## Introduction

Within Hyperledger Indy/Sovrin there is the notion of an agent or an agency respectively. The different terms are described below. In our current implementation and the IdentityChain project we use the term cloud agent as an intermediary between the ledger and the mobile app. The cloud agent fulfills the purpose of forwarding messages if the app is not online or reachable.

## Motivation

Within the IdentityChain (IDC) project the IndySDK is used on the IdentityChain API layer and the Android app layer. The API abstracts the IndySDK functionality to ease the use for developers. The Android app which is written in Java is using the IndySDK Java wrapper for exchanging values and methods with the ledger. The motivation for the introduction of a cloud agent within IdentityChain was the fact that credentials will be send, updated and revoked from the issuer's interface, in our case a web application using the IdentityChain-API. In a scenario where the app is not online and/or connected with the ledger, the IdentityChain Cloud Agent (IDC-CA) is handling temporal storing and forwarding of messages, i.e., verifiable credentials, using the Google Firebase services and APIs. The IDC-CA is also used to initiate a setup of values for the IDC Mobile App. The IDC-CA can be seen as a unique endpoint for the mobile app. In future there will be additional functionalities and use cases extending the IDC-CA. This is also inline with the notion of agents within Hyperledger Indy itself, e.g., surrogate functions, such as automated replies to certain proof requests.

## The different notions of Agents

In the following sub-sections the notion of the IDC agents is described. It may differ from the Hyperledger Indy/Sovrin meaning, which ca be found under:

https://github.com/hyperledger/indy-agent/tree/master/docs

### Edge Agent

An Edge Agent is positioned close to the user interface. If the hardware and network of the edge device is capable enough, i.e., a PC or laptop, the edge agent can talk to the ledger directly and is also hosting a wallet which includes all keys and credentials. This can also be a more potent machine, i.e., a server in an institution, such as a bank, which provides functionalities for user interfaces, such as web-based once. This edge agent is then called institutional edge agent.

Otherwise, there are mobile edge agents for devices which are not always online and need additional functionalities to connect to other agents. In our case this is a mobile app, which is the interface for the identity holder and is connected to a cloud agent.

#### Mobile Edge Agent

As described above, a mobile edge agent is a user interface device, such as a smartphone which is not always online and is connected to a cloud agent which serves it as a message broker, etc. In our case the IDChain mobile Android app is an mobile edge agent.

### Institutional Edge Agent

An institutional edge agent is capable of communicating with other edge and cloud agents and serves as the interface for the employees of the institution. In our case the institutional edge agent is the IndySDK together with the REST-API serving the front-end Admin PortalUI.

### Cloud Agent

A cloud agent is serving devices which have less capabilities such as smartphone apps. It redirects messages without knowing their contents to the specific edge agent. In our case we use key-value stores to be able to forward the incoming messages to the correct mobile using Firebase service from Google. The cloud agent can either run at an agency (service), see below or on the controlled hardware of the identity holder, such as a home NAS, Router, or similar.n It is to be noted that a cloud agent serving one app should have different endpoints for the different pairwise connections as it ensures more privacy for the identity holder. As of now, we neglect this in the IDChain project implementation, as the current version of the Indy ledger does not support complex endpoints to be stored on the ledger, at the moment only IP and Port can be stored. Agency

An agency in our understanding is the hoster of cloud agents for identity holders, this can be a telephone operator, an IT service provider, bank or another trusted entity.

### Hub (text from github indy-agent)

Additionally there is the notion of a hub in Indy/Sovrin

A Hub is much like a Cloud Agent, but rather than focusing only on messaging (transport) as defined above for a Cloud Agent, the Hub also stores and shares data (potentially including Verifiable Credentials), on behalf of its owner. All of the data held by the Hub is en/decrypted by the Edge Agent, so it is the data that moves between the Edge and Hub, and not keys. The Hub storage can (kind of) be thought of as a remote version of a Wallet without the keys, but is intended to hold more than just the Verifiable Credentials of an Edge Agent wallet. The idea is that the user can push lots of, for example, app-related data to the Hub, and a Service would be granted permission by the Owner to directly access the data without having to go to the Edge Agent. For example, a Hub-centric music service would store the owner's config information and playlists on the Hub, and the Service would fetch the data from the Hub on use instead of storing it on it's own servers. # Overview The following figure illustrates the overall IDC components also linked to the IndySDK and the ledger. At the top there are the three user interfaces for the issuer, the verifier, and an IDC-AdminUI. They are all accessing the IDC-API, which abstracts the IndySDK for the components. The IDC-Schema-Compiler which can be accessed via the IDC-AdminUI is also integrated in the API. The IDC-CA is an intermediary between the IDC-Mobile-App and the IDC-UIs and API. The IDC-Mobile-App is additionally using the IndySDK Java wrapper to access the ledger directly.

