university of groningen / faculty of science and engineering

IBM

Self-Sovereign Identity for the Internet of Things: A Case Study on Verifiable Electric Vehicle Charging

Filipe Alexandre Rosa Capela

university of groningen / faculty of science and engineering

IBM

Rijksuniversiteit Groningen

Self-Sovereign Identity for the Internet of Things: A Case Study on Verifiable Electric Vehicle Charging

Master's Thesis

To fulfill the requirements for the degree of
Master of Science in Computing Science
at the University of Groningen under the supervision of

dr. Vasilios Andrikopoulos (Computer Science, University of Groningen)
prof. dr. Dimka Karastoyanova (Computer Science, University of Groningen)
Arne Rutjes (Blockchain Practice Lead Europe, IBM Client Innovation Center)
Arjen van Veen (Lead Developer Self-Sovereign Identity, IBM Client Innovation Center)

Filipe Alexandre Rosa Capela (s4040112)

July 10, 2021

Contents

Page

List of Figures v

List of Acronyms vi

Acknowledgements viii

Abstract ix

1 Introduction 1
- 1.1 Research Question . . . 1
- 1.2 Thesis Outline . . . 2

2 Background & Related Work 3
- 2.1 Background Concepts . . . 3
  - 2.1.1 (Digital) Identity . . . 3
  - 2.1.2 Identity Management Systems on the Internet . . . 4
  - 2.1.3 Paper Credentials Model . . . 5
- 2.2 Fundamental Concepts . . . 6
  - 2.2.1 Self-Sovereign Identity . . . 6
  - 2.2.2 Verifiable Credentials Model . . . 7
  - 2.2.3 Decentralized Identifiers . . . 7
  - 2.2.4 Verifiable Credentials . . . 9
  - 2.2.5 Zero-Knowledge Proof . . . 10
  - 2.2.6 Revocation of VCs . . . 10
  - 2.2.7 Management of DIDs and VCs . . . 10
  - 2.2.8 Relation with Distributed Ledger Technology . . . 11
- 2.3 Blockchain-based Self-Sovereign Identity with IoT . . . 11
  - 2.3.1 Blockchain as an IdMS . . . 11
  - 2.3.2 Link between SSI and IoT . . . 12
- 2.4 Existing Technologies . . . 14
  - 2.4.1 Hyperledger Indy / Sovrin . . . 15
  - 2.4.2 IOTA Tangle/Identity . . . 15
  - 2.4.3 Other Technologies . . . 16

3 Case Description and Requirement Elicitation 17
- 3.1 Case Description . . . 17
  - 3.1.1 Users and Information Flows . . . 17
  - 3.1.2 Current Liabilities . . . 18
- 3.2 List of requirements . . . 19
  - 3.2.1 Key Drivers . . . 19
  - 3.2.2 Functional Requirements . . . 19
  - 3.2.3 Non-Functional Requirements . . . 20

4 System Design 22
- 4.1 Technological Decisions . . . 22
  - 4.1.1 Distributed Ledger . . . 22
  - 4.1.2 Agent Technology . . . 23
  - 4.1.3 Mobile Agent . . . 25
- 4.2 Architecture for SSI with IoT . . . 25
- 4.3 Electric Vehicle Charging System with SSI . . . 26
  - 4.3.1 Architecture of the System . . . 26
  - 4.3.2 Schemas and Credential Definitions . . . 28
  - 4.3.3 Information Flows with SSI . . . 29

5 Implementation and Deployment **37**
5.1 Implementation Details . . . 37
5.1.1 Agent Configuration . . . 37
5.1.2 Adopted Aries RFCs . . . 38
5.1.3 Application Logic . . . 39
5.2 Deployment Details . . . 41
5.2.1 Steps for the deployment . . . 41
5.2.2 Steps for running the use case scripts . . . 43

6 Evaluation **46**
6.1 Methodology . . . 46
6.2 Benchmark "Hyperledger Indy Network" and "Hyperledger Aries Cloud Agent - Python" . . . 46
6.3 Qualitative Evaluation from domain experts . . . 48
6.3.1 Preliminary Form Results . . . 48
6.3.2 Results from experts' evaluation of system . . . 49
6.4 Cross-check requirements . . . 50
6.4.1 Functional Requirements . . . 50
6.4.2 Non-Functional Requirements . . . 51
6.4.3 Limitations . . . 52
6.5 Evaluation of the General Architecture of SSI with IoT . . . 54

7 Conclusion **55**

8 Future Work **56**

9 Bibliography **57**

Appendices **59**
A How to Install Trinsic Wallet Agent Mobile Application . . . 59
B Application Screenshots . . . 60
B.1 Service Providers Flow with SSI Screenshots . . . 60
B.2 EV Owner and EV Interactions with SSI Screenshots . . . 62
B.3 Charging and Billing with SSI Screenshots . . . 64
C Aries Cloud Agent - Python Benchmark Runs . . . 69
C.1 No Revocation + Mediation . . . 69
C.2 Revocation + No Mediation . . . 69
C.3 No Revocation + No Mediation . . . 70
C.4 Revocation + Mediation . . . 70
D Requirement Evaluation . . . 70

List of Figures

1 Structural View on the concept of (Digital) Identity . . . 3
2 Interactions between different parties in a traditional IdMS . . . 4
3 Paper Credential Model . . . 6
4 The Verifiable Credentials Model . . . 7
5 Overview of DID architecture . . . 8
6 Example of how attributes from a paper crendential can be stored in a Verifiable Credential . . . 9
7 Relation between credential schema and credential definition . . . 9
8 High-level overview of Case Study . . . 18
9 Aries Agent Architecture . . . 24
10 General Architecture for SSI with IoT . . . 26
11 High-Level Architecture Electric Vehicle Charging System with SSI . . . 27
12 Service Providers Flow System Sequence Diagram . . . 31
13 EV and EV Owner Interactions System Sequence Diagram . . . 33
14 Charging and Billing Use Case System Sequence Diagram . . . 36
15 Implementation Components Architecture and Protocols . . . 38
16 Call stack to Start Credential Issuance Process . . . 39
17 Docker Desktop with running containers . . . 42
18 pgrok in action, tunneling the localhost ports to specific endpoints, depending on the configuration . . . 42
19 How to extract the connection IDs after executing the script . . . 43
20 How to extract the schema and credential definition IDs after executing the script . . . 43
21 Sovrin Network (Hyperledger Indy nodes) metrics from 12 June 2020 to 12 June 2021 . . . 47
22 Thread discussing future plans for ACA-Py on the RFC-104: Chained Credentials . . . 53
23 Current Electric Vehicle and Charging Station protocols . . . 54

List of Acronyms

ACA-Py	Aries Cloud Agent - Python. 24
CP	Control Party. 4
CPO	Charging Point Operator. 17
CS	Charging Station. 17
DID	Decentralized Identifier. 6
DIDComm	DID Communication. 8
DLT	Distributed Ledger Technology. 5
eMSP	eMobility Service Provider. 17
EP	Electricity Provider. 17
EV	Electric Vehicle. 17
EV Owner	Electric Vehicle Owner. 17
FR	Functional Requirement. 19
GDPR	General Data Protection Regulation. 21
IdMS	Identity Management System. 1
IdP	Identity Provider. 4
IoT	Internet of Things. 1
IoV	Internet of Vehicles. 1
ISO	International Organization for Standardization. 1
ITU	International Telecommunication Union. 3
NFR	Non-Functional Requirement. 20
OEM	Original Equipment Manufacturer. 17
PII	Personally Identifiable Information. 20
REST	REpresentational State Transfer. 24
RFC	Request For Comment. 24
RP	Relying Party. 4
SSD	System Sequence Diagram. 2

SSI Self-Sovereign Identity. 6

SSO Single Sign-On. 5

TA Transportation Agency. 18

TPS Transactions Per Second. 19

VBC Vehicle Birth Certificate. 17

VC Verifiable Credential. 6

VIN Vehicle Identification Number. 17

VON Verifiable Organizations Network. 23

W3C World Wide Web Consortium. 6

ZKP Zero-Knowledge Proof. 6

Acknowledgments

It is with immense gratitude that I acknowledge the support and help from my supervisors from both the IBM CIC Groningen, Arjen van Veen and Arne Rutjes, as well as prof. dr. Dimka Karastoyanova and dr. Vasilios Andrikopoulos, from the Rijksuniversiteit Groningen for their valuable suggestions, ever encouraging and motivating guidance, as well as the trust deposited in me during the span of this project is something I will forever cherish.
A special word goes to dr. Vasilios, for helping me overcome all the limitations faced during this project, and for being available as much as one can be, to provide me with detailed feedback on this manuscript as well as during the course of this Master project.
I consider it an honor to have performed this work under the opportunity provided to me by IBM, a company that has given me all the tools necessary to conduct my research.
I wish to thank the University of Groningen, for allowing me to develop myself in order to become a better student, thinker and person.
I would still like to extend my gratitude to all of the IBM professionals I had the chance to contact with, as well as the people that have taken the time of evaluating my system during its development.
I wish to dedicate this thesis first and foremost to my girlfriend, Rita Gaspar, for pushing me everyday to give my all in any circumstance. To my family, for supporting me every single day and allowing me to fulfil my dreams. To my friends José Vicente, Inês Paredes and Diogo Nogueira, for their support, constant friendship and care. To all of you, I would like to say a few words in my mother tongue, Portuguese: "*Eu amo-vos muito e espero ter-vos deixado orgulhosos*".
Gostaria de dedicar esta tese em primeiro lugar à minha namorada, Rita Gaspar, por me motivar todos os dias e a dar tudo de mim em todas as circunstâncias. À minha familia, por me apoiarem todos os dias e permitirem-me atingir os meus sonhos. Aos meus amigos José Vicente. Maria Inês Paredes e Diogo Nogueira, pelo constante apoio, amizade e carinho. A todos vocês gostaria de dizer que vos amo muito e espero ter-vos deixado orgulhosos.

Abstract

Self-Sovereign Identity (SSI) is a new approach on data privacy that puts data in the hands of its owner. These concepts were previously only tied to humans, and for this reason the assessment of the requirements to use these concepts to manage the digital identity of Internet of Thing (IoT) devices is a topic that needs further investigation. This study aims to understand the requirements that could enable an SSI-driven system with IoT devices. The focus is set on Electric Vehicle Charging Networks, and how the current system could be migrated to an approach that eliminates the identified privacy, security and transparency issues. To achieve this, a proof-of-concept was developed to assess how the new system would compare to the previous one, and understand the constraints introduced by this new approach. Making use of Hyperledger Indy and Aries, two projects centered around providing means for SSI, a novel system was built that allows users to hold their data in their own virtual wallets, which they can later use to present any claims needed by the system. While investigating the drawbacks of this system, the immaturity of the ecosystem was identified as the most pressing matter to effectively implement the envisioned system, as well as eliminating possibly introduced privacy issues. Aided by contributions from domain experts, a positive assessment of the system was made, with the unfulfilled requirements being outlined and with a road-map being designed to mitigate these flaws.

1 Introduction

"Trust is broken on the internet": this is the general idea that is passed in the 2019 "Internet security & trust" survey conducted by the Center for International Governance Innovation [1] as well as in news articles[1], and one of the many reasons for the appearance of a new concept called Self-Sovereign Identity (SSI). The field of Self-Sovereign Identity tackles the difficulty to manage digital identities on the internet by means of its Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs), which puts governance into peoples hands, allowing them to discretize only the information that is needed, while having the power to revoke that same information. According to a study published by Statista[2], more than half the people in the world are using the internet, which means all of these have the potential to create an account in any service available, or 10, or 100, all of which are generated in approximately the same manner: the user is requested information that the company managing that service will hold on to, in order to create an account for that user, allowing it to access the service via an id (usually the email or an username) and a password combination. In a research conducted by NordPass[3], a company specialized in password management, they claimed that the average person has 70-80 passwords that they need to memorize[4], which shows how easy it can be for someone to lose track of their credentials and lose access to their own data, which in turn is solely put in the hands of the companies. Nowadays, one of the most widely used forms of identity management system relies on federated solutions, which allow companies to share data on one customer, and allow the customer to have one login credential to assess multiple systems. This somewhat addresses the issue of having too many passwords to memorize, but still allows multiple companies to hold the data on behalf of its owner. The latter is one of the many matters that aims to be addressed by SSI.
Bridging the topic to the field of Internet of Things (IoT), providing digital identity to edge devices is a discrete use case that is not accounted for in many Identity Management Systems (IdMSs), where the digital identity is mostly linked to people or enterprise systems that are also managed by people [2]. The adoption of IoT devices has been increasing at a fast pace according to the "*New Eclipse Foundation's IoT Commercial Adoption Survey*" conducted by the "Eclipse Foundation" [5] where 40% of the industries surveyed reported the intent to supplement their systems with IoT. Also, in a surveyed report by Business Insider [6], it is expected that there will exist close to 41 billion IoT devices connected by the year 2027, which forces more discussion on security and privacy in such an expanding market. The International Organization for Standardization (ISO) defined IoT as "An infrastructure of interconnected objects, people, systems and information resources together with intelligent services to allow them to process information of the physical and the virtual world and react" [3]. Allowing IoT devices to have their own identity sheds light into new possibilities, since nowadays there is substantial presence of "autonomous" or "smart" devices, which can act and make decisions alone or partially aided by humans. These decisions generally involve communication between different devices, and using SSI to prove that the devices are who they claim they are enlightens many use cases to be more secure and less prone to attacks [4] [5]. A concrete example of an application of SSI with IoT, which was also be the scope of this project, is the necessity for electric vehicles to ensure they are who they are in order to connect to charging stations, which in turn will operate and manage with the energy grid to supply energy to the vehicle, depending on variables inherent to each vehicle and its owner.

1.1 Research Question

This thesis, conducted with the collaboration of the IBM Client Innovation Center in Groningen and with the assistance of their domain expertise, has the purpose to tackle the following research question:

RQ. *What are the requirements for Self-Sovereign Identity (SSI) to be implemented effectively in an Internet of Things (IoT) environment?*

In order to assess this, a particular type of IoT devices was selected for assessment, namely the Internet of Vehicles (IoV) devices.
For this end, the following work aims at exploring a case study under the contours defined in the "Guidelines for conducting and reporting case study research in Software Engineering" [6].

