Universal Ledger Agent

Creating Hyperledger Aries plugins for the Universal Ledger Agent

Timo Glastra
Karim Stekelenburg

University of Applied Sciences Utrecht
Institute for Information Technology
Minor Blockchain

Supervisors
David Lamers
Claudia Terpstra
Marnix van den Bent
Bernhard van der Biessen
Jaap Goedegebuur

v1.0
Utrecht, 18th January 2020

Preface

In the context of the 2019/2020 blockchain minor at the HU University of Applied Sciences Utrecht we, Karim Stekelenburg and Timo Glastra, worked on a new and open sourced piece of self-sovereign identity technology created by the Rabobank Blockchain Acceleration Lab. This document describes the goals of the project, the research done, the results that came out of this research, and the problems faced getting these results.

Contents

Chapter 1

Introduction

In the last four years a digital movement originated that recognizes an individual should own and control their identity without the intervening administrative authorities. This movement is described with the term self-sovereign identity (SSI) (Allen, 2016) and allows people to interact in the digital world with the same freedom and capacity for trust as they do in the offline world. In current implementations trust is created with verifiable credentials (Sporny, Dave & Chadwick, 2019). A verifiable credential is a piece of information that is cryptographically trustworthy. Decentralized identifiers (DIDs) (Reed et al., 2019) are used to provide verifiable, decentralized identity that can be linked to verifiable credentials.

1.1 Rabobank and the Universal Ledger Agent

Rabobank is an early adopter of the SSI movement and recognizes the problems it solves. Right from the beginning in 2016, when the term SSI got coined, Rabobank is doing research on self-sovereign identity using blockchain technology. Together with a wide range of partners they are exploring possibilities. One problem they faced is that partners were using different standards and blockchains. This inspired Rabobank to create the Universal Ledger Agent (Lamers, 2019). The Universal Ledger Agent (ULA) is a modular framework designed to solve interoperability issues. The ULA makes use of plugins. Each ledger can be supported by its own set of plugins.

Recently Rabobank open sourced the ULA with a set of plugins. They want to encourage others to build plugins for the ULA. With open sourcing, Rabobank also hopes to get more involvement from the community.

1.2 Sovrin, Hyperledger Indy and Hyperledger Aries

Sovrin (Sovrin Foundation, 2018) is the first global public utility exclusively for self-sovereign identity and verifiable credentials. It is based entirely on open standards and is open source. The Sovrin code base is now available as Hyperledger Indy (Hyperledger, n.d.-c) under control of the Linux Foundation. Sovrin is the most well known network and is still based completely on Hyperledger Indy. This makes Sovrin and Indy a good addition to the ULA.

Recently a new interoperable toolkit designed for initiatives and solutions focused on creating, transmitting and storing verifiable digital credentials got introduced named Hyperledger Aries (Hyperledger, n.d.-b). Currently Hyperledger Aries has a limited number of implementations and only supports Hyperledger Indy based networks. However, because the core of all SSI implementations rely on the same verifiable credential and DID standards Hyperledger Aries can bridge the interoperability gap between different SSI solutions.

Chapter 2

Project Definition

The goal of the project is to provide the ULA with support for the Sovrin network in a way that is immediately adoptable and useful to end-users of the ULA. Therefore it is wise to define a few requirements for the project.

2.1 Mobile devices

Because of the fact that in a real world scenario the holder is likely to have to provide credentials while on the go, it is important that all plugins work on mobile devices. This could potentially introduce some challenges regarding the availability of cryptographic libraries for mobile devices.

2.2 The browser

Because the current set of plugins the ULA has, work from within the browser, these should too.

2.3 Offline

Besides performing self-sovereign identity related actions, users of the plugins should also be able to react to incoming interactions/requests. However, we cannot assume the device running the plugins is always online. Therefore the plugins need to be able to pick up interactions where it left off after going offline.