### Software Components

# Cloud Agent Message Flow

This section illustrates the message flow of the Cloud Agent and the corresponding actors.

## Message Formats

#### authcrypted

#### anoncrypted

#### Notes

- It is assumed that the TA has been previously onboarded and has Trust Anchor privileges to write on the ledger.
- FCM limits the message payload to 4KB. Messages with payloads >=4KB will be temporarily stored on the CA. The CA provides a unique URL to retrieve the message payload.
- Follow the Swagger documentation for specific formats on exchanged messages.

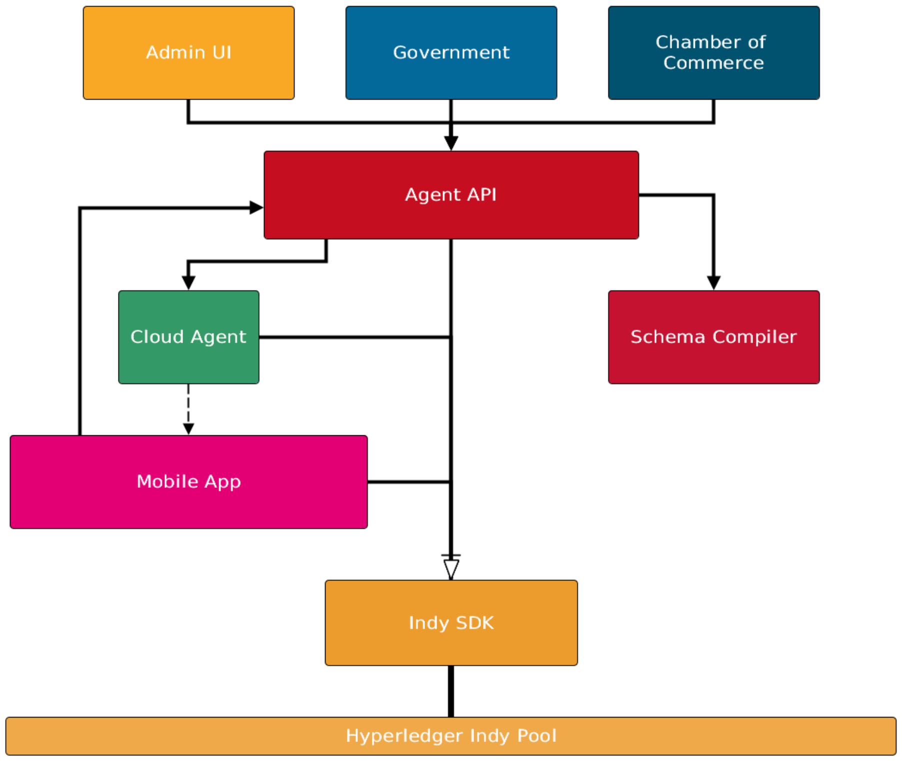

Figure 1: IDC components

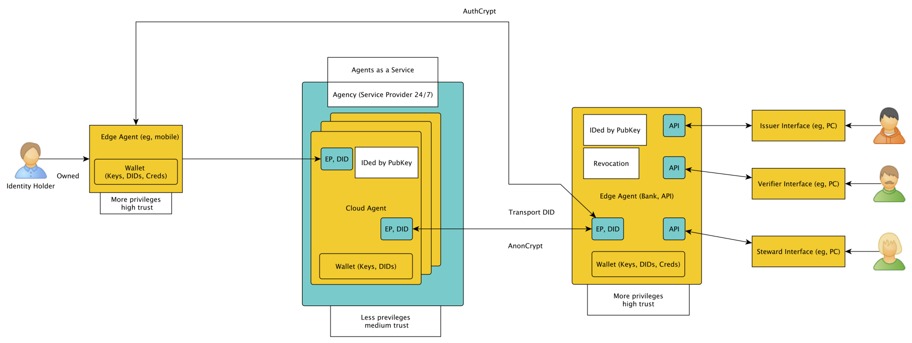

Figure 2: Agent Role Actors

## Sequence Flow

1. The IDC-CA requires an initial onboarding from a Trust Anchor (TA) in order to be known on the Ledger. Therefore it sends a connection request to the TA, which should be acknowledged by a connection response. The TA will onboard the CA with a NYM-request, including the CA_DID, the CA_VerificationKey, and the CA_Endpoint, to the ledger.

2. The IDC-Mobile-App sends a connection request to a known CA containing the App_DID, the App_Verification-Key and the Endpoint in form of a Firebase Token. The CA replies a connection response containing the CA_DID and CA_Endpoint which can be used by the APP for further communication.

3. The APP follows the pairwise connection pattern with responding to a TA_Connection_Offer with a Connection_Request and provides the CA_DID, and CA_Endpoint to be reached by the TA alongside its own DID and verification key (App_DID, App_VerKey). The TA sends a NYM_request to the ledger including the TA_APP_DID.