[1] https://www.startupdaily.net/2019/05/csiro-the-internet-is-fundamentally-broken-when-it-comes-to-trust-and-malicious-threats/
[2] https://www.statista.com/topics/1145/internet-usage-worldwide/
[3] https://nordpass.com
[4] https://southernmarylandchronicle.com/2020/02/26/new-research-an-average-person-has-more-passwords-than-an-average-pop-song-has-words/
[5] https://www.eclipse.org/org/press-release/20200310-iot-commercial-adoption-survey-2019.php
[6] https://www.businessinsider.com/internet-of-things-report?international=true&r=US&IR=T

Knowledge from domain experts was used to answer the aforementioned research question. The case study that was analysed in this project corresponds to an SSI-powered Electric Vehicle Charging Network in order to mitigate the flaws detected in the current implementation of the system.

1.2 Thesis Outline

In Chapter 2 an explanation of the concepts and background information is provided, alongside an in-depth look at the current state-of-the-art in the field of Self-Sovereign Identity with IoT. A brief introduction to Identity Management Systems is given, followed by how the evolution of these lead to blockchain concepts being used to securely store and validate identities (used for SSI). Also a detailed explanation of SSI is provided, by providing more specifications and definitions of its core concepts (DIDs, VCs,...) Also in this chapter an overview of the current technologies and application domains is presented to display the current state-of-the-art when developing using SSI concepts.
In Chapter 3, the assumptions and requirements related to the selected use case of Electric Vehicle Charging are developed, which includes network topology, frameworks and other variables.
Chapter 4 presents the design of the Verifiable Electric Vehicle Charging system. This includes a description of the technology stack motivations, the components of the system as well as a description of the VCs included in the novel architecture. Furthermore, this chapter presents a thorough description of each different flow in the system, making use of System Sequence Diagrams (SSDs).
In Chapter 5 the implementation and deployment details are presented, illustrating how the proof-of-concept was developed, the architecture of the latter, and what are the steps needed to reproduce the system, with the aid of code listings.
Chapter 6 contains the evaluation of the research topic, as well as the system. It contains the methodology used to assess the requirements fulfillment, with the support from benchmarks to the system's components as well as feedback from domain experts on the presented solution. At the end of this chapter a discussion on the more relevant requirements is made to highlight some of the more pressing matters to address in the future, or the capabilities of the new system that enable key requirements.
The thesis concludes with Chapter 7 and Chapter 8, with a summary of the main contributions as well as a reflection on the future work for this project, respectively.

2 Background & Related Work

In this chapter the background information necessary to comprehend Self-Sovereign Identity will be presented. The chapter will start with background concepts to provide some context to the topic, and then it will proceed with what Self-Sovereign Identity is and its core components. Then, related work to this thesis will be presented by means of establishing the state-of-the-art in the area of SSI in the IoT domain.

2.1 Background Concepts

In this section an overview of the concepts that will be covered during this project will be given in order to provide context and guidance to the rest of this study. At the most introductory level the concept of identity will be grasped, followed by how identities are represented in a digital environment. Afterwards a roadmap will be made on the key architectures for Identity Management Systems on the Internet, where different systems will be covered.

2.1.1 (Digital) Identity

At the time of writing, the way people perceive identity in the real world differs from an actual implementation in the digital world. Before elaborating in this statement, it is best to provide a definition of identity from a philosophical point of view. Looking at Aristotle's Law of Identity, defined in formal logic as "**A is A – for any A - Everything is itself**" which can be translated to "each thing is identical with itself" [7].
Looking at identity from a more recent and practical perspective, provided by the International Telecommunication Union (ITU), identity refers to the representation of an entity in the form of attributes that allow for an entity to be distinguished in a given context [8].
On a real-life example, the identity is something originated on a per-relationship basis and that can change over time. The way we present ourselves to someone and the information that recipient gathers is what creates an identity within that relationship. When a person introduces themselves as "Alice" to someone, the other end will process information such as the name, the face, the voice and other relevant attributes to formulate an identity for that specific relation. Transporting the concept of identity to the digital world is something that was not envisioned when the internet was created. As a result, an identity layer was missed, leaving users without full control over their identities and leaving the internet designed in a non-user-centric style.
Transporting ITU's definition of identity into the concept of Digital Identity leads to the following definition: "A digital identity is the digital representation of an entity detailed enough to make the individual distinguishable within a digital context" [8]. A (digital) identity can consist of identifiers (UserID, email, URL, etc.), credentials (certificates, tokens, biometrics, etc.) and attributes (roles, positions, privileges, etc.)[7] and a clear view of how a subject can hold many identities with different identifiers, credentials and attributes can be seen in Figure 1.

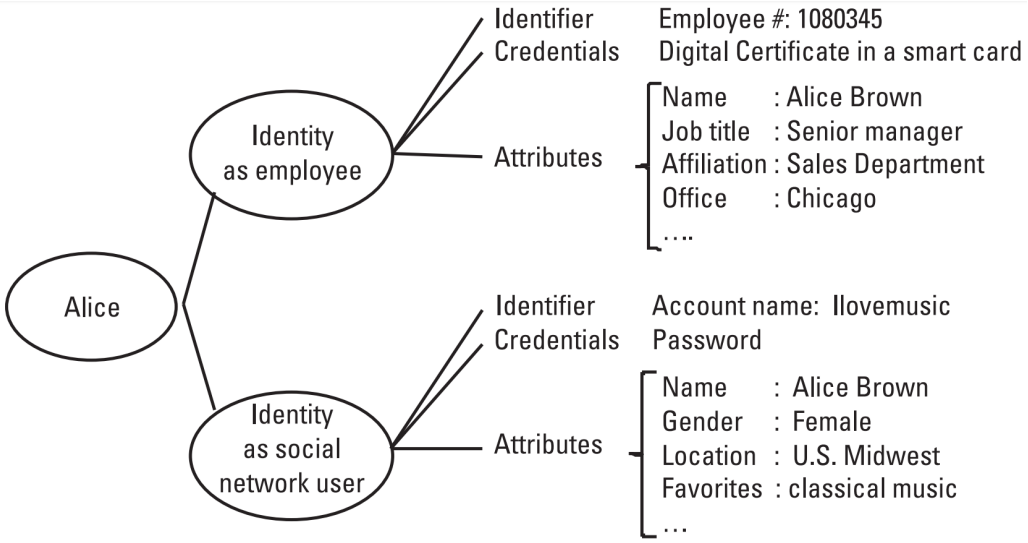

Figure 1. Structural View on the concept of (Digital) Identity [9]

[7] https://www.itu.int/rec/T-REC-Y.2720-200901-I/en

2.1.2 Identity Management Systems on the Internet

Identity Management Systems (IdMSs) are essentially a framework with a predefined set of rules on how digital identities are meant to be commissioned, managed and revoked. In this section light will be shed on the current implementations available to store and manage identities on the Internet, dividing it between two paradigms: Traditional and Decentralized IdMSs.

Traditional IdMSs The way Traditional Identity Management Systems were designed was on the basis of three or four actors: Subjects, Relying Parties (RPs), Identity Providers (IdPs) and in some cases Control Parties (CPs) [9]. Note that depending on the literature the nomenclature for "Relying Parties" has other definitions such as "Service Provider" [10]. The traditional IdMSs are mainly dependent on a centralized IdP that performs operations of creating, updating, managing and deleting identities of users [11]. To understand clearly what each party represents, a description extracted from the book by Bertino and Takahashi (2010) [9], can be found below.

- "**Subjects** are the parties whose identity attributes are digitally recorded and used for transactions and other purposes."
- "**Identity Providers** are the parties that provision identities to subjects."
- "**Relying parties** are the parties that in order to provide services to users (or agents on behalf of users) or access to resources require the submission of proper credentials by users."
- "**Control parties** are typically law enforcement agencies and regulatory bodies that may need access to identity information."

The way these parties interact with each other can be summarized using the communication diagram shown in Figure 2 where it is clear the flow of information between Subjects, RPs and IdPs. Note that since most IdMSs do not require a CP, it has been exempt from this example, but if placed in the diagram it would fall in the center of the triangle and would communicate with all parties as it acts as a regulatory party. The parties depend on each other in the following way: the subject requests access to services from the RP, which in turn requires the IdP to challenge the subject identity through the authentication protocol [10].
At the dawn of identity management, the solution was truly centralized, since the RPs and the IdPs are combined into one unique entity that acts as the resource server while holding the username and password. This way, traditional identity management models face many challenges including the fact that they rely on a centralized architecture, which in turn leads to single-point-of-failure faults. Also these architectures are more prone to common attacks (e.g. Distributed Denial of Service) given there is a centralized server that can be targeted. Furthermore, in an IoT scenario scalability issues arise, also due to centralization issues. Lastly, and given that most things happen under-the-hood, most systems operate in a non-transparent manner [11]. This lifts privacy matters, since the users are not fully aware of the practices of an organization, which for example might include their data being sold to 3rd parties.

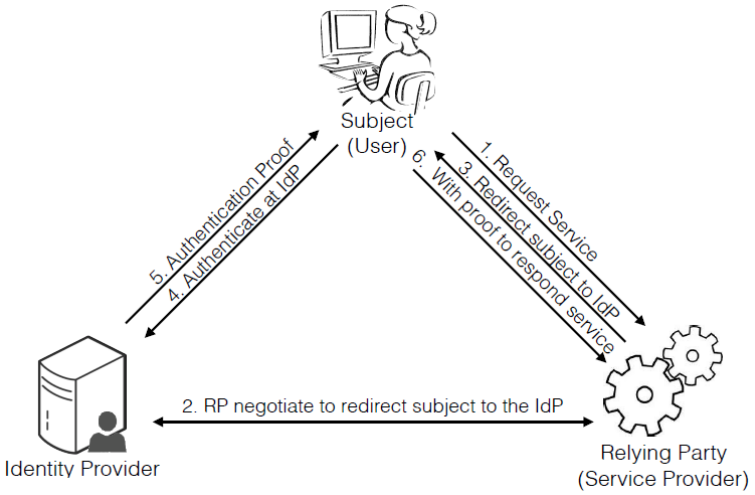

Figure 2. Interactions between different parties in a traditional IdMS [10]

There is not a single model on how to utilize the Traditional IdMSs. There are four (4) different definitions of models: Isolated, Centralized, Federated and User-Centric. Depending on the systems requirements, a different model is used. However, in general terms, the past IdMSs have evolved from isolated to centralized, and then to federated/user-centric models [12].

Isolated Models are similar to what the web reassembles to today in its majority. Users create different identities for each service they use (mostly usernames and passwords) and they are required to memorize each different identity since the IdP and the SP are combined into one. For example, Facebook creates the identity of the user, but also manages it and manipulates it as they wish (according to regulations). Since it is up for the users to memorize their identities, and since each service requires different parameters of security (e.g. password must contain 12 characters, 5 numbers and the square root of π) this can possibly lead to mismanaging identities, password reuse and forgetting credentials [11].

Centralized Models differs from the Isolated Model since it makes use of Single Sign-On (SSO), enabling multiple SPs under the same authority to make use of a single IdP. This helps reducing the number of credentials a user needs to memorize, but adds the security liabilities of having a single point of failure, lack of scalability and also introduction to other vulnerabilities and attacks.

Federated Models are similar to Centralized, with the exception that instead of having one authority that controls the Single Sign-On, a consortium of authorities agree with one login that serves for many different services, hence reducing credential management on the user side.

User-Centric Models are more aligned with the need to shift to a paradigm that puts governance in the hands of the users who own the data. This model revolves around users storing the credentials they have in either smart cards or smartphones, but still having the credentials stored in a centralized system provided by the SP. Although it was a step in the right direction it still suffered from the issues causes by relying on a centralized architecture.

Decentralized IdMSs With problems such as single point of failure of a centralized IdMS, the privacy concerns that have been reported (Facebook and Equifax data breaches [10] [13] [8]), the approach shifted to a decentralized alternative. This approach has in its core the concept of using blockchain, a type of Distributed Ledger Technology (DLT).

Blockchain was first introduced by Satoshi Nakamoto in the Bitcoin Whitepaper [14] and as been defined by the ISO standards as a "distributed ledger with confirmed blocks organized in an append-only, sequential chain using cryptographic links" [15]. Blockchain consists of a decentralized structure composed of blocks which are linked together in a chronological order. In the Bitcoin implementation proposed by Nakamoto, a Proof-of-Work consensus algorithm is used, this reflects on how the transactions within each block are confirmed and how the blocks themselves get added to the chain. In this consensus mechanism that is allowed by miners who compute highly computational intensive tasks which in turn may return tokens. This latter statement is not the case for all blockchain implementations, but rather it was an example to highlight the usage of blocks and a consensus mechanism.

Blockchains can be of two types: permissioned, when the participants of the network are provided access to it, whereas in a permissionless implementation anyone can enter the network as a new node, which will hold the history of the blockchain, providing a decentralized way for data to be persisted [16] [17].

With DLT it is possible to register every single transaction that gets added to the blockchain. Approaches have been made towards creating such decentralized IdMSs, which include the contributions of big companies such as IBM, and are good starting points for identity to be managed in a more private, secure and distributed way[9].

2.1.3 Paper Credentials Model

Before diving into Self-Sovereign Identity and understanding how trust can be built around these concepts, it is important to realize how humans have been exchanging information and validating their claims in the past, and still to the time of writing. According to the Oxford Languages dictionary, credential is defined as a "qualification, achievement, quality, or aspect of a person's background, especially when used to indicate their suitability for something"[10]. Essentially, a credential can be anything that proves that an individual is eligible for a particular ability (e.g. a driver's license card for driving). These do not need to take the format of a card, but it needs to be something that can bear a certificate that proves it was issued by a trusted party (e.g. the government hologram on the driver's license card). This way, any individual can guarantee that that credential is valid and legitimate. To put it simply, this model handles claims in a trust-based system, since there is no actual infallible method to prove a claim is real. The basis for this system is composed of three (3) actors: An **issuer**, a **holder** and a **verifier**. Essentially their relationships are summarized in Figure 3, where it is clear that the issuer issues a credential to a holder, which when necessary will present it to a verifier. If the verifier trusts the entity which issued the document, it will acknowledge the holders claims as true.

