ODIN BJØRNEBO
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Decentralized Identity for Industrial Applications

Master's Thesis - Computer Science - June 2022

```
func (m *Manager) NewConfiguration(opts ...gorums.ConfigOption) (c *Configuration, err error) {
	if len(opts) < 1 || len(opts) > 2 {
		return nil, fmt.Errorf("wrong number of options: %d", len(opts))
	}
	c = &Configuration{}
	for _, opt := range opts {
		switch v := opt.(type) {
		case gorums.NodeListOption:
			c.Configuration, err = gorums.NewConfiguration(m.Manager, v)
			if err ≠ nil {
				return nil, err
			}
		case QuorumSpec:
			// Must be last since v may match QuorumSpec if it is interface{}
			c.qspec = v
		default:
			return nil, fmt.Errorf("unknown option type: %v", v)
		}
	}
	// return an error if the QuorumSpec interface is not empty and no implementati
	var test interface{} = struct{}{}
	if _, empty := test.(QuorumSpec); !empty && c.qspec == nil {
		return nil, fmt.Errorf("missing required QuorumSpec")
	}
	return c, nil
}
```

I, **Odin Bjørnebo**, declare that this thesis titled, "Decentralized Identity for Industrial Applications" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a master's degree at the University of Stavanger.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

“Programming is a nice break from thinking.”

– Leslie Lamport

Abstract

This thesis looks at some aspects of current authorization to applications, then explores new ways of solving this with emerging technology originated from the ever expanding field of blockchain. This accumulates to an architecture that could work for organizations that want to work together. This architecture is tried implemented, then drafts the outcome of the process.

Acknowledgements

I would like to thank my supervisors, Ron Chunming and Jiahui Geng for their enthusiasm and help with this project. Lastly, I would also thank the University of Stavanger for allowing me to work with such an interesting topic.

Contents

Acronyms

2FA Two Factor Authentication. 4, 20

ACA-Py Aries Cloud Agent - Python. 19, 20, 23, 25

API Application Programming Interface. 20, 23

DID Decentralized Identifiers. 12–14, 17, 19

ION Identity Overlay Network. 14

IPFS InterPlanetary File System. 14

OS Operating System. 9

PKI Public Key Infrastructure. 18

SSI Self-Sovereign Identity. 12, 17

VM Virtual Machine. 9

Chapter 1

Introduction

In recent years, blockchain technology has gained more and more interest from all around the globe. This technology is evolving beyond the original use case of a decentralized alternative to the banking system. Blockchain technology brings other possible use cases, especially for cooperation across organizations, and for individuals to control their data. This is mainly due to decentralized systems that can provide protection, trust and privacy in new ways that conventional solutions do not. Verifying users' identity is vital to ensure that every application today operates properly. This thesis will take a look at using new blockchain technology to authorize users.

1.1 Background and Motivation

Most parts of the world uses a username and password combination to authorize themselves to services. This information is usually stored at the service provider or in the cloud, and not in the hands of the user. Therefore, the information is constantly vulnerable to breaches and cyber attacks. In addition to this, an increase of cyber related attacks are seen in recent years. Because of this, users are more vulnerable to digital identity theft. Solutions to this exist in the form of Two Factor Authentication (2FA), but some say that even two factors of authorizations is not enough.

1.2 Objectives

This thesis focuses on designing a system for organizations to allow users to cooperate across organizations in addition to sharing resources. The focus is not on the actual system itself but defining an underlying authentication architecture that lets users trust each other. The objectives of this thesis are:

- Design a decentralized identity system
- Implement and test said system
- Integrate the system with the rest of partner thesis

1.3 Approach and Contributions

The approach taken to solve these objectives were to conduct a literature study on the different technologies available, in addition to the research being done in this area. Afterwards a consultation with the supervisors of this thesis, and the rest of the project team followed. This was done in order to choose a relevant design. Following that, experimentation with different tool kits that the projects came with, was executed. Finishing up, the design was altered and a new setup for a development environment that could tie the different components together, was started. The relevant components are shown below:

- System Architecture
- Controller API for talking to agent
- Infrastructure to setup development environment

Then after this a reflection in regards to what went good, what went wrong and how it can be improved upon later on was conducted.

1.4 Outline

In this section, information about what each of the chapters will contain, is addressed.

Chapter 1

The first chapter gives some general introduction to the objective of this thesis, and why this innovation is important to the industry.

Chapter 2

Chapter two talks about some of the essential concepts and technologies that are used in this thesis. This includes containerizing of applications, but most importantly some of the most important aspects of blockchain technology that is useful for understanding the rest of the thesis.

Chapter 3

This chapter looks at some of the existing solutions for conventional applications that deal with data sharing. In addition to examining the existing solutions for identity solutions based on blockchain technology, and how they differ.

Chapter 4

This chapter shows the choices made for designing the solution and the reasoning behind those choices. This chapter ends with an overview of the proposed architecture.

Chapter 5

Chapter five goes into the different technologies used for the implementation. What the solution looks like in terms of software and how the solution operates.

Chapter 6

In the sixth chapter, a discussion takes place to look into what went wrong, and what worked well in this project, and why that is.

1.4.1 Chapter 7

The final chapter discusses the future work opportunities that are present, what kind of testing use cases that would be relevant. It will also look at what is needed to tie it together with the rest.

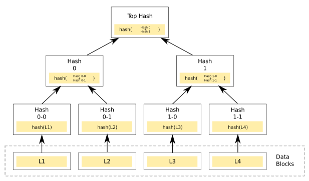
Chapter 2

Background

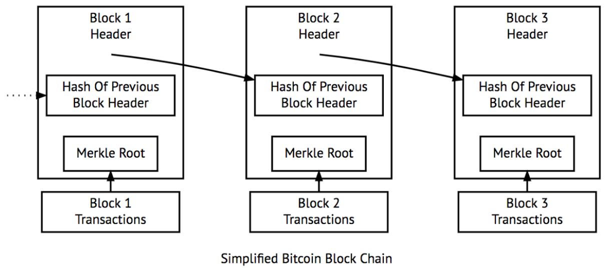
2.1 Blockchain Technology

The first decentralized blockchain system was proposed [1] by Satoshi Nakamoto in 2008, and has been what almost everyone associates with the word blockchain. A cryptocurrency that solves the double spend issue with an ever growing public ledger with all the transactions.