Chapter 3

Research

There are multiple ways to achieve Sovrin integration for the ULA. Each solution has its own set of pros and cons and are evaluated with the following questions in mind:

- What is the best SDK or framework to use for integrating Sovrin into the Universal Ledger Agent?
- What is the best way to provide serverless devices with an always reachable endpoint?

3.1 Indy SDK with Indy Agent

The most obvious option to integrate Sovrin into the ULA would be to use the Indy SDK (Hyperledger, n.d.-e). The Indy SDK is the official SDK for Hyperledger Indy and is what the Sovrin network is built on. If the Indy SDK is used, Sovrin and all other Hyperledger Indy networks would be supported. Indy SDK is written in the Rust programming language and has available wrappers for Java, Python, iOS, NodeJS, .Net and Rust. Being available in this many programming languages makes the Indy SDK an accessible SDK to use.

The ULA is completely written in TypeScript, a superset of JavaScript. To work with the ULA the Indy SDK must be able to work with JavaScript, and it sort of does. Indy SDK has a wrapper available for NodeJS. NodeJS (OpenJS Foundation, n.d.) is a JavaScript runtime that runs directly on a computer. In contrary to a lot of the JavaScript code written, with NodeJS the code does not run in a browser environment. All Indy SDK wrappers use a native c-callable library under the hood written in Rust. This reduces the numbers of duplicate implementations needed, but this makes the Indy SDK unusable in a browser environment at the moment. If the native c-callable library could be compiled to WebAssembly (Wasm) (WebAssembly, n.d.) the Indy SDK would be usable from within a browser environment. Although compilation to Wasm is on the Indy SDK wishlist (Esplin, 2019), it is currently blocked due to some dependencies. If the Indy SDK is used to integrate Sovrin in the ULA, it will only work in NodeJS environments, not in browser environments.

The Indy SDK manages most of the functionalities needed for issuing and verifying credentials but does not cover how sides set up a connection and exchange messages. For this Hyperledger Indy has the Indy Agent repository (Hyperledger, n.d.-d) that contains reference implementations of what an indy agent would look like. An agent in the context of Hyperledger Indy can be seen as agent of self-sovereign identity and acts as a fiduciary on behalf of a single identity owner. It holds cryptographic keys that uniquely embody its delegated authorization and interacts using interoperable DIDComm protocols (Hardman, 2017). The Indy Agent repository however only contains reference implementations and is deprecated and superseded by Hyperledger Aries (Hyperledger, n.d.-b) since 2019. If the Indy Agent standards and reference implementation is used to add the missing connection and message exchange functionalities from Indy SDK, the agent must be build almost from scratch and would use deprecated standards. Indy SDK could be used with the superseding project of Indy Agent, Hyperledger Aries. This option will be discussed in section 3.3.

One problem of the Indy Agent is that it needs to be always reachable to receive messages from other agents. For businesses that want to issue and verify credentials this is not that big of a problem, but for participants using a mobile device this is not possible. This does not mean that participants using mobile device should not be able to participate. To solve this the Indy Agent could be implemented as a Cloud Agent. This would mean the Agent would be hosted as a server and would be always reachable. The Cloud Agent can be controlled with standard REST API calls from within the ULA. If the Indy Agent is implemented as a Cloud Agent, it will work in NodeJS environments, as well as browser environments. Each participant would however need to run a server.

Using Indy SDK with Indy Agent to integrate Sovrin into the ULA would mean:

– Each participant needs to run a server (Cloud Agent) to be always reachable

– Agent must be build almost from scratch

– Agent would use deprecated standards

3.2 LibVCX

LibVCX (Hyperledger, n.d.-f) is a library built on top of Libindy (Indy SDK) and provides a high level credential exchange protocol. It simplifies creation of agent applications and provides better agent-2-agent interoperability for Hyperledger Indy infrastructure. LibVCX offers the same functionalities as the Indy SDK, and also adds the missing connection and message exchange functionalities. It is written in the Rust programming language and has available wrappers for Java, Python, iOS and NodeJS. For the same reasons as with the Indy SDK, LibVCX is not usable in browser environments at the moment.