[8] https://files.iota.org/comms/IOTA_The_Case_for_a_Unified_Identity.pdf
[9] http://handle.itu.int/11.1002/1000/11559
[10] https://www.lexico.com/definition/Credential

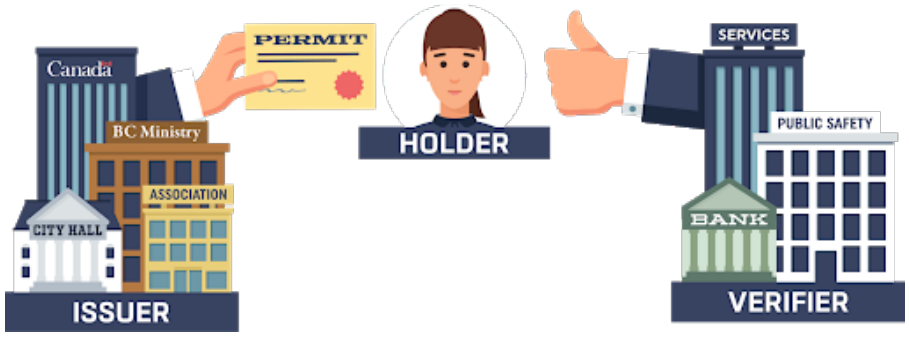

Figure 3. Paper Credential Model[11]

At its early beginning (which reports back to the Persian era [Nehemiah 2:1–9]), it was primarily used to allow people with authorization to circulate between lands, provided they could show a document signed by an official authority. Due to the inability to forge documents and create fake copies of an official document, this model worked. But with time, and with the appearance of printers, scanners and software that allows image manipulation, the forging of documents became much easier. That leads to today's age, where the people who need to check for the authenticity of a document (the verifiers) need to be experts in the art of detecting forged credentials.

2.2 Fundamental Concepts

In this section the most crucial concepts for this work are presented, with an emphasis on properly introducing the concept and the core enablers of Self-Sovereign Identity. A description of the updated Paper Credentials Model, named Verifiable Credentials Model is given. This is followed by a detailed explanation on the pillars of Self-Sovereign Identity, namely Decentralized Identifiers and the underlying communication protocol DIDComm, Verifiable Credentials and Zero-Knowledge Proofs. The World Wide Web Consortium (W3C) has been developing specifications that are widely accepted as standards on these topics, and therefore the discussion will be based on these specifications, whose references will be contained in each section.

2.2.1 Self-Sovereign Identity

Self-Sovereign Identity (SSI) is a key enabler of people being in charge of their own identities. Although it is a field which is still maturing as of the time of writing, it has been coined by Juniper Research as a field which will generate $1.1 billion in annual revenue by the end of 2024. [12]
The term was deeply detailed in an article by Christopher Allen, where he provides a brilliant bridge between the IdMS topic covered before, and the field of SSI:

> *Self-sovereign identity is the next step beyond user-centric identity and that means it begins at the same place: the user must be central to the administration of identity. That requires not just the interoperability of a user's identity across multiple locations, with the user's consent, but also true user control of that digital identity, creating user autonomy. To accomplish this, a self-sovereign identity must be transportable; it can't be locked down to one site or locale.*[13]

SSI is set to become the next standard for internet communication, as it holds potential to solve many of the internet current trust issues. A brief overview of it can be coined in the following terms: Each entity becomes in charge of its own identity, which means that each user is capable of disclosing only the information and data which they deem relevant or necessary in any relationship they hold. Also, each entity is capable of revoking any piece of data from a relationship or even revoking a relationship entirely, without the need from the other party to consent to this [18] [19]. It follows the path of the Decentralized IdMSs that were previously explained, since it utilizes a Distributed Ledger Technology (the blockchain), to enable the usage of Decentralized Identifiers (DIDs), Verifiable Credentials (VCs) and enhanced cryptography to support other features such as Zero-Knowledge Proofs (ZKPs) [9]. These concepts will be discussed in more detail in the following sections. With the core features mentioned above, SSI disables the need for a central authority

[11] https://courses.edx.org/assets/courseware/v1/cb46ed78abdc7583aa95d011426676e0/asset-v1:LinuxFoundationX+LFS172x+3T2019+type\spacefactor\@m{}asset+block/The_Paper_Credential_Model.png
[12] https://www.juniperresearch.com/press/press-releases/self-sovereign-identity-to-be-a-billion-dollar
[13] http://www.lifewithalacrity.com/2016/04/the-path-to-self-soverereign-identity.html

to hold any data, and also provides a tamper-proof system that provides transparency and less possibility of forging any credential. Because of the underlying cryptography and blockchain technology, SSI means that an entity can present claims about its identity and others can verify it with cryptographic certainty [20].

2.2.2 Verifiable Credentials Model

While the Paper Credential Model worked for a long time, due to its many issues with possible forged documents (as documented in Section 2.1.3), with SSI it is possible to implement the concept of the Verifiable Credentials Model. In 2019 the Verifiable Credential Model has been established as a standard by the W3C [21] [22].
The foundation for this model is still the same as the Paper Credential Model, there are still 3 parties, namely the Issuer, Holder and Verifier, but a Verifiable Data Registry is added to the picture as a base truth for all the claims that are exchanged in the process.
In order to illustrate each role in this model, the following example will be described making reference to the roles and actions performed by a government, an individual and a vehicle rental agency, in the event of an individual wishing to rent a vehicle. The government (issuer) will emit a signed Verifiable Credential (or Claim) to the individual (holder) attesting that they are eligible to drive. The vehicle rental agency (verifier) will request proof that the individual (holder) is eligible to drive. The individual (holder) presents the claim to the vehicle rental agency (verifier), which will check in the Verifiable Data Registry whether the government (issuer) is an entity they trust. Given the claim is cryptographically signed, this prevents it from being a forged document, hence if the vehicle rental agency has the government as a trusted source, the individual has passed the verification process and may rent a vehicle. But, if at some point, the government revokes the individual's credential to be eligible to drive, if the user tries to present the revoked claim to the agency, the agency will know that the user's credential has been revoked. This is the major difference from the paper credential model, since there is no possible way to revoke a physical card which is in the possession of an individual.
A diagram highlighting the flow of interactions in the Verifiable Credentials Model is demonstrated in Figure 4.

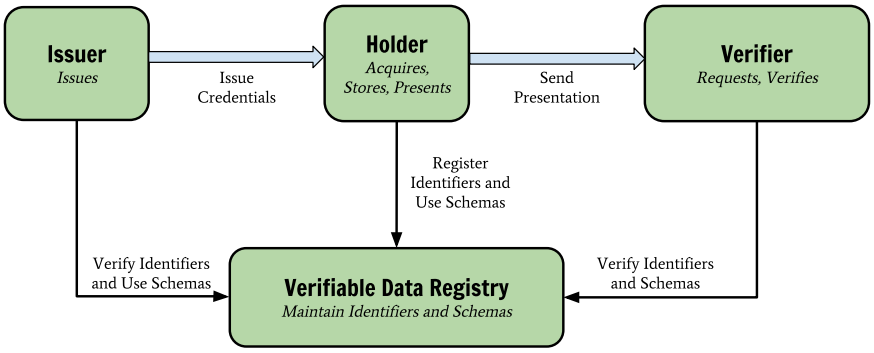

Figure 4. The Verifiable Credentials Model [21]

2.2.3 Decentralized Identifiers

Decentralized Identifiers (DIDs) are the identifiers that power SSI solutions. The aim of this identifier is to provide a scalable way to securely identify each identity that needs to be created. A DID is created for each entity and can be resolved by means of a DID Resolver, which can be seen as the DNS of SSI. Whenever a DID is resolved, it will unveil a DID Document (DIDDoc), which will hold a public key and an endpoint for the other entities to send messages to, allowing them to request for connection establishment [19]. An architectural overview of this process can be found in Figure 5. There are two forms of DID, public or pairwise DIDs. While the first is meant to be used by entities that are meant to be "public" and widely contacted (for example a government branch), the latter provides a more secure channel for two entities to connect between each other. The difference between the two in regards to implementation is that Public DIDs are stored in a verifiable public data registry (the blockchain), allowing any entity to have access to this DID. Any individual can request to connect to the entity whose DID is public by creating a connection request. Pairwise DIDs are meant for more secure communication, since for each connection between two different entities, the data for these communications is stored locally, safeguarding from any other entity from eavesdropping, while also reducing the time to start of a connection [5] [23]. One of the strongest points with regards to DIDs are that at any time, a DID can be revoked

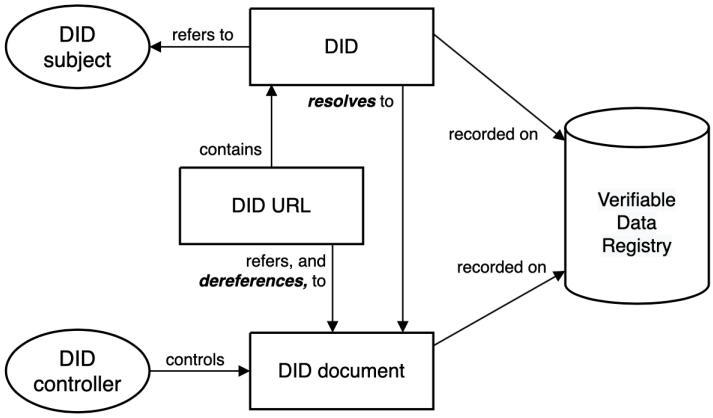

Figure 5. Overview of the DID architecture [25]

by one of the parties, not allowing for other entities who know this DID to connect to this entity through it. If they wish to do so they must request a new DID for a new relationship to be established [24].
A DID is similar to a universally unique identifier (uuid) for one's identity. Depending on the method used to create the DID, the structure will vary[14]. An example of a DID from the Sovrin network that is going to be discussed further in Section 2.4.1 can be found below:

```
did:sov:AKRMugEbG3ez24K2xnqqrm
```

The format of the DID follows the standard listed in the official documentation[15] and the W3C specification [26], where it specifies that the substring in between the commas refers to the DID method, which in this case would be of the Sovrin Network DID (*sov*). At the time of writing there are 88 different DID methods listed in the W3C repository[16].

Messaging Protocol (DIDComm) Although there are many different secure communication mechanisms as detailed in the survey conducted by Nguyen et al. in [27], DID Communication (DIDComm)[17] differentiates itself, as it is defined as a protocol whose purpose is to provide a secure, private communication methodology built atop the decentralized design of DIDs[18].
The methodology of DIDComm is based on the usage of the information that is present in the DID Documents to allow for secure, private, decentralized, transport-agnostic, routable, interoperable, extensible, efficient communication between two individuals through their agents [28].
The messaging format of DIDComm as defined by W3C uses JSON Web Messages[19], and depending on the action to perform (issue credentials, request for a credential presentation, etc), the fields will vary. An example of a "Basic Message" structure would contain the following fields:

```
{
    "id": "1234567890",
    "type": "<message-type-uri>",
    "from": "did:example:alice",
    "to": ["did:example:bob"],
    "created_time": 1516269022,
    "expires_time": 1516385931,
    "body": {"messagespecificattribute": "and its value"}
}
```

[14] https://www.w3.org/TR/did-core/#did-syntax
[15] https://github.com/WebOfTrustInfo/rwot3-sf/blob/master/did-spec-wd03.md
[16] https://w3c.github.io/did-spec-registries/#did-methods
[17] https://github.com/hyperledger/aries-rfcs/tree/master/features/0023-did-exchange
[18] https://identity.foundation/didcomm-messaging/spec/
[19] https://identity.foundation/didcomm-messaging/spec/#:~:text=DIDComm%20plaintext%20messages%20are%20based,message%2C%20and%20are%20called%20headers.

2.2.4 Verifiable Credentials

Verifiable Credentials or Claims (VCs) are the pieces of technology in the digital world that mimic the physical credentials people bear with them on a regular basis like their passport, bank account number, amongst others. Verifiable Credentials are powered by digital signatures (powered by cryptography) making them verifiable and tamper-proof. The format of the VCs follows the standard listed by the W3C specification [29].
The way VCs can be adopted into people's lifes is by following the Verifiable Credentials Model previously mentioned in Section 2.2.2. An example of how the fields (attributes) within the standard paper credentials can be ported to a Verifiable Credential is demonstrated in Figure 6, where the Issuer Signature is missing since it refers to (in the example) the California Department of Motor Vehicles. In order to issue a credential, an issuer needs to use a credential schema that

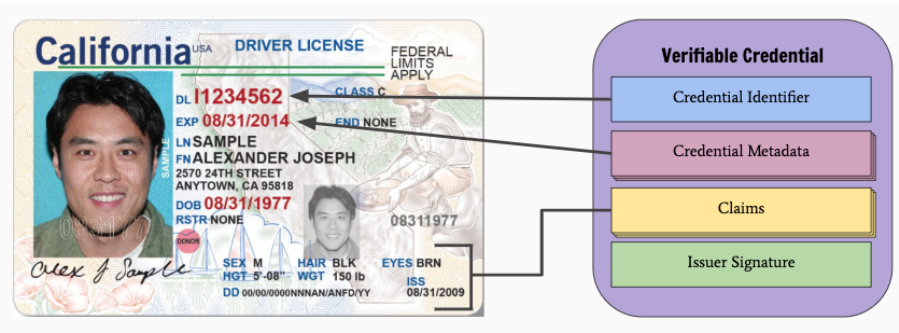

Figure 6. Example of how attributes from a paper crendential can be stored in a Verifiable Credential [20]

is present on the ledger, containing details on which information composes a credential under that schema. Whenever an issuer creates a schema, the ledger will create a response containing an identifier for that particular schema. If an issuer wishes to start issuing credentials under the schema listed above, they need to submit a credential definition onto the ledger mentioning the ID of the schema (provided earlier) and other fields like type of signature method, how to handle revocation and so on.[21]
Figure 7 holds an example of the link between a credential schema (which in the example is issued by the government), and the credential definition (who is created by an entity named "Faber"). Here is it possible to see how the credential definition contains the information regarding the schema as one of its fields.