However the technology is more than just a currency. The word *block-chain* in itself points to two aspects. A chain of blocks, a block, in this context is a data structure that consists of some transactions and a header. What is contained in the block header depends on what ledger it is part of, one of these attributes are usually the hash of a Merkle tree that is used for proving that a transaction is part of a block. However all block headers contain a hash of the previous block. This makes the structure a number of blocks that point the previous one, hence the word *block-chain*.



(a) Merkle tree illustration [2]



(b) Blockchain data structure illustration[3]

Figure 2.1: Some illustrations about blockchain datastructures

There are however more nuances to blockchain technology than the data structure, it is also the concept of a decentralized distributed ledger that is resistant to tampering. This is accomplished with a consensus algorithm maintained by all or some of the nodes that participates. Examples of such consensus algorithms are proof of work, proof of stake, and practical byzantine fault tolerance. All these algorithms bring their own advantages and disadvantages, but using them helps the ledger stay immutable.

The distributed part usually depends on technology, but most blockchain protocols require the nodes taking part by storing their own copy of the ledger. The ledger usually contains transactions tracking ownership of assets, this could be anything from a currency to an intellectual property like copyrights. These factors together contribute in making the ledger resistant towards tampering, transparent and publicly accessible. Not all ledgers are publicly accessible, some ledgers are permissioned, and others are part of a federation, this is just the accessibility of the ledger, and usually denotes who can submit transactions or read from the ledger. A federation can for example be between a set amount of companies that writes to the same ledger.

2.2 Containerizing

Then unto to containerizing, which will be used in the implementation, so it's best to look at what the technology is.

Containerizing is a method to virtualize the Operating System (OS) of a computer. This is different to virtualizing the a whole computer. In this section the differences will be examined. A Virtual Machine (VM) is used to virtualize a whole computer system, with kernel, OS and more. Applications running on these, is quite resource intensive. Containerizing on the other hand, does not make a ”full” VM. Containerization virtualizes the hosts OS to leave more headroom for the hardware. *Containers* in this context are software bundles that can run on top of this virtualized OS. The applications run isolated from each other, but can utilize the same kernel and if specified, libraries and binaries from the host. If they share resources, for example a library, the application will get read-only rights so their operations don't interfere with each other.

From Figure 2.2, the structure of a VM is located on the left-hand side, whereas a similar structure for a container is located on the right. It can be observed that

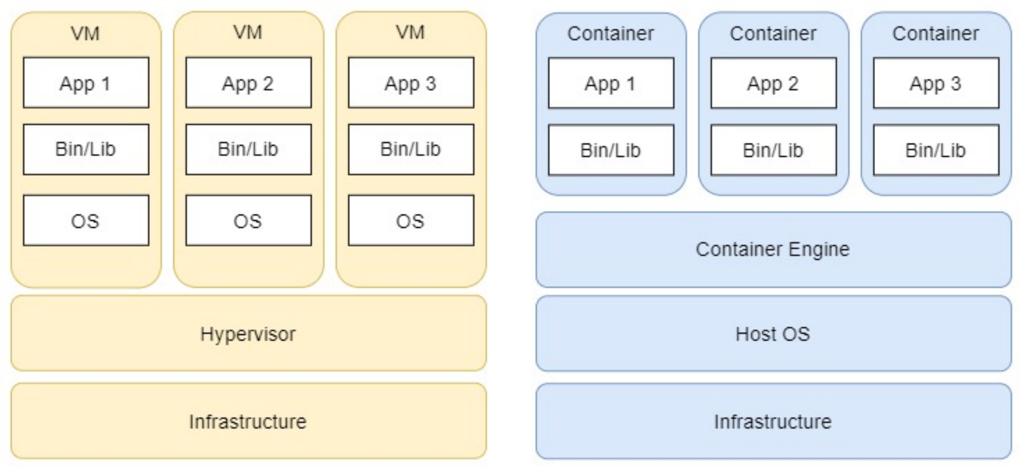


Figure 2.2: Structure of VM on the left, and structure of Container on right

the container is using the host OS to run the containers in which applications can be accessed. This differ from the VM, where all applications have their own dedicated OS. A popular container software used in recent years is Docker, which is a software built with GO to containerize applications [4]. Go is a lightweight programming language developed by Google built for fast, reliable, and efficient software at scale [5]. Large parts of the thesis is completed with containers in Docker.

Chapter 3

Approach

The projects *Diskos*[6], *Norce* and *NCS30* are platforms for data sharing. Diskos is the Norwegian national data repository for petroleum data. This is where oil companies can share seismic data and well data, and other relevant data to be publicly available. Diskos also let the companies trade data amongst each other, not disclosing the trades to the public. Norce, or ”The Norwegian Research Centre” has a solution for users to share data sets and research articles.

All these projects handle data sharing, but the traditional way of storing data is centralized, thus needing to trust a centralized server is a must. There exists other options, such as a decentralized system. The advantages of having a decentralized system in charge of authorization includes making it easier to trade secret data or give permissions to specific data sets. This thesis aims to improve upon these existing models by using new technologies and propose an alternate solution.

This is done by exploring the challenges and possible solutions for the authorization and identification part of an industrial application system, and will in conjunction with the other thesis projects make up a solution for a system of this magnitude.

3.1 Existing Approaches

The Hyperledger Foundation is an open source collaboration project that includes numerous sub-projects. This non-profit organization helps with hosting many various blockchain resources and infrastructures. This ensures that the community can utilize these resources to create commercial software or help students learn about the new technology. Two relevant projects from the HyperLedger Foundation that will be examined here are *Indy* and *Aries*.

3.1.1 Indy

Indy is a codebase provided by the Sovrin Foundation. Sovrin made a solution for a full Self-Sovereign Identity (SSI) system where all the users are in full control of their data instead of it being at the mercy of a organization. The solution is a public permissioned blockchain, with an architecture of stewards, service providers and clients. Where the blockchain contains transactions between globally unique Decentralized Identifiers (DID). Sovrin have published many papers on this topic, for example on how SSI is going to be very import for the future of the Internet [7]. They have also published a detailed explanation on how Sovrin works [8].