4. The TA sends authcrypted payload messages regarding the APP to the provided IDC-CA_Endpoint anoncrypted with the key of the CA. The CA decrypts the anoncrypted message with it's private key and forwards the payload message anoncrypted to the APP through the provided Firebase Token with FCM. The APP can directly respond to the TA_Endpoint with an authcrypted message.

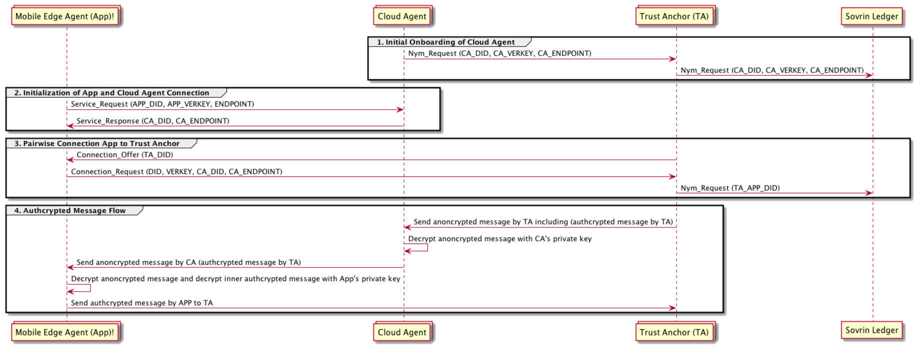
Mobile Edge Agent (App)!
Cloud Agent
Trust Anchor (TA)
Sovrin Ledger
1. Initial Onboarding of Cloud Agent
Nym_Request (CA_DID, CA_VERKEY, CA_ENDPOINT)
Nym_Request (CA_DID, CA_VERKEY, CA_ENDPOINT)
2. Initialization of App and Cloud Agent Connection
Service_Request (APP_DID, APP_VERKEY, ENDPOINT)
Service_Response (CA_DID, CA_ENDPOINT)
3. Pairwise Connection App to Trust Anchor
Connection_Offer (TA_DID)
Connection_Request (DID, VERKEY, CA_DID, CA_ENDPOINT)
Nym_Request (TA_APP_DID)
4. Authcrypted Message Flow
Send anoncrypted message by TA including (authcrypted message by TA)
Decrypt anoncrypted message with CA's private key
Send anoncrypted message by CA (authcrypted message by TA)
Decrypt anoncrypted message and decrypt inner authcrypted message with App's private key
Send authcrypted message by APP to TA
Mobile Edge Agent (App)!
Cloud Agent
Trust Anchor (TA)
Sovrin Ledger

Figure 3: CA Message Flow

## Cloud Agent Message Stack

The IDC-CA message stack is as follows. Messages from the IDC-App to the CA are transmitted via https. Messages from the CA to the app are transmitted with Google Firebase messages, unless the message is bigger than XX, in that case the Firebase message includes a link to download the message directly to the app. Messages to the CA from any other application or the API's perspective are done via https. The figure below illustrates the message stack of the CA.