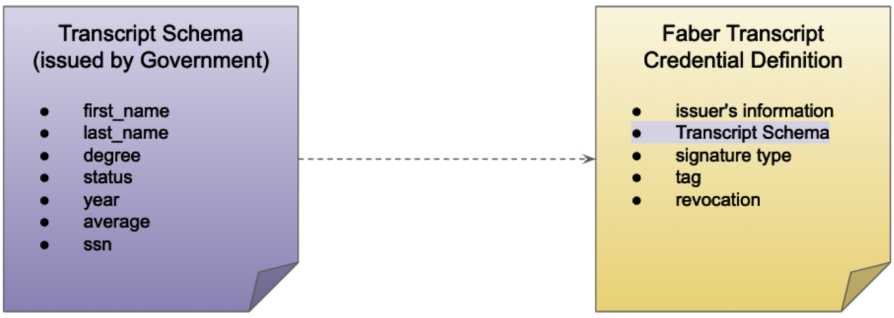

Figure 7. Relation between credential schema and credential definition [22]

Below an example of a credential object is presented, in the JSON Web Message format, containing the attributes, the credential_definition_id, revocation_id and schema_id.

```
{
  "attrs": {
    "additionalProp1": "string",
    "additionalProp2": "string",
    "additionalProp3": "string"
```

[20] https://freecontent.manning.com/the-basic-building-blocks-of-ssi
[21] https://hyperledger-indy.readthedocs.io/projects/sdk/en/latest/docs/how-tos/save-schema-and-cred-def/README.html
[22] https://kctheservant.medium.com/exploring-hyperledger-indy-through-indy-dev-example-10075d2547ae

```
    },
    "cred_def_id": "WgWxqztrNooG92RXvxSTWv:3:CL:20:tag",
    "cred_rev_id": "12345",
    "referent": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "rev_reg_id": "WgWxqztrNooG92RXvxSTWv:4:WgWxqztrNooG92RXvxSTWv:3:CL:20:tag:CL_ACCUM:0",
    "schema_id": "WgWxqztrNooG92RXvxSTWv:2:schema_name:1.0"
}
```


2.2.5 Zero-Knowledge Proof

The concepts of Zero-Knowledge Proofs is its majority powered by advancements in Mathematics and Cryptography and are one of the best ways to preserve individuals identities whenever they need to present any claim about themselves. The moving force behind ZKP is the fact that an individual must not need to disclose any of its attributes in order to validate a claim. The Verifiable Credentials Model has already linked the implementation of ZKPs together with VCs in order to enable data minimization and privacy enhancement, as provers can present proof with minimal disclosure of sensitive and personal data to the verifiers [5][21]. To enhance privacy and security furthermore, the claims presented by the VC can take the form of a predicate ZKP or selective disclosure ZKP.

Predicates in ZKP Predicates are used in ZKP whenever a holder needs to attest that he/she is in compliance with a requirement linked to one of its attributes without disclosing the data linked to that attribute [9] [30]. One of the predicates which is supported by VCs is the greater or equal to ($>=$). Using this predicate and formulating a real-life scenario would be for when an individual wants to demonstrate that he is over the age of 18 in order to request any alcoholic drink at a bar. Usually this process requires handing in the personal ID, in order for the barman to analyze the date of birth and calculate whether the individual is over the age of 18. With predicates in ZKP, the verifier (barman) would request proof that the individual (holder) is over the age of 18, which in turn would craft a Verifiable Presentation using ZKP, which can (by cryptographic means) validate that the user is older than 18, without ever disclosing the date of birth of the individual.

Selective Disclosure in ZKP Still picking up on the example of the individual who is required to demonstrate he is over the age of 18, despite that being a very ineffective process, it also discloses other information to the barman which are not relevant to that transaction. The ID is a Credential composed of many attributes which have relevance depending on the scenario. Hence, in a Verifiable Credentials Model using Selective Disclosure in ZKPs, a user can craft a presentation to show to the barman only the relevant piece of information (attributes) from a Verifiable Credential [24]. This way the individual chooses to only disclose the date of birth from its ID in the form of a VC, and shows that newly crafted VC to the barman, without ever needing to disclose other pieces of data commonly found in a paper ID [31].

2.2.6 Revocation of VCs

An issuer may want to revoke credentials, for the same reasons why paper credentials are revoked in the Paper Credentials Model: either they expire, or the holder lost privilege to hold them. Using the already mentioned driver's license example, if the driver loses too many points, the driver will be stripped off their driver's license so that they cannot drive anymore. With VCs, this event could be handled by revoking the credential that attests as the driver's license of the driver. In order to revoke a credential, the issuer of said credential writes a revocation registry and adds it to the Verifiable Data Registry. When revoked, the credential is no longer valid, and any verifier will be aware of this revocation by looking into the Verifiable Data Registry and being presented with the revocation registry of that credential. The revocation registry links back to the credential, and allows the issuer to unilaterally revoke credentials, independently of the holders. Depending on the adopted DLT, the implementation for the revocation registry might differ[23].

2.2.7 Management of DIDs and VCs

An important aspect that was not yet discussed is how DIDs and VCs are handled and where they are stored. For this end, digital identity wallets, which are pieces of software purposefully created to store DIDs and VCs were envisioned[24]. On the other hand, in order to be part of the Verifiable Data Model and interact with other entities, there needs to be a piece of software that allows to manage and use the VCs that the entities hold in their wallets. That is the role of an agent, which

[23] https://tykn.tech/identity-revocation-blockchain/#What_is_a_Revocation_Registry
[24] https://freecontent.manning.com/the-basic-building-blocks-of-ssi/#id_ftn4

can be defined as a software used by or acting on behalf of an entity to interact with other agents. Agents require access to a digital wallet in order to perform cryptographic operations on behalf of the entity that they represent[25].
Agents are in charge of sending and receiving messages; encrypting and decrypting information; sign digital information on behalf of the entity; manage information in the digital wallet and backup/restore information [28]. The SSI community has advocated that an agent can also be referred to as a wallet, since most agents offer an integrated wallet as one of its components [26]. This makes the two terms interchangeable in the literature.
Some agents contain the implementation to interact with the Verifiable Data Registry (i.e. Distributed Ledger), to allow for the adoption of the Verifiable Data Model. They can be of three different types:

- Edge agents run at the edge of the network on a local device;
- Cloud agents run remotely on a server or cloud hosting service
- Mobile agents are meant to be hosted on the entities' private device (e.g. smartphones)

The biggest difference worth denoting in the different types of agents is that cloud agents provide a public endpoint for interactions with other agents, with this endpoint being publicly available in the Verifiable Data Registry. Therefore, a person whose privacy is more important might be better using a mobile agent on their device, while an organization might have an agent running on the cloud. For IoT devices for example, it would be interesting to consider an edge agent as the preferred agent type. For the latter, lacking compute/storage resources and/or connectivity, securely communicating with a cloud agent that in turn communicates with the ledger [27] would be a possible alternative to the fact that these agents have no public endpoint.

2.2.8 Relation with Distributed Ledger Technology

After covering DIDs and VCs, it is important to understand what gets stored on the Distributed Ledger. The most relevant note to be made is that no private data is stored on the blockchain, which provides any solution using a DLT with an increased privacy-preserving solution. The blockchain will only hold the Public DIDs, the Schemas, the Credential Definitions and the Revocation Registries.

2.3 Blockchain-based Self-Sovereign Identity with IoT

In this section the related work will be discussed in detail, looking at the state-of-the-art research that has been put together with the aim of exploring Self-Sovereign Identity's integration into IoT environments.
For this purpose, the results from the numerous search queries are illustrated in Table 1. Note that for IEEEExplore and ACM (Digital Library), since these were extracted after Google Scholar, many of the papers listed under Google Scholar were also found repeated under the other two sources. The search terms utilized are listed in Table 2.

Source / Topic	Google Scholar	IEEEExplore	DL	Other[1]
SSI + IoT	[5], [11], [24], [30], [32]	[23]	[33]	[22]
SSI	[13]	-	-	-
IdMSs for IoT	[2], [10], [18], [19], [20], [34]	-	-	-
SSI + Energy	[35], [36], [37]	-	-	-

[1] Supervisor Literature, Back- and For-ward referencing.

Table 1. Summary of Literature

2.3.1 Blockchain as an IdMS

The adoption of blockchain as a Decentralized IdMS has been widely acclaimed as a good alternative to handle access control as an alternative to the previous alternatives, namely ABAC (Attribute-based access control) and RBAC (Role-Based Access Control), since it has been proved that these are inflexible, unscalable and difficult to upgrade [38]. Despite

[25] https://sovrin.org/library/glossary/
[26] https://iiw.idcommons.net/Agent/Wallet%3F_What_is_Agent%3F_What_is_Wallet%3F_Are_They_The_Same%3F
[27] https://github.com/hyperledger/aries-cloudagent-python/blob/main/docs/GettingStartedAriesDev/AriesBigPicture.md

Repository	Search Terms
Google Scholar	"Self Sovereign Identity" & "+"Self+Sovereign+Identity" +"Internet+of+Things" & "Decentralized Identifiers"
IEEEExplore	"Self Sovereign Identity with IoT Blockchain"
ACM	"Self Sovereign Identity Internet of Things Blockchain"

Table 2. Literature Search Terms

blockchain-based IdMSs being coined the best alternative to traditional IdMSs [33], these systems still present other issues compared to conventional IdMSs. Issues such as how to handle digital wallet leakage; identity changes; the underlying infrastructure and the key management of the public and private keys are some of the matters which need to be overcome in order to solidify blockchain as a definite substitute for traditional IdMSs [20].

Blockchain IdMS for IoT In a paper by Salah [39] an analysis was made on different IoT access control approaches and the novel architecture based on blockchain proved to outperform OAuth2, OAuth0, Manual Device Authentication and Blockstack in the selected quality attributes (availability, scalability, decentralization and tamper-Proofness).
Being a distributed, incorruptible and tamper-resistant ledger database, blockchain has the potential to address the critical security issues of IoT, particularly on data integrity and reliability [17].
However, not all types of blockchain are capable of serving IoT devices' needs. This is the case for permissionless blockchains, as they usually utilize proof-of-work consensus algorithms that are computationally expensive and involve high bandwidth overheads and delays. This in turn restricts the number of transactions per second that can occur [2] [40]. Hence, the best alternative is to utilize a permissioned blockchain for a possible IoT IdMS [18].
The main challenges which need to be overcome in order to provide a standardized solution for IoT blockchain-based IdMSs are: computational power, energy demand and storage capabilities of IoT devices, mobility and partition of said devices and lastly the latency and capacity of communication to and from devices [17]. On a survey conducted by Lo et al. [18] more issues have been gathered regarding IoT devices and how these can affect a possible IdMS. Again, matters of resource consumption are identified with a scope on how encryption is made on the sensor data. Single point of failure, lack of standardization on managing IoT devices and interoperability are also matters related to IoT identified by this survey.
Matters of confidentiality, privacy and data integrity are also concerns in an IoT environment, and blockchain presents itself as a candidate for solving these issues. Blockchain grants authenticity, non-repudiation, and integrity by default, and if smart contracts are used to manage authorization and automate transactions, this satisfies the need for confidentiality, privacy and data integrity [16]. Despite the good prospects of blockchain, it still contains some drawbacks, such as ensuring security of computation, since the computational capability of blockchain is still limited; and also, due to the latency caused by some consensus algorithms on public blockchains, a system running with blockchain cannot support real-time monitoring [18].
The existing solutions for IoT blockchain-based IdMSs mostly represent generic solutions. However the domain-specific solutions regard the manufacturing domain, automotive, electric trading, energy grid, eHealth and smart home, according conducted surveys on the matter [18]. One of the earliest architectures presented in 2017 referred to BIFIT, a system proposed for IoT smart homes to autonomously extract appliances signatures and create blockchain-based identity for the appliance owners, making use of an Ethereum smart-contract-based approach [34].