As stated, the Sovrin project has since been altered somewhat and incorporated with The Hyperledger Foundation. The Indy project consists of multiple repositories that are worked on by the open source community and Hyperledger itself. This includes *indy-node*, *indy-plenum*, *indy-sdk* and *indy-agent* [9]. The project also utilizes the Ursa project (as many other Hyperledger projects) for implementation of cryptographic code and interfaces. The Ursa implementations range from different signature schemes to zero knowledge proofs, all of this is modular and implemented in many different softwares. This reduces duplicate work, and sheds more eyes on this open source solution, which will give it a leg up in discovering vulnerabilities.

3.1.2 Aries

The Hyperledger Aries project on the other hand, is the tool components that make up most of the clients that interact with a ledger made for verifiable credentials, either if it issuing or verifying. This includes wallet features for clients, first of all for storing information locally in a cryptographically safe manner. It also implements a messaging system for communicating with other clients, which Aries refer to as *agents*. The messaging is done off-chain, more specifically peer-to-peer with various forms of protocols. Aries include a specific protocol for resolving DID and another for verifiable claims exchange.

Aries is based on the agents from the original Sovrin proposed solution, but has over time been drawn away from being an Indy/Sovrin helping component and is becoming its own thing. It can also have a pluggable resolver so it does not rely on one blockchain. It could resolve a transaction coming from *Bitcoin* and use those values in a higher level of abstraction, like an application. This is what makes it really versatile. There are a lot of open source Aries agents that can be used as "*plug and play*" software frameworks for specific languages, Python, .NET, etc. However, it is also possible to build a custom agent that implements the protocols of Aries.

3.1.3 Complete Hyperledger solution

Indy and Aries in tandem are one of the most used ways to setup a decentralized identity system. The projects include a variety of tools to build unique solutions to the problems at hand. Aries gives a toolkit for communicating peer-to-peer without disclosing identity or sensitive data in the process. Indy provides tools to make a standalone decentralized identity ledger, and a consensus protocol on top.

Indy and Aries fit in the *Trust over IP* technology architecture, first formalized in the paper from the Trust Over IP [11]. The paper establishes standards on how Trust over IP can be achieved and the different layers associated with this. Figure 3.1 is derived from this paper. In the figure *Indy* would be placed at layer one since it maintains the DIDs and the actual blockchain ledger. *Aries* on the other hand would be be both layer two and layer three. It enables peer-to-peer connection and the ability to issue/verify credentials.

Figure 3.2 shows the technological components of such a system, and how the different projects interacts with each other. For example, the cryptographic library *Ursa* is used for all of them. As said earlier, this makes it less likely that a refactoring is needed for all of the components if a security flaw found in the *Ursa* project is patched there.

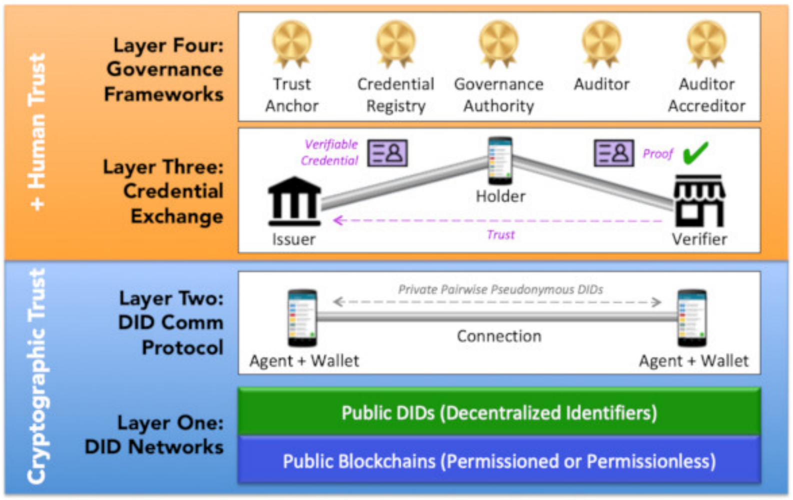


Figure 3.1: SSI architecture tech stack url [10].

3.1.4 Microsoft

In the last few years, Microsoft has also put time and resources into the decentralized identifier space. This has resulted in the launching a new product family, the Microsoft Entra [12]. They share a number of the same ideas as Sovrin/Hyperledger, but with their own twist.

First of all, they do not use the same blockchain technology, but utilize something called an Identity Overlay Network (ION). ION is based on the Sidetree protocol [13], and uses Bitcoin as the underlying ledger. This is accomplished by encapsulating the hash of an anchor file in a Bitcoin transaction. The ION network nodes use this anchor hash to get and replicate DIDs added to other ION nodes. This allows nodes to forgo running a consensus protocol; instead, the solution relies on the Bitcoin network's continued existence. The nodes share and store files using a peer-to-peer file system, InterPlanetary File System (IPFS).

Users for this service will face the "Microsoft Authenticator App", it will act as the agent for the decentralized system. It stores the verifiable credentials, makes DIDs for the user and stores the seed locally and encrypted in the wallet. This is also where the user get to decide whether or not to share the credential to a third party. They have also developed their own resolver for a *did:ion* method to deliver DID documents from the identifiers given by the ION network. Their cloud service Azure is used for issuing and verifying the credentials.

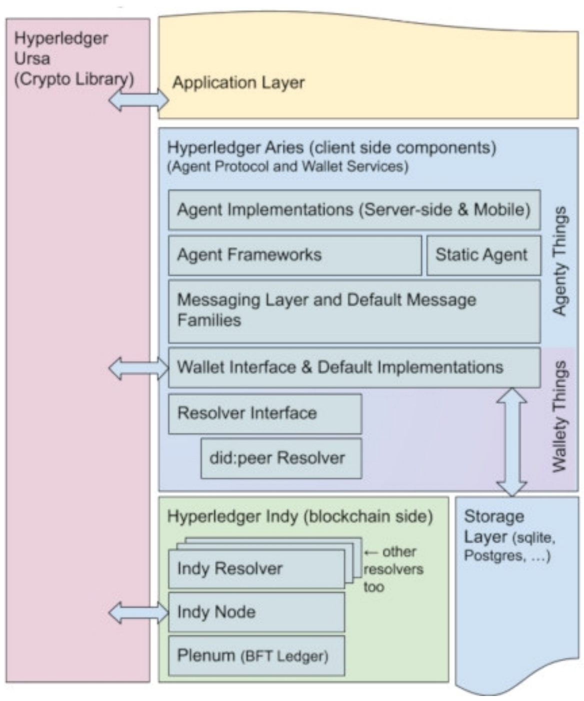

Hyperledger Ursa (Crypto Library)
Application Layer
Hyperledger Aries (client side components)
(Agent Protocol and Wallet Services)
Agent Implementations (Server-side & Mobile)
Agent Frameworks
Static Agent
Messaging Layer and Default Message Families
Agenty Things
Wallet Interface & Default Implementations
Resolver Interface
did:peer Resolver
Wallety Things
Hyperledger Indy (blockchain side)
Indy Resolver
← other resolvers too
Indy Node
Plenum (BFT Ledger)
Storage Layer (sqlite, Postgres, ...)

