to connect
                    acaPyService.start(key);

                    setupActionMenu();

                } catch (Exception ex) {
                    // When wrong username or password
                    loginPanel.authFailed();
                    loginPanel.revalidate();
                    frame.pack();
                }
            }
        });

        // When in Password field hits Enter perform button Login pressed
        keyField.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                loginButton.doClick();
            }
        });
    }

    public void setupActionMenu(){
        // Create mailBox menu and configure it
        actionPanel = new ActionPanel();
        appPanel.add(actionPanel, "2");
        appPanel.validate();
        appPanel.repaint();
        appCL.show(appPanel, "2");
        frame.setResizable(true);
        Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
        dim = new Dimension(dim.width/2,dim.height/2);
        frame.setSize(dim);
        frame.setLocationRelativeTo(null);
        actionPanel.setPreferredSize(dim);

        JButton createInvButton = actionPanel.getCreateInvButton();
        JButton fetchCredentialButton =
actionPanel.getFetchCredentialButton();
        JTextArea credTextArea = actionPanel.getCredTextArea();

        // When create connection invitation button pressed calls
        createInvButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    acaPyService.createConnectionInvitation();

                } catch (Exception ex) {
                    ex.printStackTrace();
                    JOptionPane.showMessageDialog(null,
                            "Something went wrong while creating the
invitation",
                            "Connection error",
                            JOptionPane.INFORMATION_MESSAGE);
                }
            }
        });

        // When create connection invitation button pressed calls
        fetchCredentialButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    String cred = acaPyService.fetchWalletCredential();
                    credTextArea.setText(cred);

                } catch (Exception exception) {
                    exception.printStackTrace();
                    JOptionPane.showMessageDialog(null,
                            "Something went wrong while fetching the
credential",
                            "Wallet error",
                            JOptionPane.INFORMATION_MESSAGE);
                }
            }
        });

    }

}
```

LoginPanel.java

```
import javax.swing.*;
import java.awt.*;


public class LoginPanel extends JPanel {
    // Define the layout configurations
    private GridBagConstraints constraints;

    // Define components
    private JPasswordField keyField;
    private JLabel titleLabel;
    private JLabel logoLabel;
    private JLabel passwordLabel;
    private JButton loginButton;
    private JLabel errorLabel;

    public LoginPanel() {
        // The login panel will be managed by GridBagLayout
        super(new GridBagLayout());
        constraints = new GridBagConstraints();

        // Create components
        keyField = new JPasswordField(25);
        passwordLabel = new JLabel("Wallet key:");
        titleLabel = new JLabel("SSI TRACKER");
        loginButton = new JButton("Sign in");
        errorLabel = new JLabel();

        // Creating and scaling the logo
        logoLabel = new JLabel();

        setupGUI();
    }

    public void setupGUI(){
        setBackground(Color.CYAN);

        // Add the components to the panel
        constraints.weightx = 1.0;
        constraints.weighty = 1.0;
        constraints.fill = GridBagConstraints.BOTH;
        constraints.gridx = 0;
        constraints.gridy = 0;
        constraints.anchor = GridBagConstraints.CENTER;
        constraints.insets = new Insets(40, 40, 2, 40);
        add(logoLabel, constraints);

        constraints.gridy = 1;
        titleLabel.setFont(new java.awt.Font("Calibri", Font.BOLD,20));
        constraints.insets = new Insets(2, 40, 10, 40);
        add(titleLabel, constraints);

        constraints.gridy = 2;
        constraints.anchor = GridBagConstraints.SOUTHWEST;
        constraints.fill = GridBagConstraints.NONE;
        passwordLabel.setFont(new java.awt.Font("Calibri", Font.PLAIN,14));
        constraints.insets = new Insets(5, 40, 0, 40);
        add(passwordLabel, constraints);

        constraints.gridy = 3;
        constraints.anchor = GridBagConstraints.CENTER;
        constraints.fill = GridBagConstraints.HORIZONTAL;
        constraints.insets = new Insets(0, 40, 10, 40);
        add(keyField, constraints);

        constraints.gridy = 4;
        constraints.gridwidth = 2;
        constraints.fill = GridBagConstraints.NONE;
        constraints.anchor = GridBagConstraints.CENTER;
        constraints.insets = new Insets(5, 40, 40, 40);
        loginButton.setFont(new java.awt.Font("Calibri", Font.BOLD,14));
        add(loginButton, constraints);
    }

    public void authFailed(){
        errorLabel.setText("<html><font color=red size=3><b>" +
                "Incorrect key</b></html>");

        constraints.gridy = 4;
        constraints.anchor = GridBagConstraints.WEST;
        constraints.insets = new Insets(2, 40, 10, 40);
        add(errorLabel, constraints);

        constraints.gridy = 5;
        constraints.gridwidth = 2;
        constraints.anchor = GridBagConstraints.CENTER;
        constraints.insets = new Insets(5, 40, 40, 40);
        add(loginButton, constraints);
    }


    public JPasswordField getKeyField() {
        return keyField;
    }
    public JButton getLoginButton() {
        return loginButton;
    }