The API that LibVCX exposes abstracts a lot of the complexities from Indy SDK. It does not require the user to work with the low level API from Indy SDK. This makes it a more straightforward option to use in terms of how easy it is to integrate into the ULA. Although this is a great benefit, LibVCX also adds a great disadvantage. Just like with the Indy SDK, LibVCX requires the participant to be always reachable. LibVCX does not implement this itself, but requires an external agent to achieve this. LibVCX has a standard protocol that defines how LibVCX communicates with the external agent. This is an advantage over the Indy SDK, where the protocol for communicating with the external agent is not defined. LibVCX only uses the external agent for connections and exchanging of messages. Issuing and verifying stil happens locally, which means the external agent has no control of the identity of the participant.

However there is no good, usable implementation of an agent that is compatible with the LibVCX protocol. There is a Dummy Cloud Agent (Hyperledger, n.d.-a) that, as the name suggests, is not meant for production use. It is meant as a reference implementation and lacks a lot of needed functionality. LibVCX is originally created by Evernym (Evernym, n.d.) and was used to provide an easier API for customers. It is now open source and can be used by everyone. However, LibVCX is only one side of it. The other side, the external agent, is not open source.

Using LibVCX to integrate Sovrin into the ULA would mean:

+ LibVCX exposes an easy to use API that abstracts a lot the complexities from Indy SDK

– As cryptographic actions still happen in the ULA, plugins will only work in NodeJS environments, not in browser environments

– Requires access to an external agent to fully work. Besides writing own LibVCX compatible Cloud Agent, an agent is only available through Evernym propiertary agent and Dummy Cloud Agent

3.3 Hyperledger Aries

Hyperledger Aries is the interoperability framework for SSI. It is future proof and has super strict protocols. Aries is very young and does not have a lot of implementations because of that. The implementations that do exist work with Hyperledger Indy and could be used to add Sovrin integration to the ULA.

3.3.1 Hyperledger Aries Framework - JavaScript

Aries Framework JavaScript (Hyperledger, 2019b) is a framework for Hyperledger Aries that is still in early development. At the start of this research the repository for the project had not yet been created. It is therefore not usable at the moment. However the potential of Aries Framework JavaScript for integrating Sovrin into the ULA is noteworthy.

Using Aries Framework JavaScript to integrate Sovrin into the ULA is not possible at the moment. It is still in early development.

3.3.2 Hyperleder Aries Cloud Agent - Python

Aries Cloud Agent - Python (ACA-Py) (Hyperledger, 2019a) is a full Hyperledger Aries Agent implementation ready to be used. It is a foundation for building services running in non-mobile environments.

ACA-Py is a server written in Python able to perform self-sovereign identity related actions. Actions are performed trough REST API calls from an external controller to the Cloud Agent. This makes ACA-Py great for integrating with the ULA. The ULA, taking the role of a controller, can send REST API calls to the Cloud Agent to perform actions and communicate with other Agents. One downside of ACA-Py is that, besides an external controller, an external always reachable webhook endpoint must be available. When events occur the Cloud Agent will notify the webhook receiver of the changed states. This could, for example, be when another participant wants to initiate a new connection.

As stated before, participants using mobile devices are not expected to be always reachable on a server. This is not possible. To solve this problem a webhook relay could be used. This will receive all incoming events from the Cloud Agent and hold them until the participant is able to receive the events. This does mean another server is needed besides the Cloud Agent. For participants that are not using a mobile device a server could be build directly into the ULA as a plugin.