Figure 3.2: Hyperledger technology components url [10]

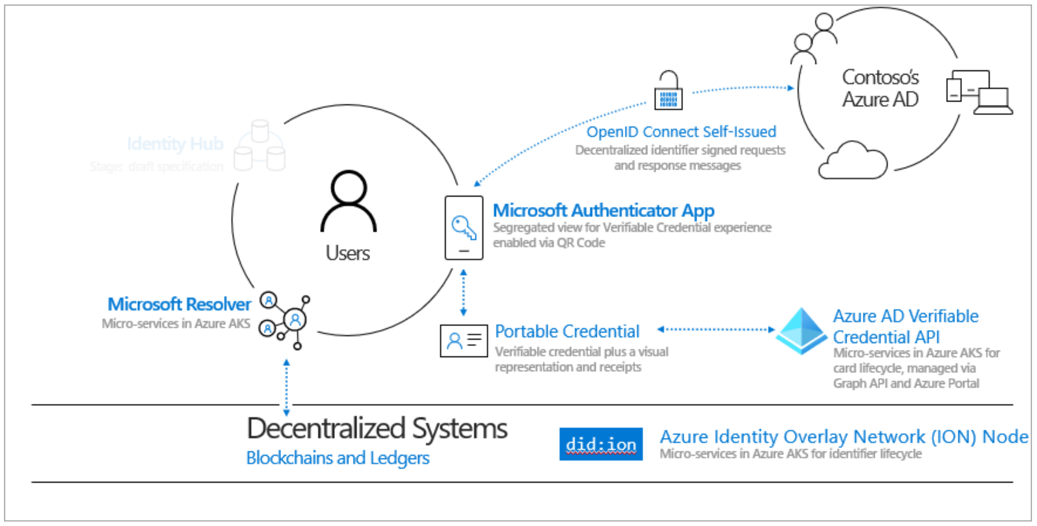

Contoso's Azure AD
OpenID Connect Self-Issued
Decentralized identifier signed requests and response messages
Identity Hub
Users
Microsoft Authenticator App
Segregated view for Verifiable Credential experience enabled via QR Code
Microsoft Resolver
Micro-services in Azure AKS
Portable Credential
Verifiable credential plus a visual representation and receipts
Azure AD Verifiable Credential API
Micro-services in Azure AKS for card lifecycle, managed via Graph API and Azure Portal
Decentralized Systems
Blockchains and Ledgers
did:ion
Azure Identity Overlay Network (ION) Node
Micro-services in Azure AKS for identifier lifecycle

Figure 3.3: Microsoft Entra architecture, URL: [14]

Chapter 4

Application Architecture Approach

Now that we have looked at the technology and the some existing solutions, we could try to find a solution the problem at hand. In this chapter, a proposed solution is presented based on the existing technology and projects examined in Chapter 2 and 3.

4.1 Analysis

This project is going to be a part of a bigger solution for what was introduced in Chapter **??**. The goal is to develop an industrial size application that can enable data sharing, data trading and royalties. The group decided to use some of the same technologies so it would be easier to piece all of them together in the end, or for further development. This project will therefore act as a puzzle that fits into a bigger picture.

4.2 Layer One

Layer one of the decentralized system will first and foremost be the underlying data structures. There were also other things to consider, like the consensus protocol (or equivalent), and the nodes that would have to be employed to ensure some form of unified state.

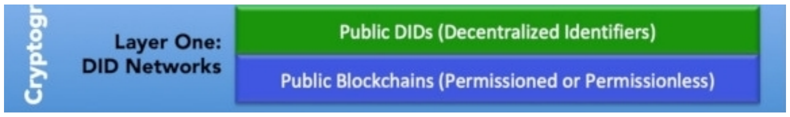

Cryptogr
Layer One:
DID Networks
Public DIDs (Decentralized Identifiers)
Public Blockchains (Permissioned or Permissionless)

Figure 4.1: Layer one in technology stack

Figure 4.1 shows a cut out from the SSI architecture Figure 3.1 in Section 3.1.3. This system will be designed for a federation that wants to control most of the aspects of the solution. A public blockchain would not be optimal, therefore a private blockchain will be used.

Indy was originally a public blockchain based on Sovrin where everyone could join, but with the new software tools it is possible to make it permissioned and private. This makes the ledger hidden from other parties, and only those that federation allows, will be able to write transactions.

Another student also saw the possibilities for using *Indy* in his project so it was decided to use that in this approach as well. There is a lot of useful built in methods and tool-kits for an *Indy* ledger approach and the community continually improve upon and develop the software.

The *Indy* DID system has a method which includes a *namespace*. The method did:indy:{namespace} determines what ledger this DID is associated with.

So for our problem a namespace for the federation of organizations could be ”datasharingfederation”. So every DID created to interact with this application could be in the ”datasharingfederation” namespace and a single identifier could look like this:

did:indy:datasharingfederation:3NWT7arTrLdNaHeHJE393

(the identifiers are base58 encoded, but this could be changed, if requested upon).

Each organization would have to run their own Indy node. That node would have their copy of the ledger stored locally, and it would be updated by that client. The organizations’ Indy node would also need to be open to communication. The node would also need to talk to the other organizations’ Indy nodes and run the consensus protocol, which is based on a redundant byzantine fault tolerance protocol [15].

4.3 Layer Two

As mentioned earlier, the second layer in the decentralized system will be the communication protocol. This is where the two agents are able to communicate without revealing sensitive information. This is usually done with multiple hops, and multiple intermediaries to relay a message. They utilize an underlying Public Key Infrastructure (PKI) to ensure that the messages are encrypted, that they have not been tampered with, and been sent from the correct person.