}
```

Pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>TFG-app</artifactId>
    <version>1.0-SNAPSHOT</version>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>11</source>
                    <target>11</target>
                </configuration>
            </plugin>
        </plugins>
    </build>

    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
    </properties>

    <dependencies>
        <!-- https://mvnrepository.com/artifact/org.json/json -->
        <dependency>
            <groupId>org.json</groupId>
            <artifactId>json</artifactId>
            <version>20220320</version>
        </dependency>
        <dependency>
            <groupId>com.squareup.okhttp3</groupId>
            <artifactId>okhttp</artifactId>
            <version>4.9.1</version>
        </dependency>
    </dependencies>
</project>
```

rfid-uid.rb

```
require 'ruby-nfc'

class Rfid_PN532

    #BLOQUEANTE
    #Espera a que el xip NFC lea una tarjeta del tipo Mifare
    #y retorna su uid en hexadecimal y mayusculas
    def read_uid
        readers = NFC::Reader.all
        readers[0].poll(Mifare::Classic::Tag) do |tag|
            begin
                uid = "#{tag.uid_hex}"
                return uid.upcase
            end
        end
    end
end
```

sensor.rb

```
require 'json'

require_relative 'rfid-uid'

class Sensor

    APP_SERVER_IP = "ip:port"
    ACA_PY_IP = "ip:port"

    connection_id = ""

    rfid = Rfid_PN532.new

    #CREATE CONNECTION WITH APP _______________________________________
    #1. Gets the connection invitation JSON from the server app
    #2. Extracts the invitation object and adds it to the http body
    #3. Calls the ACA_PY API to receive and accept the invitation
    #4. Extracts from result: connection_id
    #__________________________________________________________________
    def create_connection
        invitation_json = `curl "http://#{APP_SERVER_IP}/invitation"`
        if !invitation_json.include? "reqSignature"
            raise "ERROR CONNECTING TO THE SERVER\n\n #{invitation_json}"
        end
        invitation = JSON.parse(invitation_json)['invitation']
        invitation_res = `curl -X POST
"http://#{ACA_PY_IP}/connections/receive_invitation" -H 'Contetnt-Type:
application/json' -d '#{invitation}'`
        if !invitation_res.include? "connection_id"
            rise "ERROR CONNECTING TO THE ACA-PY\n\n #{invitation_res}"
        end
        connection_id = JSON.parse(invitation_res)['connection_id']
    end

    def wait_for_tag
        return rfid.read_uid
    end

    def states (uid)
        case uid
            when "D5D8E923"
                return "in_delivery"

            when "29E88DC1"
                return "delivered"
        end
    end

    #SEND CREDENTIAL OFFER TO THE APP __________________________________
    #1. Adds to the http body the connection_id and the state
    #2. Calls the ACA_PY API to send the credential offer
    #___________________________________________________________________
    def send_credential_offer (state)
        body = "{\"auto_remove\": true, \"comment\": \"Sensor\",
\"connection_id\": \"#{connection_id}\", \"credential_preview\": {\"@type\":
\"issue-credential/2.0/credential-preview\", \"attributes\": [{\"mime-type\":
\"plain/text\", \"name\": \"state\", \"value\": \"#{state}\"}]}, \"filter\":
{\"dif\": {}, \"indy\": {}}, \"trace\": false }"
        offer_result = `curl -X POST "http://#{ACA_PY_IP}/issue-credential-
2.0/send" -H 'Contetnt-Type: application/json' -d '#{body}'`
        if !invitation_res.include? "connection_id"
            rise "ERROR SENDING THE OFFER\n\n #{offer_result}"
        end
    end
end

if __FILE__ == $0
    sensor = Sensor.new

    loop do
        begin
            sensor.create_connection
            break
        rescue RuntimeError => e
            print_exception(e, true)
        end
    end

    loop do
        uid = sensor.wait_for_tag
        loop do
            begin
                sensor.send_credential_offer(sensor.states(uid))
                break
            rescue RuntimeError => e
                print_exception(e, true)
            end
        end
    end
end
```

Glossary

DLT	Distributed Ledger Technologies
ACA	Aries Cloud Agent
ACA-py	Aries Cloud Agent Python
API	Application Programming Interface
UID	Unique Identifier
DID	Decentralized Identifier
TAA	Transaction Author Agreement
URL	Uniform Resource Locator
SDK	Software Development Kit
RFID	Radio Frequency Identification
NFC	Near-Field Communication
IoT	Internet of Things
SSI	Self Sovereign Identity
ZKP	Zero-Knowledge Proof
ToIP	Trust Over IP
QR	Quick Response