Using Aries Cloud Agent Python to integrate Sovrin into the ULA would mean:

+ Relatively easy to implement in the ULA, only REST API calls needed
+ Follows the latest standard for message exchange and connections
+ As ULA only makes REST API calls, plugins will work in both NodeJS and browser environments
– All processing is done on the Cloud Agent, no complete control
– As Cloud Agent needs external webhook server, multiple servers are needed

3.4 Conclusion

Given all the options, the currently available tools and the limited time the most viable option would be to integrate Sovrin with Aries Cloud Agent Python. For mobile devices the Webhook Relay server can be used. For implementations where the ULA is always reachable a server could be build directly into the ULA as a plugin. Because the ULA only makes REST API calls it can run in both NodeJS and browser environments.

3.4.1 Beyond tomorrow

If Aries Framework JavaScript is fully developed and there is an Agency available that works with Aries Framework JavaScript and the Aries RFCs it would be a more viable option. An Agency is an externally hosted server that acts as an always reachable endpoint on behalf of participants. It would benefit participants using mobile devices that cannot be always reachable. It would mean all cryptographic actions are performed directly on the device of the participant and the Agency is only used to exchange messages. This gives the participant complete control over its identity without the need for a personal server. If available, the participant could join any Agency it would like. For example an Agency hosted by Rabobank. For participants that are able to run a server, Aries Framework JavaScript could be used to build an always reachable Agent directly into the ULA.

Chapter 4

Architecture

Because this project is integrating two pre-existing services, their architectures need to be taken into account when deciding on an architectural design. Another thing to keep in mind is that the different roles in the system (holder, issuer, verifier) will not require the exact same functionalities. However, some functionalities do overlap and therefore a modular structure is desirable. Luckily the ULA is a plugin-based event handler which is quite convenient when trying to achieve modularity.

4.1 Aries Cloud Agent Architecture

4.1.1 REST Endpoints

The ACA exposes an API that the controller (the party calling the API) can use to interact with the ACA. The API's endpoints are categorized in logical domains. These domains can be used to determine what plugins are needed for outgoing traffic.

Domain	Provided functionality
server	fetch server statuses and statistics
action-menu	domain specific to a demo application by British Columbia
connection	setup and maintain agent-to-agent connections
present-proof	verifying proof
issue-credential	issuing credentials
credentials	fetching and storing credentials in the wallet
schema	defining an uploading schema's to the ledger
ledger	registrering DIDs
wallet	creating and fetching DIDs from the wallet

Table 4.1: Aries Cloud Agent endpoint domains

4.1.2 Webhook Events

The ACA also accepts a webhook URL that is used to call back to the controller whenever something noteworthy has happened or an action of the controller is required. There are three possible webhook-endpoints the ACA can call, each representing one of the domains mentioned in in table 4.1. These domains are **connection**, **issue-credential**, and **present-proof**.

4.2 Webhook Relay Server

The webhook interface the ACA provides, requires the controller to host a server in order receive webhook events. This introduces difficulties regarding the requirement to support mobile systems, as well as the controller being able to go offline without missing out on any information. To overcome these issues, the decision has been made to develop a Webhook Relay (WHR) server that is hosted external to the controller. When the controller connects to the WHR's websocket interface, webhook calls made by the ACA are forwarded to the controller directly. Whenever the websocket connection closes, the WHR will accumulate all webhook calls made by the ACA until the controller reconnects. When this happens, all accumulated calls are 'fast-forwarded' to the controller.

4.3 Required Plugins

In order to maintain an understandable architecture, the decision has been made to create a separate plugin for each of the logical domains as seen in table 4.1. However, the *action-menu* domain will be omitted since this domain is specifically made for a demo application build by the government of British Columbia, and serves no use to the ULA. In order to provide a convenient interface for reacting to webhook events, a separate *abstract* plugin for each webhook domain is created. The user will need to extend these plugins in order to react to events.

Fig. 4.3 provides a full overview of the plugin architecture in relation to the ACA and WHR.