2.3.2 Link between SSI and IoT

Although there are approaches mentioned in literature that relate to the usage of blockchain as an Identity Management System within an IoT environment, for the related work of this project the main scope is how the Self-Sovereign Identity concepts (DIDs, VCs, ZKPs) are put into practice in an IoT use case. Therefore, from the list of papers presented in Table 1 the literature from line "SSI + IoT" will be analyzed in this section. Given that this field is currently maturing there are not many papers which specifically target the field of SSI with IoT, with the ones found reporting novel architectures or solutions, and some choosing to focus on the most urging challenges of integrating SSI with IoT or reflect on the benefits of SSI in detriment to the current implementations. In order to summarize the captured key benefits mentioned in the analyzed papers and also the challenges that need to be overcome in order to align SSI concepts with IoT, two tables are presented for such purposes, Table 3 and Table 4, respectively.
The paper provided by Bartolomeu et al. [30] reflects on the use cases, technologies and challenges of SSI for Industrial IoT. In this paper the authors provide a detailed comparison on the five different frameworks which focus on SSI, namely

Hyperledger Indy, uPort, BlockStack, VeresOne and Jolocom (to be discussed further in the following section). The challenges reported by the paper illustrate the major issues with implementing SSI in IoT (lack of development to date or changing the mindset and standards of developers and people).
The paper by Fedrecheski et al. [24] puts more focus in explaining the concept of SSI and comparing its Digital Identity models (DIDDoc and Verifiable Credentials) with other models such as PGP and X.509, and highlights that SSI has an all-around better structure to account for more scenarios, such as adding Semantic Schemas, ZKP and Selective Disclosure.
The remainder of the papers extracted as part of the related work comprise novel architectures which were developed to tackle specific issues. In the paper by Kulabukhova et al. [32] a small discussion is made on which type of blockchain is the most indicated for the task at hand, presenting the likes of Bitcoin and Ethereum as non-viable options given the transaction costs inherent to those blockchains. On the other hand, this paper presents the IOTA Tangle (also to be discussed in the following section) as a viable option and expands it by demonstrating a possible architecture which could cover the likes of Logistic Chains. The proposed architecture does not comply entirely with the SSI concepts, since they propose that the user's mobile wallet acts as the identity provider. Then, if the user wants to use SSI he would require the usage of his local identity from a centralized IdMS to interact, which defeats the purpose of SSI.
In the paper by Terzi et al. [5] an in-depth analysis is made on how to utilize SSI in order to guarantee that the emission rates produced by Internet of Vehicles (IoV) devices (a sub-type of IoT devices) are verifiable and untampered. From the related work research conducted, this paper is the one which more deeply relates to the task aimed with this project, since it also focuses on IoT directed towards the automotive domain. The team behind the paper integrated Hyperledger Indy to authenticate and authorize entities on a Hyperledger Fabric network. Their approach puts authorities as issuers who have write-access to the ledger, allowing them to register the vehicles and issue credentials. The paper reinforces that the emissions data is sent periodically (every 24h) and are stored in the Hyperledger Fabric blockchain. The paper also illustrates the sequence of interactions between a vehicle and a credential provider in order for the vehicle to hold a Verifiable Credential to attest that the vehicle is compliant with regulations.
In the paper by Gebresilassie et al. [11] more emphasis is put into how to manage the identity within a SSI context. Essentially the paper reflects on the current IdMSs and provides some guidance on the IdMSs that can provide a decentralized approach (powered by SSI). The paper covers the likes of uPort, Hyperledger Indy and a project named ADEPT (which at the time of writing has been deprecated). Then the authors propose a novel approach based on the IOTA Tangle aimed at solving a vehicle rental agency use case, which has not been tested to date.
In the paper by Kortesniemi et al. [22] the main focus is on providing privacy for IoT devices. This paper sheds light on the DIDs covered in Section 2.2.3, demonstrating how it can reduce correlation between parties, which in turn improves the privacy of a system making use of DIDs.

Benefits \ Literature	Bartolomeu et al.	Fedrecheski et al.	Kortesniemi et al.	Gebresilassie et al.
Owner-Centric	X	X		
Privacy-preserving	X	X	X	X
Decentralized	X	X	X	X
End-to-end security		X		
Layered authentication	X	X		
Standardized and open approach		X	X	
JSON-based encoding		X		
Scalability improvements				X

Table 3. Captured Benefits of SSI in the literature

Challenges \ Literature	Bartolomeu et al.	Fedrecheski et al.	Kortesniemi et al.	Lo et al.
Technical[1]	X	X	X	X
Best Practices and Standardization		X		X
Organizational and Applicational		X		X
Software availability	X			

[1] Constrained devices; Asymmetric Cryptography; Communication overhead; DID Resolution.

Table 4. Captured Challenges of incorporating SSI with IoT in the literature

2.4 Existing Technologies

In this section the most prominent technologies for the purposes of SSI with IoT integration will be explained, according to the time of writing. Most of the focus will be put into how these technologies are contributing directly or indirectly to put in practice the concepts of Self-Sovereign Identity and the Verifiable Credentials Model.

In Table 5 it is possible to see the different frameworks that have been detected in the literature, either from actual implementations or from experimenting with said frameworks. In the following sections the most occurring frameworks will be analyzed, to understand whether they are still viable alternatives for SSI implementations.

Frameworks \ Literature	[2]	[5]	[10]	[11]	[17]	[18]	[20]	[23]	[30]	[32]
Hyperledger Indy/Sovrin		X	X	X			X	X	X	X
IOTA Tangle	X			X	X	X				X
uPort				X			X		X	X
BlockStack									X	
Veres One									X	X
Jolocom									X	X
Civic										X
Ontology										X
Remes										X
ADEPT				X	X					
ShoCard							X			
Bitcoin					X	X				
Ethereum					X	X				
Hyperledger Fabric					X	X				
EOSIO					X					
IPFS					X					
UCOT					X					
slock.it					X					
Power Ledger					X					
MultiChain						X				

Table 5. Captured Frameworks in Literature

Looking at the results presented by Table 5, the majority of the analyzed papers grasped Hyperledger Indy (7 out of 10), with the IOTA Tangle also getting some attention (5 out of 10). According to these results, a brief overview will be given on Hyperledger Indy and the IOTA Tangle, with an additional section reserved for the remainder of the frameworks mentioned in the table. This overview will mostly refer to the history of the framework and its current state.

2.4.1 Hyperledger Indy / Sovrin

At the time of writing and according to Table 5, Hyperledger Indy / Sovrin Governance Framework is the most widely used framework for handling Self-Sovereign Identity. To provide some context, the Hyperledger Indy[28] project is part of the Hyperledger ecosystem, hosted by The Linux Foundation, and its main aim is to provide an ecosystem to develop Decentralized Identity solutions. Sovrin is the name of the Foundation which essentially built its core network around the Hyperledger Indy framework. The Sovrin Foundation[29] is an active SSI network which is based around generous partners who willingly manage and take part in an SSI network, enabled by the Indy tools. Given that the Sovrin network is one of the most successful case studies built on the Hyperledger Indy ecosystem, the two terms have become interchangeable when referred in the literature.

From Indy to the Indy, Aries and Ursa Stack To summarize Hyperledger Indy's history, it started as a project which would handle all SSI related matters, from the DLT, the agents and the cryptography. The project has evolved over time since it launched in 2015, and a separation of concerns occurred, which brought to light two projects under the Hyperledger environment:

- Hyperledger Aries[30], which would become a tool kit that included the Agent implementations.
- Hyperledger Ursa[31] was left responsible for all things related to cryptography and is now used in several Hyperledger projects who require cryptographic operations.

The Indy Distributed Ledger is characterized as a public, permissioned ledger and consensus is achieved using the Plenum algorithm [5]. There are multiple deployments of Indy networks, but the most widely know is hosted by the Sovrin Foundation. It consists of a single, global instance of Hyperledger Indy that resembles to the identity equivalent of the Domain Name System (DNS). Hyperledger Indy nodes pool is an interconnected network of nodes that maintain the same state of the ledger by signing every communication using elliptic curve cryptography [28].
It is relevant to mention that the Indy DLT is composed of a DID registry, Claims Schemas and Credential Definitions, the Revocation Registries and at last the Key Management Component.

2.4.2 IOTA Tangle/Identity

The second-most mentioned framework in the gathered literature was the IOTA Tangle. IOTA[32] itself corresponds to a cryptocurrency aimed at micro-payments for connecting human and machine economies and at providing blockchain solutions for IoT networks.
The IOTA Tangle is comparable to the blockchain technology, but with a twist. The Tangle consists of a Directed Acyclic Graph (DAG) structure, instead of Bitcoin's chain structure. Although the structure differs, the Tangle implementation still holds the anti-tampering benefits of blockchain [16] [17]. When created, the Tangle network was designed to provide a platform for trusted communication between individuals, organizations and things. On top of Tangle, IOTA has developed a new solution named IOTA Identity[33] prides on the use of the W3C's Verifiable Data Model standard for a digital identity framework. Their solution is described in their whitepaper, illustrating the concepts for a Self-Sovereign Identity framework (DIDs, VCs,...)[41].
The Tangle is proven to be highly scalable [2] [11] [16] and is a network which requires no miners, eliminating miner fees, one of the possible restrictions of a scalable IoT blockchain [16] [17]. According to the implementation proposed by Luecking et al. the IOTA Tangle and its permissionless nature enables the creation of identities for IoT devices at any time without significant effort [2]. The security of IOTA as been discussed in Gebresilassie et al. and given that it uses Winternitz One-Time signature scheme for its encryption, the Tangle can withstand quantum attacks [11].
One possible drawback is mentioned in Kulabukhova et al. where the authors highlight that given the Tangle's design to use snapshots, these allow full nodes to remove a transaction history that comes before the actual snapshot transaction. Since this is happening a Public Key Registry could be eliminated easily [32].
Regarding IOTA Identity, although the whitepaper does provide an indication on the direction taken by the development team, there is an introduction of the topic and not presented solutions. Also, at the time of writing, the framework is still in Beta Release phase, having most of the core functionalities (credential exchange, revocation) still left not implemented,

[28] https://www.hyperledger.org/use/hyperledger-indy
[29] https://sovrin.org/four-pillars-of-an-ssi-network/
[30] https://www.hyperledger.org/use/aries
[31] https://www.hyperledger.org/use/ursa
[32] https://www.iota.org
[33] https://www.iota.org/solutions/digital-identity

revealing some of the immaturity of the created framework. For these reasons, this framework will not be considered further by this work.

2.4.3 Other Technologies

In this section a few of the remainder technologies will be analyzed. The uPort[34] and Jolocom[35] implementations were also mentioned in the literature quite often and are solutions which are open-source, allowing for a larger community to contribute to its development. One of its drawback is that they are based on the Ethereum blockchain, which endures in high cost-per-transaction and slower transaction-per-second times. On top of that, the information on the credentials are stored in the form of hashes directly on the blockchain, which goes against the core principles of SSI listed in Section 2.2.1, that mentions that Personally Identifiably Information (PII) should be held exclusively by its owner, and not stored in any place besides the owner wallet [11].

[34] https://www.serto.id
[35] https://jolocom.io

3 Case Description and Requirement Elicitation

In this chapter an overview will be given on the Case Study at hand. Alongside this description, light will be shed on the requirements to consider for further implementation and architectures.

3.1 Case Description

As covered in the previous sections of this project, Self-Sovereign Identity for IoT devices is a field which has yet to discover its full potential. Exploring the capabilities of SSI concepts in this field will allow further researchers to understand whether certain limitations of these environments caps the usage of DIDs, VCs and more SSI concepts. A field within the IoT scope which has seen an increase in demand in recent times, is the Electric Vehicles (EVs) interactions. These type of interactions are included in a subset of the IoT devices, namely the IoV domain, which was already mentioned in Section 2.3.2.
The Case Study which will be the aim of this project is how to enable Self-Sovereign Identity within an EV Charging interaction, powered by Electric Vehicle Charging Stations.
For the purpose of extracting information on the current architecture of the system, a domain expert was interviewed to assess the details of the current system, as well as to extract the requirements for a novel approach targeted for the same end. The following description of the system was developed over the span of several weeks and over a number of interactions with the expert, to match the details with the current reality of EV Charging.
A high-level overview of the system can be observed in Figure 8. In this diagram it is possible to see the involved parties as well as the types of communication some parties have between each other in the current system architecture. Alongside the diagram it is possible to see a legend with a description of the different parties.
The main goal of this work is to provide the benefits of Self-Sovereign Identity mentioned in Section 2.3.2 into the Electric Vehicle Charging use case. Multiple parties will need to trust each other in order for an exchange of energy to take place. Making use of the Verifiable Credentials explained in Section 2.2.4 it is possible to guarantee that a vehicle belongs to a given owner, that the vehicle is itself, and several other claims which will be described later in this document.
More clarity will be given on the parties and their roles in the current system, and what types of communication occur in this system.

3.1.1 Users and Information Flows

The diagram presented in Figure 8 displays the parties that are involved in this case study, and how these connect between each other. The system comprises of three distinct flows: the "Service Providers", the "EV Owner and EV Interactions" and at last the "Charging and Billing".

Service Providers The primary party in this flow is an eMobility Service Provider (eMSP), whose main goal is to provide the largest network of charging points available to its clients, the Electric Vehicle Owners (EV Owners). If an EV Owner wishes to charge their EV, there is the need to sign a contract with an eMSP (Thin Yellow Line).
In order to expand the network, the eMSP company may hire subsidiaries whose role is to act as a Charging Point Operator (CPO). These CPOs may have a Peer-to-Peer contract (**Bold Green Line**) or be part of Roaming Hubs of the company (**Bold Purple Line**) and have a Roaming Contract (**Bold Red Line**). The CPOs are the ones that own the Charging Stations (CSs) and are responsible for guaranteeing their good functioning. The CPOs are the ones who negotiate with the Electricity Providers (EPs) directly to obtain electricity for the Charging Stations (**Bold Blue Line**). Additionally, the CPOs are the ones who directly communicate with the Energy Grid if one is to consider bilateral communication to balance the grid (Vehicle-To-Grid) (**Bold Yellow Line**).
In some occasions the eMSPs may have a preferred CPO contract, which in turn provides a cheaper price for EV Owners. In the occasion where the EV Owner tries to charge the vehicle in a different CPO (who also has a contract with the same eMSP), the fee may be larger. This also addresses another important point: when an EV Owner charges the EV at a new CPO, the CPO will not have the user credentials ready for authentication. When this happens the CPO may directly connect to the eMSP looking to authenticate the user; if this fails, the Roaming Hub will act as an intermediary between CPOs and eMSPs, in order to provide an alternative authentication route. Whenever an EV Owner wishes to charge their EV, the Roaming Hub will check whether the user has been authenticated in one of the other CPOs in the Hub. This has an associated cost, since reaching to more parties means taking a longer authentication route.

EV Owner and EV Interactions In order for an EV to be driven by someone, it requires documentation to attest a vehicle's unique identifier, which is named Vehicle Identification Number (VIN) and may be contained in a Vehicle Birth Certificate (VBC), and whose issuer is the respective Original Equipment Manufacturer (OEM) (Thin Blue Lines).