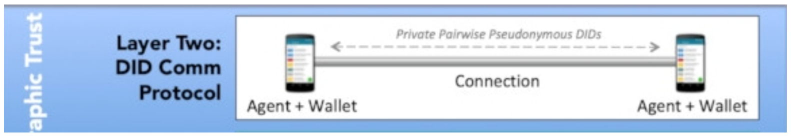


Figure 4.2: Layer two in technology stack

4.3.1 Choice

Since *Indy* became the chosen technology, it also made sense to use *Aries* for the second layer of communication. An *Aries* agent can communicate with others and can write transactions to the ledger. As mentioned in Section 3.1.2, there are multiple frameworks already present, and there are pros and cons with most of them. So it is best to look at what the agent would need to do, and how it could communicate with the rest of the system. The main objective here is authentication of users and usability.

4.3.2 Static Agent

From the get-go, a static *Aries* agent was attempted to be implemented with the necessary features, like writing to a ledger. This did however take a lot of time, this was partly due to the lack of good tutorials for development, will touch on this later in the discussion.

4.4 Layer Three

A new perspective was needed, and further investigation of what the *Aries* agent actually was going to solve was needed. Especially since the *Aries* agent will also need to serve the purpose of credential exchange in layer three as seen in figure 4.3.

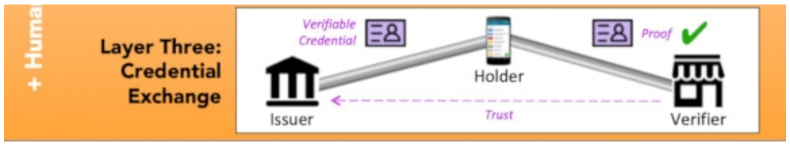


Figure 4.3: Layer three in the technology stack

When looking at the structure in Figure 4.3, and thinking about the governance for a system like this, one can see that most issuing will happen from organization to employees, and not cross organizations. A great way to make it easier for the users in this case is for them to only deal with their own organization and have the system overall decide whether or not they can do something at that organizations place. This could be a proxy, so users would interact like normal, with username-passwords and have the organization keep track of their own users decentralized identities for them. I.e. a service that creates DIDs, wallets, credentials and when the user interacts with another organization it will just need present their own info and the foreign service will reach out to their originating provider.

4.5 Architecture

After researching many *Aries* frameworks and the features that they offered, a possible architecture that could solve many of the issues that we set out to accomplish was suggested. This being the advantages of having verifiable credentials rooted in blockchain technology and the modularity with these open source libraries.

4.5.1 Blog inspiration

What especially inspired the solution was a blog [16] by Laurence de Jong. It details a system with a controller that interacts with the *Aries* agent. This gave the idea of a controller that could interact with the rest of the application and an agent. The author also utilized the Aries Cloud Agent - Python (ACA-Py), and it seems to be tailored for this use case.

4.5.2 Aries Cloud Agent -Python

This *Aries* agent has a lot of built-in features and abstracts some of the protocols so it is easier to develop with. As the naming suggests the agent is supposed to run on a server instead of mobile devices. This in somewhat contrast to what we've heard about agents earlier, but this does make sense. There has to be two kinds of agents, one for local storage of credentials and wallets for all users and then services that they can interact with, this is where this kind agent framework comes along. It is more centered around having it constantly running and a controller interact with the admin API. One of the built in features to be utilized is the multi-tenant option. This enables the agent instance to manage multiple sub wallets through a base wallet.

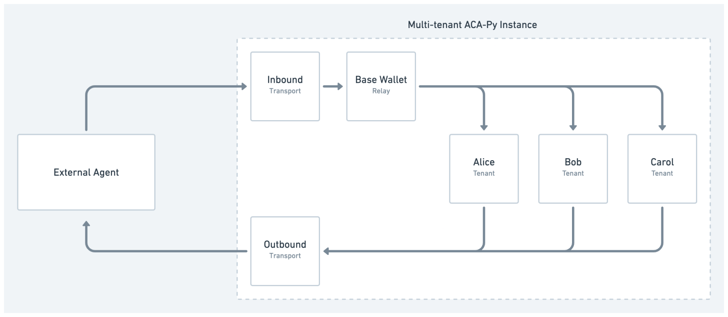


Figure 4.4: Multi-tenant illustration[17]

4.5.3 Proposed architecture

In figure 4.5 a proposed solution architecture is shown. The solution consists of a controller that interacts with the ACA-Py instance, has its own database and the overall application as well. The ACA-Py instance is set to multi-tenancy mode and has a persisting built-in wallet. The users login to the application with an assigned method, this could be a browser on the application with username-password, any form of 2FA or a mobile application of some sort. The controller can act as a proxy between the application and the decentralized system of authorization. Users only interact with the application and the underlying authorization system with *Aries* and *Indy* fixes and resolves the rest. This by having a reverse proxy expose the controller and agent to the rest of the internet.

With the addition of a controller, there could be applied logic logins as well.