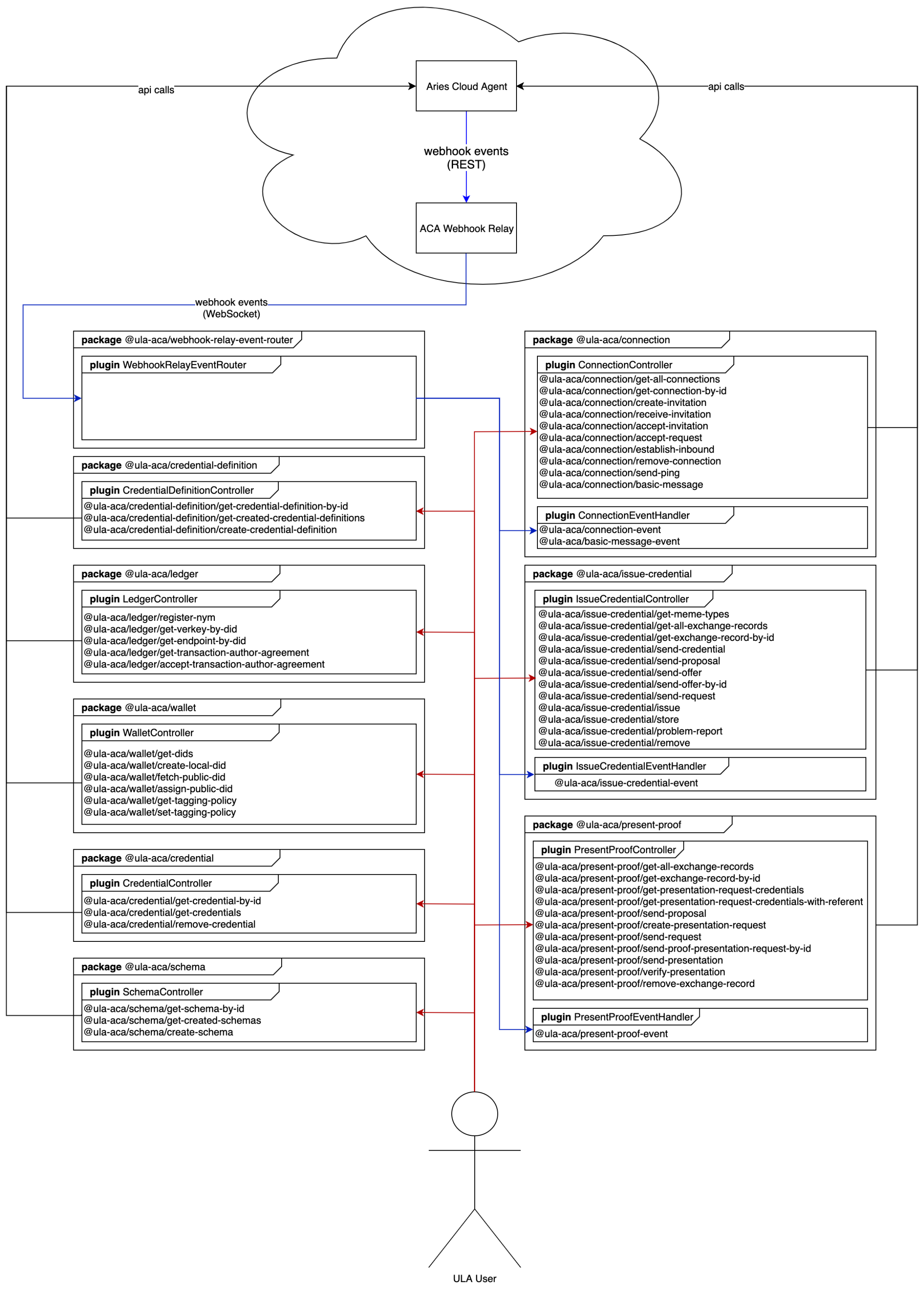
Aries Cloud Agent
api calls
api calls
webhook events
(REST)
ACA Webhook Relay
webhook events
(WebSocket)
package @ula-aca/webhook-relay-event-router
plugin WebhookRelayEventRouter
package @ula-aca/credential-definition
plugin CredentialDefinitionController
@ula-aca/credential-definition/get-credential-definition-by-id
@ula-aca/credential-definition/get-created-credential-definitions
@ula-aca/credential-definition/create-credential-definition
package @ula-aca/ledger
plugin LedgerController
@ula-aca/ledger/register-nym
@ula-aca/ledger/get-verkey-by-did
@ula-aca/ledger/get-endpoint-by-did
@ula-aca/ledger/get-transaction-author-agreement
@ula-aca/ledger/accept-transaction-author-agreement
package @ula-aca/wallet
plugin WalletController
@ula-aca/wallet/get-dids
@ula-aca/wallet/create-local-did
@ula-aca/wallet/fetch-public-did
@ula-aca/wallet/assign-public-did
@ula-aca/wallet/get-tagging-policy
@ula-aca/wallet/set-tagging-policy
package @ula-aca/credential
plugin CredentialController
@ula-aca/credential/get-credential-by-id
@ula-aca/credential/get-credentials
@ula-aca/credential/remove-credential
package @ula-aca/schema
plugin SchemaController
@ula-aca/schema/get-schema-by-id
@ula-aca/schema/get-created-schemas
@ula-aca/schema/create-schema
package @ula-aca/connection
plugin ConnectionController
@ula-aca/connection/get-all-connections
@ula-aca/connection/get-connection-by-id
@ula-aca/connection/create-invitation
@ula-aca/connection/receive-invitation
@ula-aca/connection/accept-invitation