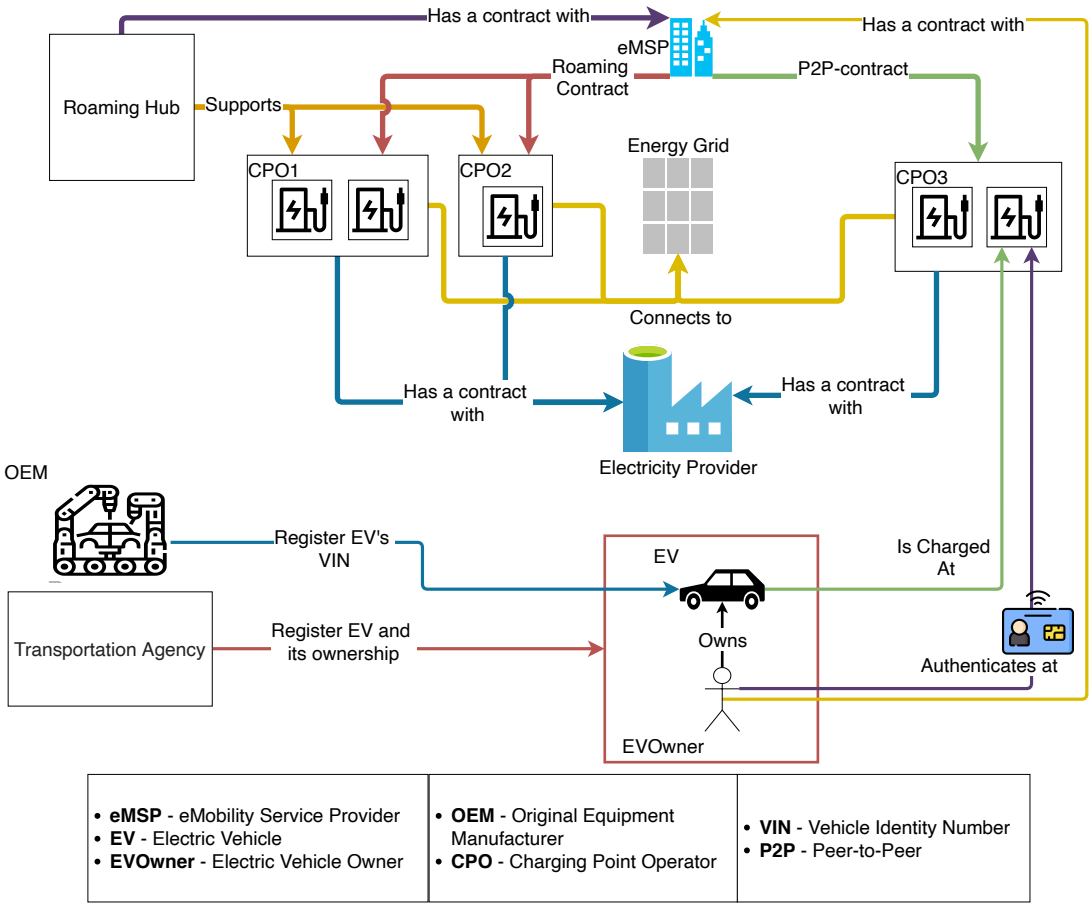

Figure 8. High-level overview of Case Study

For someone to charge their vehicle they must be the owners of said vehicle[36]. This means that the EV Owner and the EV must be registered together in the Transportation Agency (TA) of the respective country (RDW or "Rijksdienst voor het Wegverkeer" in the Netherlands) so that an EV Owner is legally allowed to drive their EV on the roads (Thin Red Lines).

Charging and Billing After the contract with the eMSP is signed (as mentioned in the "Service Providers" flow), the EV Owner may charge the EV at any of the CSs offered by CPOs who are contracted by the same eMSP. This requires the EV Owner to authenticate at the CSs using smart cards or login credentials on a mobile app provided by the eMSP for NFC authentication (Thin Purple Line). Only if the authentication succeeds, the CS will start powering the EV (Thin Green Line). If an EV Owner tries to charge the EV at a CPO that does not have a contract with its eMSP, the charging is not possible.
The current system architecture's billing system comprises of the EV Owner making a monthly payment to the eMSP with the total of the expenses for the charging transactions in that given month. The CPOs obtain their profits from reselling the energy from the Electricity Providers to the EV Owners and also through a fees-model paid by the eMSP which is directly related to the number of EVs charging at the CPOs Charging Stations. The Roaming Hubs profit from providing an authentication route to the eMSPs, in case the CPOs have no past record of the user in the system. The EPs make profit from selling the energy directly to the CPOs with whom they have a contract.

3.1.2 Current Liabilities

In the current implementation of the Electric Vehicle Charging system, there are some liabilities which have been extracted based on the input provided by the domain experts. These include problems of transparency, security, data privacy and functionality.

[36]This is a simplifying assumption for the purposes of this work and in reality this assumption does not always hold (e.g. when lending the vehicle to a relative, or when being the owner of a vehicle rental agency, more on this will be covered in the evaluation chapter)

- The EV Owners are not aware of the price rate of the kWh at the time of charging, and are only aware of the total expense of one transaction when it is described in the monthly invoice. Consequently, CPOs have the chance to boost rates and the price of the energy sold to the EV Owners since the rates are not presented to the EV Owners before charging.
- Problems with current authentication is leading to identity fraud. With the usage of smart cards there is the chance for these to be cloned and used by malicious entities. With the usage of login credentials there is the problem with account hijacking which also leads to identity theft. Reports have been made for multiple charging sessions taking place far apart at close times, under the same identity.
- Still regarding authentication, in the current architecture the CPOs will try to authenticate the EV Owners by going directly to the eMSPs. If something goes wrong in this communication (e.g. fault in the network), and if the CPOs are contained within a Roaming Hub, they have the option to authenticate the EV Owners by means of those Roaming Hubs. When this occurs both CPOs and eMSPs are at a loss, given that the revenue is reduced by the introduction of a new party in this transaction (the Roaming Hub).

3.2 List of requirements

When designing a possible novel architecture, it is important to consider the current limitations in the field of SSI with IoT, and understand which technologies have the power to mitigate or erase such constraints. Also, given the current system architecture for the Charging of EVs, the requirements for that system should also be accounted for in a new architecture. Following the diagram presented in Figure 8 which depicts an overview of the current system architecture, the requirements for a new architecture will be listed, followed by the respective rationales.

3.2.1 Key Drivers

In order to extract the key drivers of this system, an analysis on the literature was made in Section 2.3 to understand which are the most analyzed quality attributes when assessing SSI alone or together with its integration with IoT.

Scalability The literature highlighted as an urgent need that an SSI-based solution for IoT devices has the possibility to scale as the number of connected devices grow [2] [10] [18] [34]. The reason is evident, as mentioned in the introduction, the number of IoT devices is expanding and the need for a scalable IdMS is extremely important.

Performance Another important aspect to consider is the performance of the network. If the network contains low throughput, the number of Transactions Per Second (TPS) may not satisfy the high volume of data collected by IoT devices [18].

Security and Privacy Matters of data privacy, security and tampering are some of other aspects which need to be accounted when dealing with massive quantities of data and the identity of billions of IoT devices [2] [10] [16] [23]. These were matters addressed in literature which focused on an IdMS solution for IoT, which demonstrated to be sufficiently supported by blockchain solutions [18].

Other Relevant Quality Attributes While not key drivers, other quality attributes that were mentioned in the literature should also be taken into account when designing a new system architecture. Interoperability [2] [10] and availability [23] are examples of other NFRs which will also be considered in the selection of the components of the new systems architecture.
Regarding the availability of the system, according to Transport & Environment, Europe's leading clean transport campaign group, a system whose goal is to provide electric vehicle charging stations to clients requires a 97% uptime guarantee, according to the latest published report [42].

3.2.2 Functional Requirements

The Functional Requirements (FRs) displayed in Table 6 relate to the technical functionality of the Electric Vehicle Charging system, and are organized to better display what each functionality will add to the system. The requirements have been categorized using the MoSCoW prioritization technique[37], where each requirement is tagged with one of the following levels of priority: "Must have", "Should have, "Could have" and "Will not have".

[37] https://www.productplan.com/glossary/moscow-prioritization/

ID	Priority	Requirement
FR-1.1	Must	*The system must allow for an EV Owner to charge its EV at a CS*
FR-1.2	Should	*The rate at which the kWh is being sold to the EV Owner should be presented before the charging starts*
FR-1.3	Must	*The EV Owner must be prompted to accept/reject the price of the kWh*
FR-1.4	Could	*At the end of a transaction, the EV Owner could receive proof of the transaction with the amount due*
FR-2.1	Must	*A CS must belong to a CPO*
FR-2.2	Must	*A CPO must have a contract with an eMSP*
FR-2.3	Must	*A CPO must have a contract with an EP to provide power to its Charging Stations*
FR-3.1	Should	*EV Owners should prove they are the owners of their EV to a CS*
FR-3.2	Must	*EV Owners must attest they have a valid contract with an eMSP that the CS's CPO has a contract with*
FR-3.3	Could	*An EV Owner could be made aware that the CS belongs to the CPO it claims*
FR-3.4	Should	*The process of setting up the connection and verifying credentials should be executed with the least EV Owner intervention as possible.*
FR-4.1	Must	*An EV and the information regarding its EV Owner must be registered with the TA in order for the EV to be on the roads and to be charged at a CS*
FR-5.1	Must	*The system must not write Personally Identifiable Information (PII) in a publicly accessible data repository*
FR-5.2	Must	*Every credential issued must be verifiable using cryptographic proof*
FR-6.1	Must	*An EV Owner must be able to delegate (temporary) possession of its EV to another driver*

Table 6. Functional Requirements

3.2.3 Non-Functional Requirements

The Non-Functional Requirements (NFRs) presented in this section are closely related to the key drivers identified in Section 3.2.1. The key drivers act as important guidelines in the development of the system. The system-to-be aims to achieve the desired key attributes by fulfilling the non-functional requirements mentioned in Table 7. The requirements have been categorized using the MoSCoW prioritization technique [38], where each requirement is tagged with one of the following levels of priority: "Must have", "Should have, "Could have" and "Will not have". Each set of requirements directly links to a key driver according to the following match:

- **NFR-1.X** - Scalability
- **NFR-2.X** - Performance
- **NFR-3.X** - Security and Privacy
- **NFR-4.X** - Other Relevant Quality Attributes

ID	Priority	Requirement
NFR-1.1	Must	*The system must scale according to the growth of the market. Supposing an adoption in the Dutch Market (300k EVs in 2020 [39]) , the system must be able to support this number of vehicles and the estimated growth (3M EVs by 2030 [40])*

[38] https://www.productplan.com/glossary/moscow-prioritization/

[39] https://www.rvo.nl/sites/default/files/2020/11/Statistics%20Electric%20Vehicles%20and%20Charging%20in%20The%20Netherlands%20up%20to%20and%20including%20October%202020%20-%202.pdf

[40] nklnederland.com/news/sparkcity-3-million-evs-by-2030-2/

NFR-2.1	Should	*The system should be able to kickstart the charging process in a reasonable amount of time (no longer than the time required without VCs, less than 1 minute)*
NFR-2.2	Must	*The system must withstand multiple users accessing the system at the same time, with no loss of information*
NFR-2.3	Must	*The user must be presented with information on how to engage with the CS in less than 5 seconds*
NFR-2.4	Should	*The process of validation the client's credentials should not take longer than the original method (designed for 2 seconds, with the worst case scenario being 30 seconds)*
NFR-3.1	Should	*The system should be constantly updated to mitigate the risk of quantum-attacks, by employing state-of-the art cryptography*
NFR-3.2	Must	*The system must comply with Dutch regulations, including General Data Protection Regulation (GDPR)*
NFR-3.3	Must	*The system must be protected against forgery of identity, so that in the case of theft of identity the malicious party cannot use the system*
NFR-4.1	Must	*A CS must have an active internet connection to authenticate the EV Owner*
NFR-4.2	Must	*The systems implementation must allow for as many agent implementations as possible, opting for solutions that favor interoperability*
NFR-4.3	Must	*The system must offer a 97% uptime guarantee [42]*

Table 7. Non-Functional Requirements

4 System Design

In this chapter a novel system design is proposed to tackle the requirements highlighted in the previous chapter and bring SSI concepts to this use case. This will start with a decision on the technologies to use for the different components of the system, followed by a presentation of a generic SSI with IoT system topology. After this, the Verifiable Electric Vehicle Charging system's architecture with SSI will be described.

4.1 Technological Decisions

In this section the decisions and rationales for the technologies to use for this project will be presented. In Table 8 a summary is provided for the decisions taken for each aspect of the technology stack. In the sections to follow the rationale for these decisions is presented, mentioning relevant literature and the prime aspects considered when assessing the Distributed Ledger to use and consequently which network to select, and the best solutions available regarding agent implementations.

Technology	Decision
Distributed Ledger	*Hyperledger Indy*
Network Selection	*BCovrin Test Indy Network*
Agent	*Hyperledger Aries Cloud Agent - Python (ACA-Py)*
Mobile Agent	*Trinsic Wallet*

Table 8. List of decisions on technologies to be adopted for the proposed solution

4.1.1 Distributed Ledger

In this chapter the motivation for the selected framework will be made, picking up on the foundations laid by Table 5, where an overview of the existing SSI frameworks was presented according to the literature. By following the discussion in Section 2.4, the decision on which framework to use was narrowed down to Hyperledger Indy or the IOTA Tangle. Most of the quality attributes listed in Section 3.2.1 are satisfied by both Hyperledger Indy[41] and IOTA[42], so it is relevant to assess where these diverge to make a decision on which framework to utilize.

Considerations \ Frameworks	Hyperledger Indy	IOTA Tangle
Active Development	+	+
Decentralized IdMS	+	+
Permissioned	+	-
Transaction Costs	+	=
Open-Source	+	+
Support for SSI	+	+
Easy-to-find Resources	+	-
Open-Source Agents	+	=
Easy-to-Deploy	+	=
Standardization	+	=

The "+", "=", "-" icons demonstrate that the framework is a good, neutral (not enough information) or bad solution, respectively, regarding a given aspect.

Table 9. Pros and Cons of Hyperledger Indy compared to IOTA Tangle

Looking at the aspects described in Table 9, it is possible to see that Indy is favorable in the aspects considered in that table.