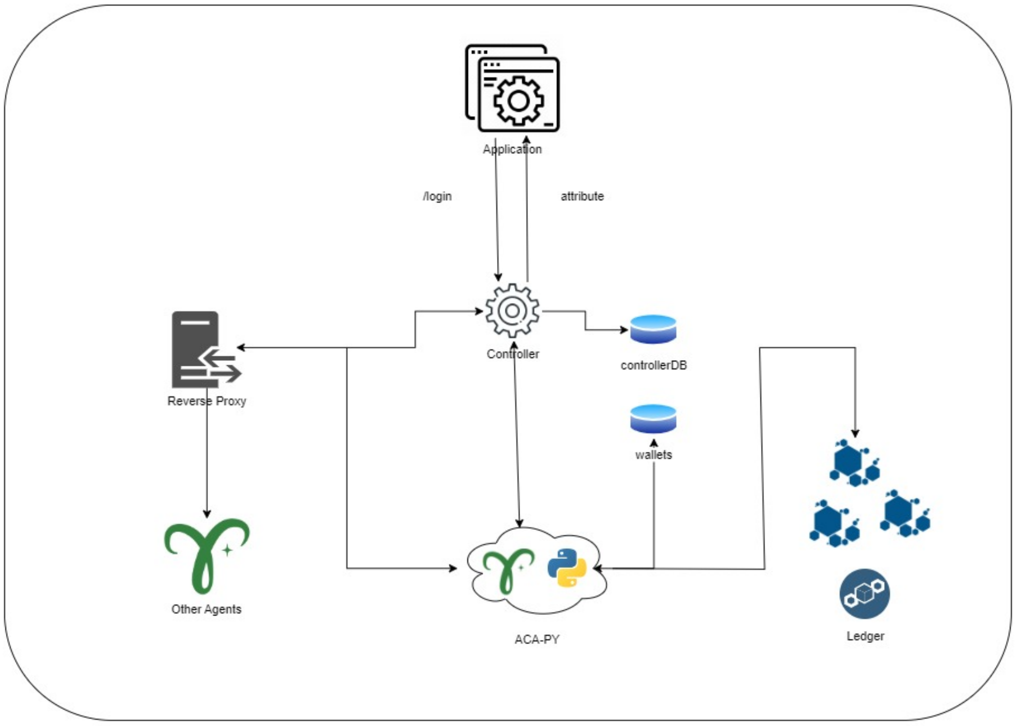


Figure 4.5: Architecture with symbols

For example, if a user of Organization 1 wants to use a resource of Organization 2, it could try to connect, but you could have a policy that accepts or reject this based on organization credential, or a fixed number at a time.

Chapter 5

Experimental Implemetation

5.1 Technologies

The experimental solution will be using the architecture described earlier and implement the technological choices made. Since there is both a decentralized part, and some conventional pieces, a presentation of technologies used is found in Figure 5.1.

Figure 5.1: Technologies implemented in this project

As pictured and described earlier, Hyperledger Aries and Hyperledger Indy is used as the decentralized technology tool kits. The *Aries* agent is the Aries Cloud Agent - Python (ACA-Py) that contains many smart implemented features. One of these are a persistent wallet made with PostgreSQL. However an instance of a PostgreSQL database for the controller is also needed. The controller on the other hand, is a Go application. These are instantiated with the help of Docker.

5.2 Docker Compose

To make it easier to develop upon and to launch all of the software infrastructure for someone else a *docker compose* is set up. A generated *.yaml* file defines the different services that the application consists of. When running the *docker-compose up* command at the root of the project, the program does a *docker run* for all of the services in the *.yaml* file.

5.2.1 ACA-Py

Ended up running a static image of the Aries Cloud Agent, but it's not too difficult to alter the code to a custom one. This can be done be replacing the *image* statement in the docker compose with a *build* statement of the local project. Multiple ways to do this, either a custom new docker file or git pull the repository locally and alter the code there. Other than that there is a lot of command line arguments that need to be set up for it to work. So everything from ledger type to secrets.

5.2.2 Go Application

The controller is made from scratch and includes startup functions, and some routes. The controller can sign up new users, and it will create a wallet on the Aries agent through the admin API

5.2.3 Controller DB

This is a local instantiated database from the latest postgres image docker can pull. A postgres admin instance is also run so we can alter the database ourselves in a web browser and can explore the data present.

Chapter 6

Discussion

6.1 Background knowledge

6.1.1 Documentation

Some of the issues I had when doing this project was the different forms of documentation. There are code examples, and some demos of how stuff should work, but these may not always be up to date, and other aren't even implemented yet. Say I've had all this knowledge about how the systems work and interact with each other I could've made way more progress in one month, with production code than the whole semester put together.

6.1.2 Time spent learning

I spent some time learning or trying to setup old protocols that is not even begin used today. So it's somewhat hard to get an overview of what works, and what to use time at. For example, when trying to test Aries agents towards some ledger there is a repository that can launch the minimum of four *Indy* nodes that could work as testing environment. However i didn't find out this before trying to setup some failed attempt.

6.2 Architecture design

This is where most of my time was spent. It was difficult to find out exactly what would be best, in terms of implementation, the users and the organizations. What software to use and what use cases to prioritize. So I ended up with something that may work, but it is not tested through and through. However the result seems like a good halfway point, it is not too decentralized so it becomes alien for the users, however you get some of the advantages of decentralized technology. It was also not goal of having a fully decentralized system, but divide some of the authority between organization or to make it more transparent.

6.3 Implementation

This is a field I should have spent more time on and the actual application has suffered some from this. The problem is that this is pretty new technology that is constantly being developed, and issues as mentioned in 6.1.1 does not help in speeding up the process. It is also a whole new way of looking at identity than we have learned in school, and it is therefore harder to get into, and I did not have all the background knowledge ingrained into my head. After a while i also figured out that golang might not be the best language for a controller. Something like JavaScript or C# could work better, this is because of the responses given by the ACA-Py agent are big and they need to be heavily typed in golang to avoid compile errors.

6.4 Project work

This was part of a bigger project to make a platform for data sharing, and that meant that certain considerations had to be made. There were 5 different projects with different fields, and all of these should be able to fit together. The authorization part is not connected to something, but it won't be hard to accomplish since it was designed with this is mind.

Chapter 7

Conclusions

This thesis did manage to design a system that could work for authorizing users locally . It has a underlying decentralized structure that lets organizations issue verifiable claims to both their own employee/users. However the implementation is not complete and contextualized resolving is missing.

7.1 Future work

7.1.1 Putting together

The system was designed with the others in mind, however the pieces has not been put together. So the application that needs to authorize users needs to connect to the controller to utilize its features.

7.1.2 Finalizing implementation

The most important part would be to connect the Aries agent with a Indy ledger that can track changes in verifiable claims, and be a source of truth. A controller in golang may not be the best way of interacting with bigger response objects, so this could probably be changed to something else. The controller also needs more methods to interact with the system. Another important future development could be to implement local policies for verifiable claims. The reverse proxy for exposing the agent to the rest of the internet is also not implemented and would be vital in a finished solutuion. Before any of this could be put into a production environment there is a lot of static secrets and IPs that need to be abstracted into variables that could be set by a `.env` file.

7.1.3 Testing

I did not get to test the setup since it was not finished enough to emulate a system. There are however many things that could be tested. This is everything from revocation of claims, cross organization with and without policies that allow this. A general throughput test could also be executed to find out where bottlenecks(if any) appear.

List of Figures

Bibliography

[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

[2] Azaghal. Merkle tree illustration, . URL https://upload.wikimedia.org/wikipedia/commons/thumb/9/95/Hash_Tree.svg/1920px-Hash_Tree.svg.png.