@ula-aca/connection/accept-request
@ula-aca/connection/establish-inbound
@ula-aca/connection/remove-connection
@ula-aca/connection/send-ping
@ula-aca/connection/basic-message
plugin ConnectionEventHandler
@ula-aca/connection-event
@ula-aca/basic-message-event
package @ula-aca/issue-credential
plugin IssueCredentialController
@ula-aca/issue-credential/get-meme-types
@ula-aca/issue-credential/get-all-exchange-records
@ula-aca/issue-credential/get-exchange-record-by-id
@ula-aca/issue-credential/send-credential
@ula-aca/issue-credential/send-proposal
@ula-aca/issue-credential/send-offer
@ula-aca/issue-credential/send-offer-by-id
@ula-aca/issue-credential/send-request
@ula-aca/issue-credential/issue
@ula-aca/issue-credential/store
@ula-aca/issue-credential/problem-report
@ula-aca/issue-credential/remove
plugin IssueCredentialEventHandler
@ula-aca/issue-credential-event
package @ula-aca/present-proof
plugin PresentProofController
@ula-aca/present-proof/get-all-exchange-records
@ula-aca/present-proof/get-exchange-record-by-id
@ula-aca/present-proof/get-presentation-request-credentials
@ula-aca/present-proof/get-presentation-request-credentials-with-referent
@ula-aca/present-proof/send-proposal
@ula-aca/present-proof/create-presentation-request
@ula-aca/present-proof/send-request
@ula-aca/present-proof/send-proof-presentation-request-by-id
@ula-aca/present-proof/send-presentation
@ula-aca/present-proof/verify-presentation
@ula-aca/present-proof/remove-exchange-record
plugin PresentProofEventHandler
@ula-aca/present-proof-event
ULA User