[41] https://github.com/hyperledger/indy-sdk
[42] https://github.com/iotaledger/identity.rs

One aspect which was discussed in Section 2.4 is that in order to provide an SSI-based IoT solution, the selected framework used and its underlying DLT implementation need to be analyzed with respect to the access protocol used (permissioned or permissionless) [2]. When analyzing the literature one of the main reasons why the adoption of Indy was made at the detriment of IOTA [10] was due to the permissioned nature of the first, which cuts down any transaction costs [18].

Decision Indy is based on the Hyperledger ecosystem, which offers more standardization compared to the rest of the frameworks available. Aside from that, Indy provides one aspect which others do not: the fact that no public data is written on the ledger, which strengthens the security and privacy of the system. At last, given that Indy has a continuous support and has a very active community at the time of writing, and that it offers itself as a solution for the digital identity use case, it will be the framework used in this project.

Choice of Network Whilst deciding on the Hyperledger Indy DLT ensures the technology to use for the blockchain component of this system, there are different networks to use, and depending on the requirements of the system, a choice needs to be made.
Deploying a network of Indy nodes manually is a laborious task that suffers from lack of documentation. As an alternative to this, a project called Verifiable Organizations Network (VON)[43], contributed by the Province of British Columbia already exists. The latter allows to mitigate the aforementioned issue by not only providing the means to deploy a portable development level Indy Node network, but also the maintainers have already-deployed networks which are possible to use. The rationale for this project mentions that *"The Verifiable Organizations Network (VON) is a community effort to establish a better way to find, issue, store and share trustworthy data about organizations—locally and around the globe. Community partners are using jointly developed software components to enable the digitization of government-issued public credentials—registrations, permits, and licenses. Currently, VON components are based on Hyperledger Indy distributed ledger technology"*[43].
The decision to use a network provided by this project aligns with the needs of the proposed solution, since its components are based on the Indy DLT as mentioned before.
Therefore, there are three alternatives for using this projects networks:

- Local Deployment of the VON Network[44]
- Use the already-deployed BCovrin Test Indy Network (Ideal for Proof-of-Concepts)[45]
- Use the already-deployed BCovrin Prod Indy Network (Ideal for stable production-level solutions)[46]

At an early stage of the development it is possible to use a local deployment of the VON network, since it is good for making tests on connecting agents and issuing credentials. The reason for this is that if an error is committed (wrong transaction is put in the ledger for example), the network can be brought down and restarted with ease.

Decision Given that this study ultimately aims at developing a proof-of-concept as the means of evaluating the proposed system architecture, the BCovrin Test Network would be ideal. If this project reaches a production state, it is possible to move to the BCovrin Prod Indy Network.

4.1.2 Agent Technology

Following the description of what agents are in Section 2.2.7, a decision on which agent to use and its implementation needs to be made. The decision will mostly be between using an open-source agent built using the Hyperledger Aries framework or creating an agent from scratch using existing libraries and resources available.
At the time of writing, the most widely used agents are created under the Hyperledger Aries framework concepts. The choice for which agents to use needs alignment with the adopted DLT, which in the case of this system is Hyperledger Indy. The goal for Hyperledger Aries agents is to become blockchain-agnostic. This means that the aim is that for SSI-related implementations the usage of these agents are not restricted to Hyperledger Indy's blockchain. However, at the time of writing, there is only support for Hyperledger Indy, which complies with the aforementioned decisions regarding the DLT.

[43] https://vonx.io/
[44] https://github.com/bcgov/von-network
[45] http://test.bcovrin.vonx.io/
[46] http://prod.bcovrin.vonx.io/

The Hyperledger Aries agents are essentially a piece of software that allow for any entity to assume any role in the Verifiable Credentials Model (Section 2.2.2) and allows the entities to interact with each other, whilst also creating an environment capable of handling verifiable credential exchange.
The core components of an Aries agent are demonstrated in Figure 9. Here it is possible to see four different components: the Key Management Service, which will handle the private and public keys; the Agent Messaging Interface, which will handle communication with other agents; the Ledger Interface, to directly communicate with the ledger; and at last the controller, which can be programmed according to the given use case.

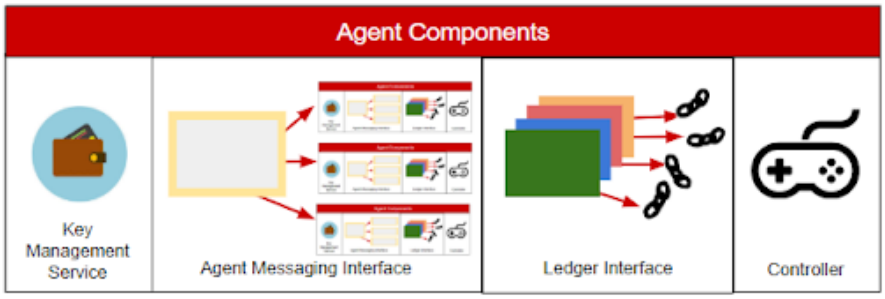

Figure 9. Aries Agent Architecture[47]

For this project the agents that will be used are the Aries Cloud Agent - Python (ACA-Py) implementation. According to the development team, "*ACA-Py is built on the Aries concepts and features that make up Aries Interop Profile (AIP) 1.0, as well as many of the protocols that will be in AIP 2.0. ACA-Py's supported Aries protocols include, most importantly, protocols for issuing, verifying, and holding VCs that work with a Hyperledger Indy distributed ledger and the Indy "AnonCreds" credential format. Contributors are actively working on adding support for other ToIP Layer 1 DID Utilities, and for other VC formats.*"[48]. At the time of writing, the project has a very active community with constant updates and implementation of Aries Request For Comments (RFCs) to accommodate to the standards adopted for SSI (whether Hyperledger standards or ones by W3C).
The ACA-Py agents expose a variety of REpresentational State Transfer (REST) endpoints, that when hit, will spark an action that can range from connecting to another agent, issue a credential, revoke a credential, and many others. These are all documented when deploying an ACA-Py agent, presented using SwaggerUI[49], allowing for an easy way to interact with the agents and be mindful of the necessary parameters for specific actions (for example a credential object needs to be attached to the specific POST request for issuing a credential).
An important aspect to consider when thinking of selecting an agent implementation is whether it supports mediation or not. In the context of agents, mediation is needed in two scenarios: when agents are not guaranteed to be online 100% of the time, or when the agents do not want to have their Public DIDs written to the ledger (for privacy reasons), which leaves them without an endpoint for other agents to connect to. In those events, there should be a mediator agent acting similar to an inbox for e-mail, that stores incoming messages and sends them as soon as the offline agent is back online, addressing the first scenario. When the mediated agent connects to other agents via a pairwise DID communication, the endpoint of the mediated agent will appear to the agent it connects to as the one from its mediator, resolving the second scenario. Mediation is facilitated within the Hyperledger ACA-Py agent, making it a strong agent implementation regarding this aspect. The mediator is never aware of the final recipient of the messages, nor about the content of the message, since onion routing is baked-in the current implementation of the agents, allowing the solution to benefit from higher privacy measures.

Decision Given the fact that the ACA-Py agent is at a stage where it can be used for the majority of actions regarding SSI (establish connections, issue and handle credentials, revocation, mediation, etc), is open-source and has a Ledger Interface compatible with Hyperledger Indy (the Distributed Ledger adopted as highlighted in Section 4.1.1), it will be the agent implementation used in the solution.

[47] https://courses.edx.org/assets/courseware/v1/acc3247dbf74b710586a75f0854699ab/asset-v1:LinuxFoundationX+LFS172x+3T2019+type\spacefactor\@m{}asset+block/Agent_Components.png
[48] https://github.com/hyperledger/aries-cloudagent-python
[49] Software used to visualize and interact with the API's resources - https://swagger.io/tools/swagger-ui/

4.1.3 Mobile Agent

At the time of writing there are several implementations for mobile agents (also mentioned as wallets), which are meant to be used by the individuals on their mobile devices. These offer an already defined logic for controlling the actions regarding handling connections to other agents and how to use verifiable credentials to verify proof requests. One limitation regarding mobile agents is that they do not have a specific endpoint for other agents to connect to. Instead they require an agent (which can be running in the cloud) to act as its mediator. Mediation is a mechanism which will be explained in Section 4.3.1.
The current agents that support Self-Sovereign Identity, identified from literature and online research are: Lissi[50], eSatus[51], Trinsic[52] and Connect.me[53]. There is also the Aries Mobile Agent Xamarin[54] which is an open-source project hosted under Hyperledger ecosystem.
Although in this case the Xamarin agent would be a solution within the same ecosystem (Hyperledger), it is not yet mature enough for testing, while the Trinsic Wallet agent offers a ready-to-use solution, that has the possibility to connect to any Hyperledger Indy Network. This last statement indicates that this agent implementation complies with the choice for the selected network (BCovrin Test Indy Network), which has been adopted following the reasons mentioned in Section 4.1.1. Additionally, the Trinsic Wallet agent offers under-the-hood support for mediation, satisfying a privacy concern that will be described ahead.

Decision For the reasons mentioned in this section, the Trinsic Wallet App has been adopted as part of the proposed solution for the Mobile Agents' implementation.

4.2 Architecture for SSI with IoT

In this section an overview of a generic solution for a system which uses SSI applied to IoT devices is presented. In Figure 10 a diagram illustrates the solution which is generic and highlights the necessary components for a system which has:

- Enterprise agents (presented in **blue** on the left side as Entity A, B and C);
- Mobile agents (in **red**) and its need for a mediator agent (in **purple**);
- IoT devices which are resource constrained (in need of a static agent);
- IoT devices that are not resource constrained.

In a general solution designed for SSI with IoT, the necessary components are:

- A Distributed Ledger to store the DID registry, schemas, credential definitions and revocation registries, as mentioned in Section 2.2.8;
- Agents running on behalf of each entity with pre-configured logic on how to interact between each other;
- For the logic of which actions are to be performed by the agents it is necessary to have a controller which will vary depending on the given use case;
- It is also possible that the agents have a webpage running in front of the controller, which provides a User Interface (UI) to manage interactions with the agent.

In the diagram represented in Figure 10 there are three different types of agents which have already been discussed in Section 2.2.7, 4.1.2 and 4.1.3. There are the Edge and Cloud agents as well as the Mobile agents.
A point worth reflecting on in this section is the fact that some agents should not have a public DID available in the ledger (for example the agents that represent individuals or IoT devices). In these events, the mediator is necessary to provide a routing path between these agents and the outside world. The Mobile Agent will need the mediator agent as mentioned previously, with said mediator running in the cloud. This mediation is also observed in Figure 10 since there are mediators present for the Resource-Constrained IoT devices as well as Private-Non-Resource IoT devices.

[50] https://lissi.id/
[51] https://esatus.com/
[52] https://trinsic.id/
[53] https://www.connect.me/
[54] https://github.com/hyperledger/aries-mobileagent-xamarin

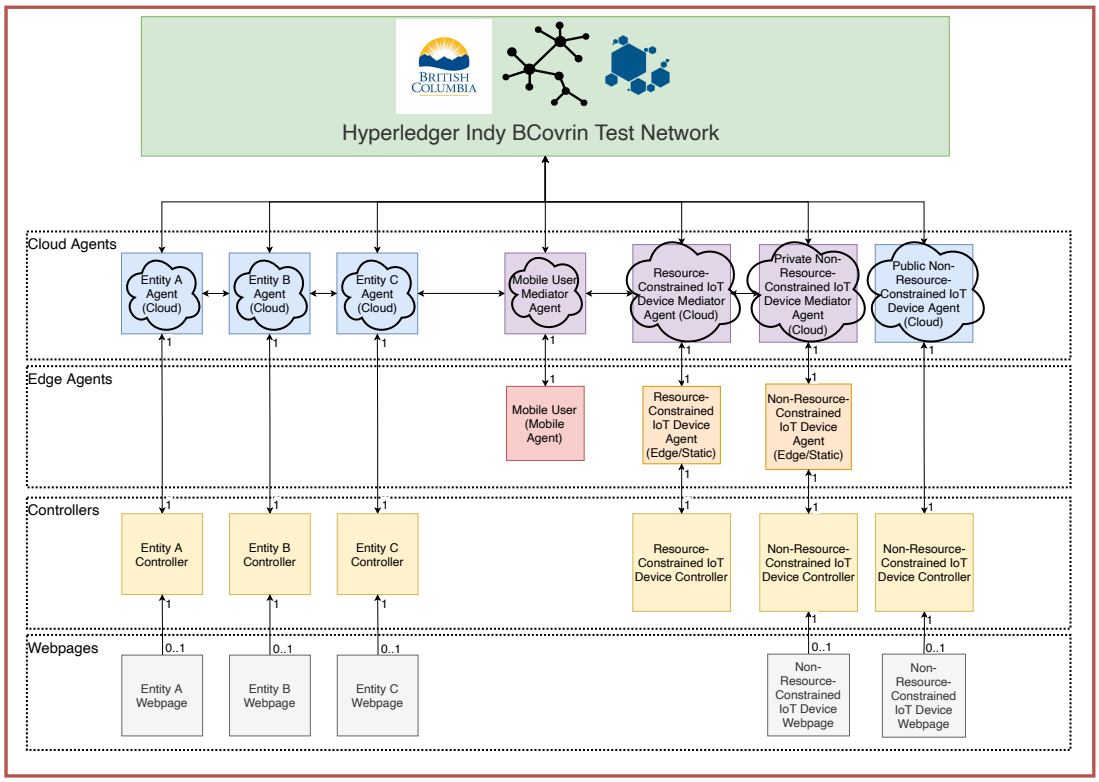

Figure 10. General Architecture for SSI with IoT

4.3 Electric Vehicle Charging System with SSI

For the design of the system to-be, the idea is to port the core concepts of the architecture presented in the general SSI with IoT design in Section 4.2 into the architecture of the Electric Vehicle Charging System. This has to be done while still having in mind the core functionality for authentication of users and vehicles on this system, as presented earlier in Section 3.1.