[3] Azaghal. Illustration from: Blockchain for dynamic spectrum management - scientific figure on researchgate, . URL https://www.researchgate.net/figure/The-structure-of-a-Blockchain-A-block-is-composed-of-a-header-and-a-body-where-a-h
fig1_33730638.

[4] Home - docker. URL https://www.docker.com/.

[5] The go programming language. URL https://go.dev/.

[6] Norwegian Petroleum Directorate. Diskos: About us. URL https://www.npd.no/en/diskos/About/.

[7] Andrew Tobin and Drummond Reed. The inevitable rise of self-sovereign identity. *The Sovrin Foundation*, 29(2016), 2016.

[8] Drummond Reed, Jason Law, and Daniel Hardman. The technical foundations of sovrin. *The Technical Foundations of Sovrin*, 2016.

[9] Tracy Kuhrt and Ry Jones. Hyperledger indy wiki. URL https://wiki.hyperledger.org/display/indy.

[10] Evernym. Evernym: About page. URL https://www.evernym.com/blog/hyperledger-aries.

[11] Matthew Davie, Dan Gisolfi, Daniel Hardman, John Jordan, Darrell O'Donnell, and Drummond Reed. The trust over ip stack. *IEEE Communications Standards Magazine*, 3(4):46–51, 2019.

[12] Microsoft entra main page. URL `https://www.microsoft.com/en-us/security/business/microsoft-entra`.

[13] Sidetree protocol specification. URL `https://identity.foundation/sidetree/spec/`.

[14] Microsoft Documentation/Azure. Introduction to azure active directory verifiable credentials (preview). *Microsoft Documentation*, 2022. URL `IntroductiontoAzureActiveDirectoryVerifiableCredentials(preview)`. Updated: 14.06.2022.

[15] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. Rbft: Redundant byzantine fault tolerance. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*, pages 297–306. IEEE, 2013.

[16] Laurence de Jong. Building an aca-py controller: Architecture. URL `https://ldej.nl/post/building-an-acapy-controller-architecture/`.

[17] Aca-py multitenant github. URL `https://github.com/hyperledger/aries-cloudagent-python/blob/main/Multitenancy.md`.

Appendix A

Instructions to Compile and Run System

Github link: `https://github.com/TheseusDeus/distributed-identity-master`

A.1 Requirements

To be able to run the code, docker has to be installed on the device it's going to be launched at. The docker compose file format has version 3.0, so this means that the docker engine has to be of release >1.13.0+. If running this on windows it could be beneficial to have Windows Subsystem for Linux (WSL) installed and run it there, and docker desktop is also an advantage.

A.2 Running the code

1. Pull the GitHub repository

```
git clone https://github.com/TheseusDeus/distributed-identity-master
```

2. Move to the correct directory

```
cd distributed-identity-master/system
```

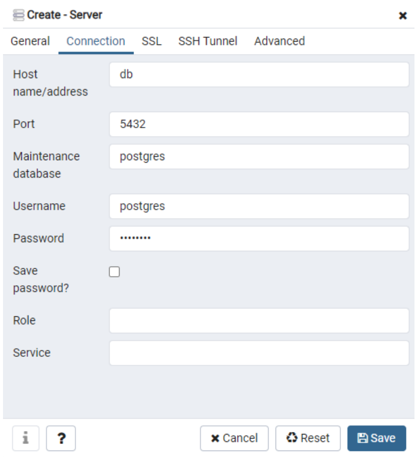
3. Run infrastructure

```
docker-compose up
```

If you have docker desktop installed, all of the containers should pop up there, and you can watch their logs from their, or all of them in the terminal that launched the program.

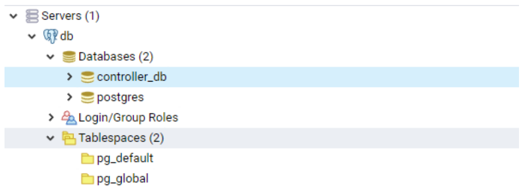
A.3 Looking at Database

To look at the database, you must go to `http://localhost:5050` If promted for a password, use ”*password*”. From there the admin tool will show all servers associated with the login on the leftside. If the database is not connected, you need to add a new server. Press the first button under quick links. On the first page give it any name, then go to the *Connection* tab and fill in hostname with *db* and password with *password*, and save. It should now appear on the left side.



Host name/address	db
Port	5432
Maintenance database	postgres
Username	postgres
Password	••••••••
Save password?	☐
Role	
Service	

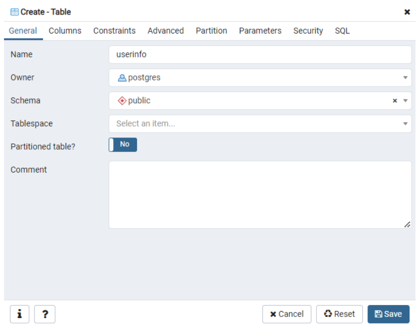
(a) fill in this for connection tab



```
Servers (1)
  db
    Databases (2)
      controller_db
      postgres
    Login/Group Roles
    Tablespaces (2)
      pg_default
      pg_global
```

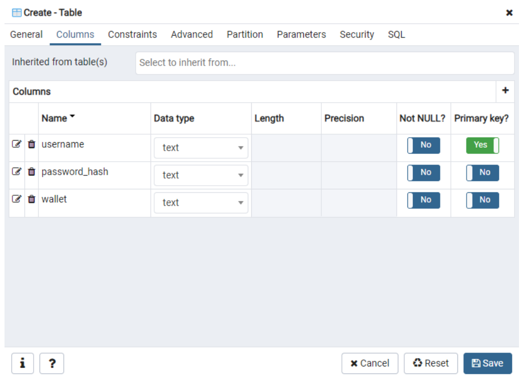
(b) new server added

If the database does not have a tables. (can be found under, controller_db -> Schemas -> Tables), you need to create a new one as shown below.



Name	userinfo
Owner	postgres
Schema	public
Tablespace	Select an item...
Partitioned table?	No
Comment	

(a) general tab



Inherited from table(s): Select to inherit from...

Columns

Name	Data type	Length	Precision	Not NULL?	Primary key?
username	text			No	Yes
password_hash	text			No	No
wallet	text			No	No

(b) columns tab

Now it is possible to right click the *userinfo* table and press view all data to show it on the right side.