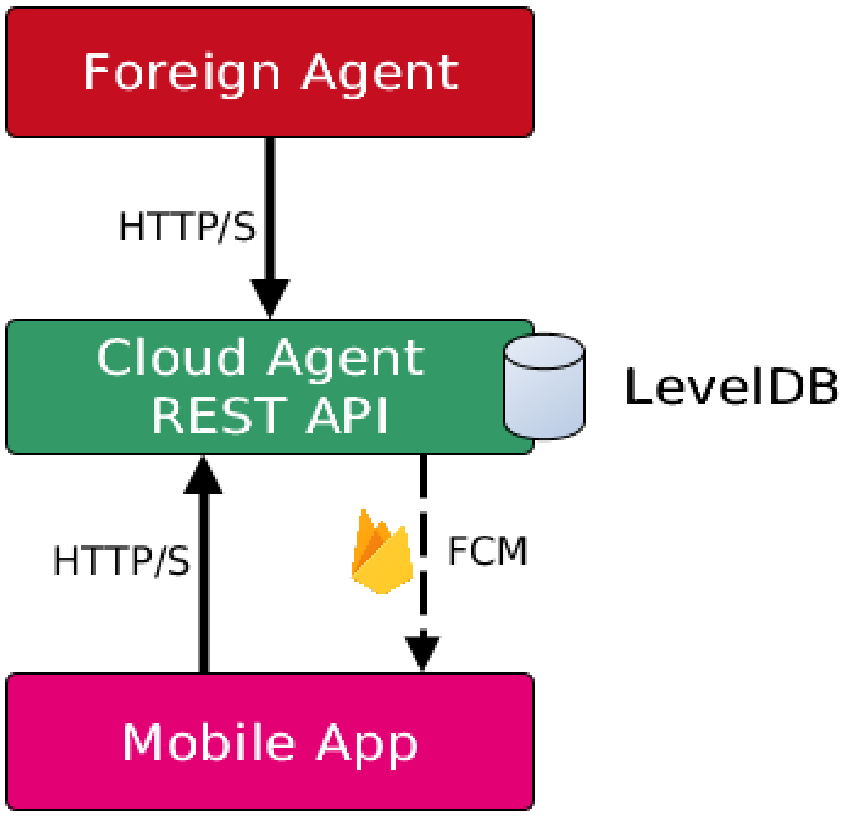

Figure 4: IDC-CA Message Stack

### Working together with other IDC components

### Differences between the Indy/Sovrin CA and IDC CA

## Setup

### Deployment of the Cloud Agent/Agency* API

The deployment of the cloud agent/agency is described in the following.

## Prerequisites

For a brief overview on which parameters are required for a working deployment, check the example.env in the repository.

### Firebase Cloud Messaging

The implementation of the IDChain Cloud Agent implements Firebase Cloud Messaging (FCM) for communication with mobile applications. A prior registration of a project at http://console.firebase.google.com is required to use FCM. Set the following env-variables accordingly. The file we currently used was uploaded to the servers. Current name: 'eit-idchain-app-firebase-adminsdk.json'. It was generated with Bersant Google account and used for the time being.

`FIREBASE_ADMIN_PATH='/path/to/xxx-adminsdk.json'`

`FIREBASE_PROJECT_URL='https://xxx.firebaseio.com'`

### LevelDB

The Cloud Agent uses LevelDB for storing data and keeping track of the connections between unique URLs provided to App instances and FCM Tokens. Make sure the application has write access to the path set as described below.

`DB_PATH='./data'`

### Hyperledger Indy Pool Access

The Cloud Agent requires access to a Hyperledger Indy ledger pool in order to verify DIDs and Endpoints. For this connection a name and the pool transactions genesis of the pool is required.

`POOL_IP=172.16.0.100`

`POOL_NAME=poolApi`

`GENESIS_TXN=pool_transactions_genesis.docker-compose`

## Deployment Parameters

The application runs on the host and port defined in the environment variables.

`CA_APP_HOST=127.0.1.1`

`CA_APP_PORT=8080`

In addition for providing unique URLs to mobile apps, the application requires additional settings for the domain and port where the Cloud Agent is actively run at.

```
DOMAIN_HOST=127.0.1.1
```

```
DOMAIN_PORT=8080
```

Finally, the Cloud Agent implements a Hyperledger Indy Wallet which is used for storing own and foreign DID information, encrypting and decrypting outgoing and incoming messages. Therefore the following env-variables need to be set.

```
CA_WALLET_NAME='example_ca_wallet_name'
```

```
SECRET='your-secret'
```

```
CA_DID='Cloud Agent DID for initial onboarding'
```

## Deploy the App

It is recommended to deploy the Cloud Agent API behind an Nginx webserver and let Nginx act as a proxy for incoming queries to the API. Also, in order to restart and reboot easily and accessing monitoring of the API we recommend the deployment with pm2.

### PM2

Here are some helpful commands with pm2.

#### Generating Server-specific Startup Scripts

Generating server-specific startup scripts for reboot ready startup. This command will check the underlying server environment and will propose the fitting script to be executed.

```
pm2 startup
```

```
pm2 startup ubuntu
```

#### Start/Restart an application with pm2

```
pm2 start app.js -n your_app_name
```

```
pm2 restart your_app_name
```

#### List all applications

```
pm2 list
```

#### Monitor all applications

```
pm2 monit
```

#### Show application-specific information

```
pm2 show your_app_name
```

#### Show application-specific logs

```
pm2 logs your_app_name
```

### Docker

Dockerfiles added

Deployment with docker-compose thru IDChain commons, uses local Dockerfile

In IDChain commons there is a overall en file for settings, also including all other IDChain components

Updated: October 26th, 2018

Note: *Cloud Agent/Agency are currently terms used interchangeably. There seems no clear and valid written description on roles and components in Hyperledger Indy yet (as of 09/03/2018).

```
How is the CA setup and configured
    Working together with ledger
    How does the ledger need to be configured
Setup of IDC App
    IDC Test App integration
Test - how to test a certain CA setup
```

## API

```
What does the CA API calls do
How can a developer use the API
```

## Encryption

```
What kind of encryption is used between the different components
Authcrypted
Encrypted
```

## Developer

```
What does a dev need to know about the CA
How can the functionality be extended
```

## Example

```
Example Setup
```

## Open Issues

```
What needs to be done
Roadshow - Next implementation steps
```

## License

```
What is the open source license of the CA
```

## Contact

```
Contact the developers
```