Figure 4.1: Universal Ledger Agent plugin architecture for the Aries Cloud Agent

Chapter 5

Closing Words

First of all we'd like to thank Rabobank Blockchain Acceleration Lab for giving us the opportunity to work on this amazing project. Special thanks goes out to Claudia Terpstra, David Lamers and Marnix van den Bent for the excellent mentorship and frequent meetings during the entire project. We would also like to thank Jeroen van Megchelen and Bryan Georges for meeting with us and providing us with vital information to the project. We also would like to thank Lisk Center Utrecht for providing a pleasant and inspiring workspace with like minded blockchain enthusiasts and professionals. Last, but certainly not least we would like to thank the HU University of Applied Sciences Utrecht and more specifically Bernhard van der Biessen for the guidance and flexibly that has lead to this successful end result.

We worked on this project with great pleasure and have learned more than we could have wished for. Without these people, this would not have been possible.

References

Allen, C. (2016). *The Path to Self-Sovereign Identity.* Retrieved 2020-01-15, from http://www.lifewithalacrity.com/2016/04/the-path-to-self-soverereign-identity.html 2

Esplin, R. (2019). *Indy SDK Wishlist - Hyperledger Indy.* Retrieved 2020-01-17, from https://wiki.hyperledger.org/display/indy/Indy+SDK+Wishlist 4

Evernym. (n.d.). *Evernym — The Self-Sovereign Identity Company.* Retrieved 2020-01-17, from https://www.evernym.com/ 5

Hardman, D. (2017). *Indy HIPE - Agents.* Retrieved 2020-01-17, from https://github.com/hyperledger/indy-hipe/tree/master/text/0002-agents 4

Hyperledger. (n.d.-a). *Dummy Cloud Agent.* Retrieved 2020-01-17, from https://github.com/hyperledger/indy-sdk/tree/master/vcx/dummy-cloud-agent 5

Hyperledger. (n.d.-b). *Hyperledger Aries.* Retrieved 2020-01-15, from https://www.hyperledger.org/projects/aries 2, 4

Hyperledger. (n.d.-c). *Hyperledger Indy.* Retrieved 2019-11-01, from https://www.hyperledger.org/projects/hyperledger-indy 2

Hyperledger. (n.d.-d). *Indy Agent.* Retrieved 2020-01-17, from https://github.com/hyperledger/indy-agent 4

Hyperledger. (n.d.-e). *Indy SDK.* Retrieved 2019-11-08, from https://github.com/hyperledger/indy-sdk 4

Hyperledger. (n.d.-f). *VCX.* Retrieved 2020-01-17, from https://github.com/hyperledger/indy-sdk/tree/master/vcx 5

Hyperledger. (2019a). *Aries Cloud Agent - Python.* Retrieved 2020-01-17, from https://github.com/hyperledger/aries-cloudagent-python 6

Hyperledger. (2019b). *Aries Framework - JavaScript.* Retrieved 2020-01-17, from https://github.com/hyperledger/aries-framework-javascript 6

Lamers, D. (2019). *Universal Ledger Agent - Rebooting the Web of Trust 8.* Retrieved 2019-10-31, from https://github.com/WebOfTrustInfo/rwot8-barcelona/blob/3ec5d8040fd8ba48bcc54c48af466a39a428c0ff/topics-and-advance-readings/universal-ledger-agent.md 2

OpenJS Foundation. (n.d.). *Node.js.* Retrieved 2020-01-17, from https://nodejs.org/en/ 4

Reed, D., Sporny, M., Longley, D., Allen, C., Grant, R. & Sabadello, M. (2019). *Decentralized Identifiers (DIDs) v1.0.* Retrieved 2020-01-15, from https://www.w3.org/TR/did-core/ 2

Sovrin Foundation. (2018). *Sovrin ^{TM} : A Protocol and Token for Self-Sovereign Identity and Decentralized Trust A White Paper from the Sovrin Foundation* (Tech. Rep.). Retrieved from https://sovrin.org/wp-content/uploads/2018/03/Sovrin-Protocol-and-Token-White-Paper.pdf 2

Sporny, M., Dave, L. & Chadwick, D. (2019). *Verifiable Credentials Data Model 1.0.* Retrieved 2020-01-15, from https://www.w3.org/TR/vc-data-model/ 2

WebAssembly. (n.d.). *WebAssembly.* Retrieved 2020-01-17, from https://webassembly.org/ 4