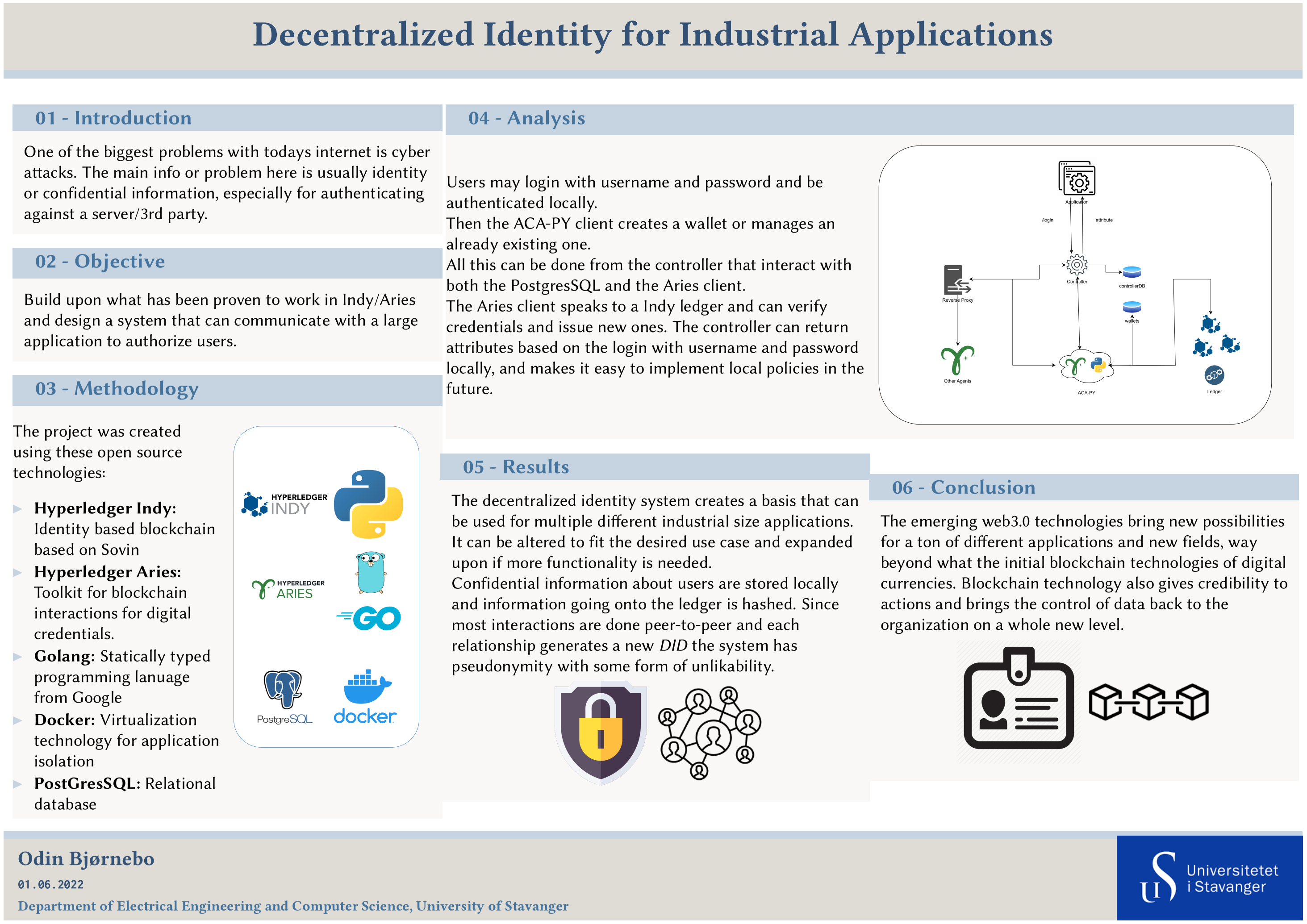
A.4 Video

Here is a short video of the system up and running, where a user signs up, the data is saved to the database, and a wallet is created for the user. `https://drive.google.com/file/d/1Pw-fbysFipIj_3n_SwgLjh1f5OvgVHgp/view?usp=sharing`

Appendix B

Master presentation

Poster from presentation can found below.


/login
attribute
Application
Controller
controllerDB
wallets
Reverse Proxy
Other Agents
ACA-PY
Ledger
HYPERLEDGER INDY
HYPERLEDGER ARIES
GO
PostgreSQL
docker
Universitetet i Stavanger

Decentralized Identity for Industrial Applications

01 - Introduction

One of the biggest problems with todays internet is cyber attacks. The main info or problem here is usually identity or confidential information, especially for authenticating against a server/3rd party.

02 - Objective

Build upon what has been proven to work in Indy/Aries and design a system that can communicate with a large application to authorize users.

03 - Methodology

The project was created using these open source technologies:

- **Hyperledger Indy:** Identity based blockchain based on Sovin
- **Hyperledger Aries:** Toolkit for blockchain interactions for digital credentials.
- **Golang:** Statically typed programming lanuage from Google
- **Docker:** Virtualization technology for application isolation
- **PostGresSQL:** Relational database

04 - Analysis

Users may login with username and password and be authenticated locally.
Then the ACA-PY client creates a wallet or manages an already existing one.
All this can be done from the controller that interact with both the PostgresSQL and the Aries client.
The Aries client speaks to a Indy ledger and can verify credentials and issue new ones. The controller can return attributes based on the login with username and password locally, and makes it easy to implement local policies in the future.

05 - Results

The decentralized identity system creates a basis that can be used for multiple different industrial size applications. It can be altered to fit the desired use case and expanded upon if more functionality is needed.
Confidential information about users are stored locally and information going onto the ledger is hashed. Since most interactions are done peer-to-peer and each relationship generates a new *DID* the system has pseudonymity with some form of unlikability.

06 - Conclusion

The emerging web3.0 technologies bring new possibilities for a ton of different applications and new fields, way beyond what the initial blockchain technologies of digital currencies. Blockchain technology also gives credibility to actions and brings the control of data back to the organization on a whole new level.

Odin Bjørnebo
01.06.2022
Department of Electrical Engineering and Computer Science, University of Stavanger

Figure B.1: Master presentation from 1. June

University
of Stavanger

4036 Stavanger
Tel: +47 51 83 10 00
E-mail: post@uis.no
www.uis.no

Cover Photo: Hein Meling