In this section a run-down of the flow of the system and its components will be presented, with focus on the interactions made between the agents, the different assumptions made, as well as a rationale for the decisions taken that combined into the proposed architecture. This is explained in detail in Section 4.3.1. A detailed analysis on the mapping between real-life credentials and the Verifiable Credentials in the proposed architecture is made in Section 4.3.2. The different flows present in this system and their specific use cases are illustrated in Section 4.3.3. A brief description of how each flow is ported to the new system is made. Additionally, for each use case description the agent interactions are highlighted with the appropriate level of detail, describing each party's role in the communication and the different assumptions made.

4.3.1 Architecture of the System

In Figure 11 a high-level architecture diagram of the Electric Vehicle Charging System with SSI is demonstrated. This diagram highlights the interactions between each party and between the parties and the Hyperledger Indy Network.

Agents Adding Agents to the parties listed in Section 3.1.1, and also including the Indy Distributed Network which implements the DLT are the first steps to migrate to the new architecture. Each party in this diagram represents a group of entities, which will have one agent each. For the sake of simplification, in the diagram presented in Figure 11 only one of each entity is displayed.

All parties except the EV Owner are bearers of an ACA-Py, following the choices made in Section 4.1.2. All of these ACA-Py agents need to be registered to the network (listed in Section 4.1.1) in order to be successfully deployed/started. Although all agents except the EV Owners are running the ACA-Py agent, they are not all configured the same. The EV is considered to be an entity whose identity and details should remain anonymous, similarly to the EV Owner. In these two instances (the EV and EV Owner) no public DIDs should be written to the ledger, in order to preserve privacy. In the case of the EV running the ACA-Py agent, the difference from the other parties is that no Public DID is registered to the network, and that way no other agent on the ledger will know how to connect to this EVs agent.

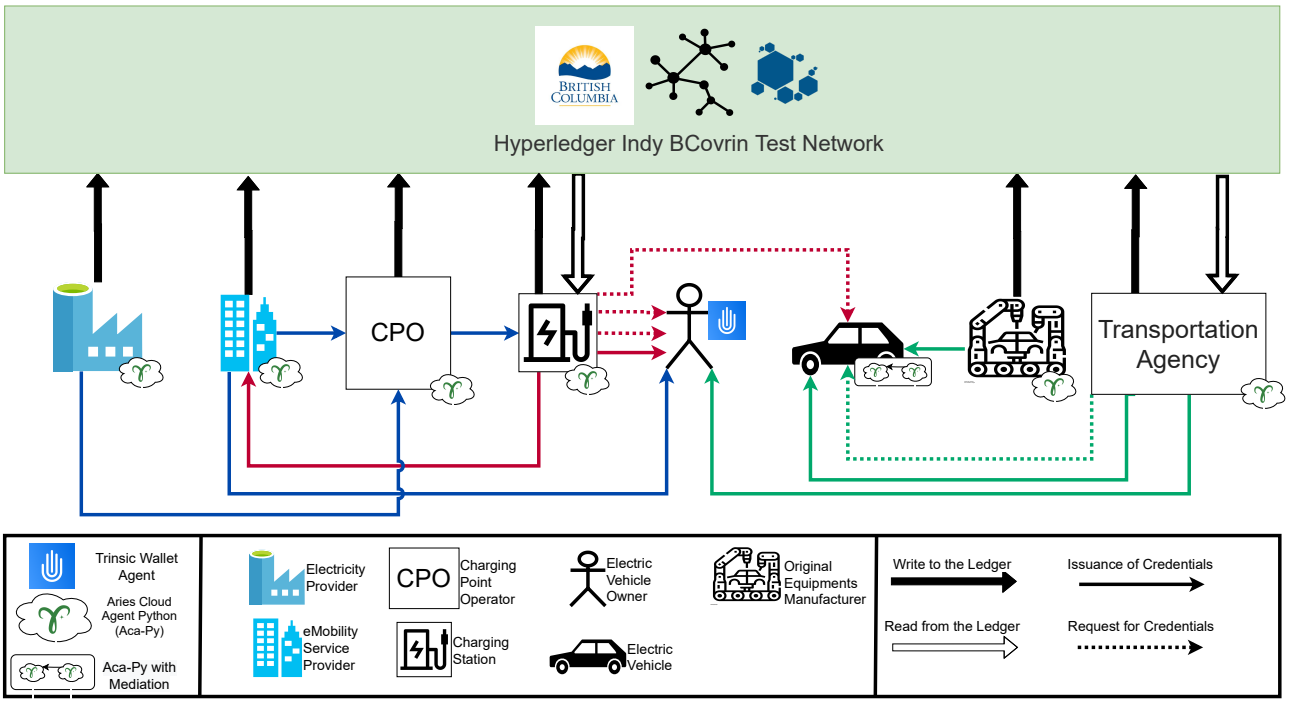

Figure 11. High-Level Architecture Electric Vehicle Charging System with SSI

The EV Owner's agent is meant to be used as a personal device agent, hence the usage of the Trinsic Wallet Mobile App (available on the App Store and Google Play Store). This agent offers an easy-to-deploy scenario for users, without the need for any additional configurations on the developer or end-user end. This agent, as mentioned previously, is allowed to interact with the BCovrin Test Network without a Public DID. This allows for greater privacy since no details are written onto the ledger.
Following the introduction of the ACA-Py agents in Section 4.1.2, and in order to provide a better understanding of the capabilities of the agents and how these can be used independently of the use case, an example of how to use the primary REST endpoints exposed by the agent implementation is presented below. In order for an entity agent to issue a credential to another agent, it is first necessary to create a *credentialJSON* object following the specification provided in the agents documentation, and then perform a POST request to the agents endpoint using the following format:

```
1    curl -XPOST -H "Content-type: application/json" -d '{credentialJSON}' 'issuerAgentEndpoint/issue
     -credential/send'
```

Since the implementation of the proposed solution makes use of the automation offered by the agents (by tweaking the deployment parameters, explained later in the deployment chapter), the flow of issuing a credential from one agent to another is automated after the first POST call is made (marked in the above snippet of the script as 'issuerAgentEndpoint/issue-credential/send'). Therefore, none of the other REST endpoints regarding accepting the credential or storing it in the entities' wallet is needed as this is done automatically by the agents.

Mediation Given that the EV and the EV Owners agents should not have public DIDs and therefore do not have endpoints to connect to, brings the need for mediation in the proposed solution, a term already introduced in Section 4.1.2. For the EV agent mediation, another ACA-Py agent is deployed whose purpose is to act solely as a mediator for one EV agent. This mediation can be established during the deployment of each EV agent. Mediation is already implemented under-the-hood for the EV Owner, running the Trinsic Wallet agent, given that the latter handles mediation internally. This way, the connections established between the EV Owner/EV and their respective mediators, as well as the mediators with all other agents is end-to-end encrypted, preserving the privacy of the EV Owner and EV and allows them to receive messages later, if offline.

Network Roles and fees Given that the proposed solution uses a test network, there are no costs related to performing any actions. But, in a production environment, writing information to the ledger may cost money (for example Sovrin

charges $50 to write a Schema[55] to its production network). In Figure 11 this information regarding who writes to the ledger is marked with the lines going to the Hyperledger Indy BCovrin Test Network, which essentially marks the entities that in a production level network would need to pay fees.

Distributed Ledger The Distributed Ledger is responsible for storing the Public DIDs of issuing or mediator agents, and information regarding the schemas and credential definitions. Additionally it contains information about the revocation registries, so that revocation can be allowed in this system. Every time an agent wishes to issue a new credential type to another agent, it needs to hold a credential definition previously written to the ledger, stating that it can issue credentials to others under a set of attributes defined in a linked schema (this schema may be issued by the same agent or a different one). From that point on, that agent is authorized to issue credentials under that credential definition.

Revocation Registry The revocation registry is used as a means to provide revocation to the credentials. It is not inherently included in the deployed network, but it is a component whose purpose is relevant together with an Hyperledger Indy network, hence this was abstracted in Figure 11 and included it in the "Hyperledger Indy BCovrin Test Network". When a credential definition is created, and when the revocation flag is set to *true*, a revocation registry will automatically be created, and a tails file will be uploaded to the revocation registry (also named as tails server). Revoking credentials is made the same way as issuing a credential, in the way that these actions are handled by the agents by means of REST APIs. When an agent wishes to revoke a credential that was previously issued, it adds a revocation registry entry to the tails server indicating the revocation of a given credential, so that verifiers can look up the credential on the tails server and validate whether the credential is still valid.

4.3.2 Schemas and Credential Definitions

One of the main goals of transposing the architecture of the current paper-based model to the SSI-powered system is to port credentials into VCs. These can be credentials that attest signed contracts, ownership attestations, legal documentation, etc. In this system there are a number of exclusive parties that will be allowed to create schemas and credential definitions, which will be used as the template to issue credentials to the remaining parties. After dissecting the problem and understanding where it is possible to create and adapt VCs, the following mapping listed in Table 10 has been made.

Credential ID	Schema and CD Issuers	Target Holders	Verbose
1	OEM	EV	Credential containing information on the EVs' VIN.
2	TA	EV	Credential attesting the EV is registered and allowed to be driven on the streets.
3	TA	EV Owner	Credential attesting that the EV Owner is allowed to drive the EV they possess.
4	eMSP	EV Owner	Credential representing signed contract between eMSP and EV Owner.
5	eMSP	CPO	Credential representing signed contract between eMSP and CPO.
6	CPO	CS	Credential attesting that a specific CS is owned by the respective CPO.
7	CS	EV Owner & eMSP	Credential containing receipt of charging transaction.
8	EP	CPO	Credential representing signed contract between EP and CPO.

* For the sake of the proposed solution, the issuers of the schemas are the parties who issue said credentials. In a production environment these schemas can be issued by Governmental Parties or Domain-Regulatory Parties, in order to ensure interoperability between all the different vendors.

Table 10. Mapping of Paper-Based Credentials to Verifiable Credentials

The attributes of the most important credentials in Table 10 are listed below. More fields can be added to the credentials, allowing these to be used for other use cases and applications.

[55] https://sovrin.org/issue-credentials/

- **Credential 1**
  - VIN - The VIN number is used by the TA in order to validate that the EV is registered with a valid number by the OEM.
- **Credential 2 and 3**
  - registrationID - This registrationID is the link between the EV and its EVOwner, since both hold the same registrationID.
- **Credential 4**
  - eMSPcontractID - This contractID is used in the charging process as an identifier of the EVOwner to a contract that is valid, provided by the eMSP.
- **Credential 7**
  - kWh - This attribute holds the information regarding the price per kWh at the time of charging.
  - Price - This attribute holds the information regarding the price of the charging transaction.
  - eMSPContractID - This attribute is used to link transactions to a specific eMSPContractID (Credential 4).

4.3.3 Information Flows with SSI

In this section the different flows highlighted in Section 3.1 and how they are handled in the new system will be discussed, marking the similarities between the previous architecture and the proposed architecture with SSI, while also explaining the state of the wallet of the agents after each flow.

Service Providers Flow with SSI In order for the system to be operational for authentication of EV Owners and charging of the EVs, credentials #4, #5, #6 and #8 need to be present upon the systems initial configuration. In the original architecture, an EV Owner needs to hold a contract with an eMSP in order to be allowed to charge the EV, for example. Other details about each credential and their purpose will be explained in-depth with each use case description. During this flow, credentials will only be issued from one party to another without any previous credential verification (at least with the current assumptions made). It is important to mention that the interactions in Figure 11 with the **Blue Lines**, correspond to the information provided in the **Service Providers** flow in Section 3.1.1.
Before making this analysis, there are a number of assumptions that need to be made:
The EVOwner has a contract signed with the respective eMSP and the details have been agreed on by both parties in the physical world.
Table 11 highlights the parties and their respective roles (Issuer, Holder, Verifier and/or Trusted Party) in this particular flow.

Role / Party	Issuer	Holder	Verifier	Trusted Party
eMSP	✓			
Charging Point Operator	✓	✓		
Charging Station		✓		
Electricity Provider	✓			
EV Owner		✓		

Table 11. Roles of the parties involved in the "Service Providers Flow"

In this flow, the steps of the communication are represented in the System Sequence Diagram in Figure 12. Note that given that this flow incorporates four different use cases, that are not dependent on each other, these have been separated both in the figure as well as in the description below, but can be performed in no particular order.

- Use Case "EVOwner/eMSP"
  1. eMSP: The interaction between the eMSP and the EVOwner should begin by the eMSP generating a connection invitation in the form of a QR Code using the HTTP endpoints provided by the agent.

2. EVOwner: The user scans the QR Code using the Trinsic Wallet App, and accepts the connection invitation. After this point the two agents are connected via a pairwise-DID channel.
3. eMSP: The eMSP will then proceed to issue a "Contract between eMSP and EVOwner" (Credential #4) credential to the EVOwner.

- Use Case "eMSP/CPO"

1. eMSP: The eMSP agent will send out a connection request to the CPO agent.
2. CPO: The CPO agent accepts the connection request and after this point the two agents are connected via a pairwise-DID channel.
3. eMSP: The eMSP will issue a "Contract between eMSP and CPO" (Credential #5) to the CPO agent.

- Use Case "EP/CPO"

1. EP: The EP agent will send out a connection request to the CPO agent.
2. CPO: The CPO agent accepts the connection request and after this point the two agents are connected via a pairwise-DID channel.
3. EP: The EP will issue a "Contract between EP and CPO" (Credential #8) to the CPO agent.

- Use Case "CPO/CS"

1. CPO: The CPO agent will send out a connection request to the CS agent.
2. CS: The CS agent accepts the connection request and after this point the two agents are connected via a pairwise-DID channel.
3. CPO: The CPO will issue a "Ownership of CS" (Credential #6) to the CS agent.

During the "Service Providers" flow the mentioned agents should receive the following credentials in their wallets:

- EV Owner Agent Wallet

1. Credential (#4) with Contract information with eMSP

- CPO Agent Wallet

1. Credential (#5) with Contract information with eMSP
2. Credential (#8) with Contract information with EP

- CS Agent Wallet

1. Credential (#6) with Proof of Ownership from CPO

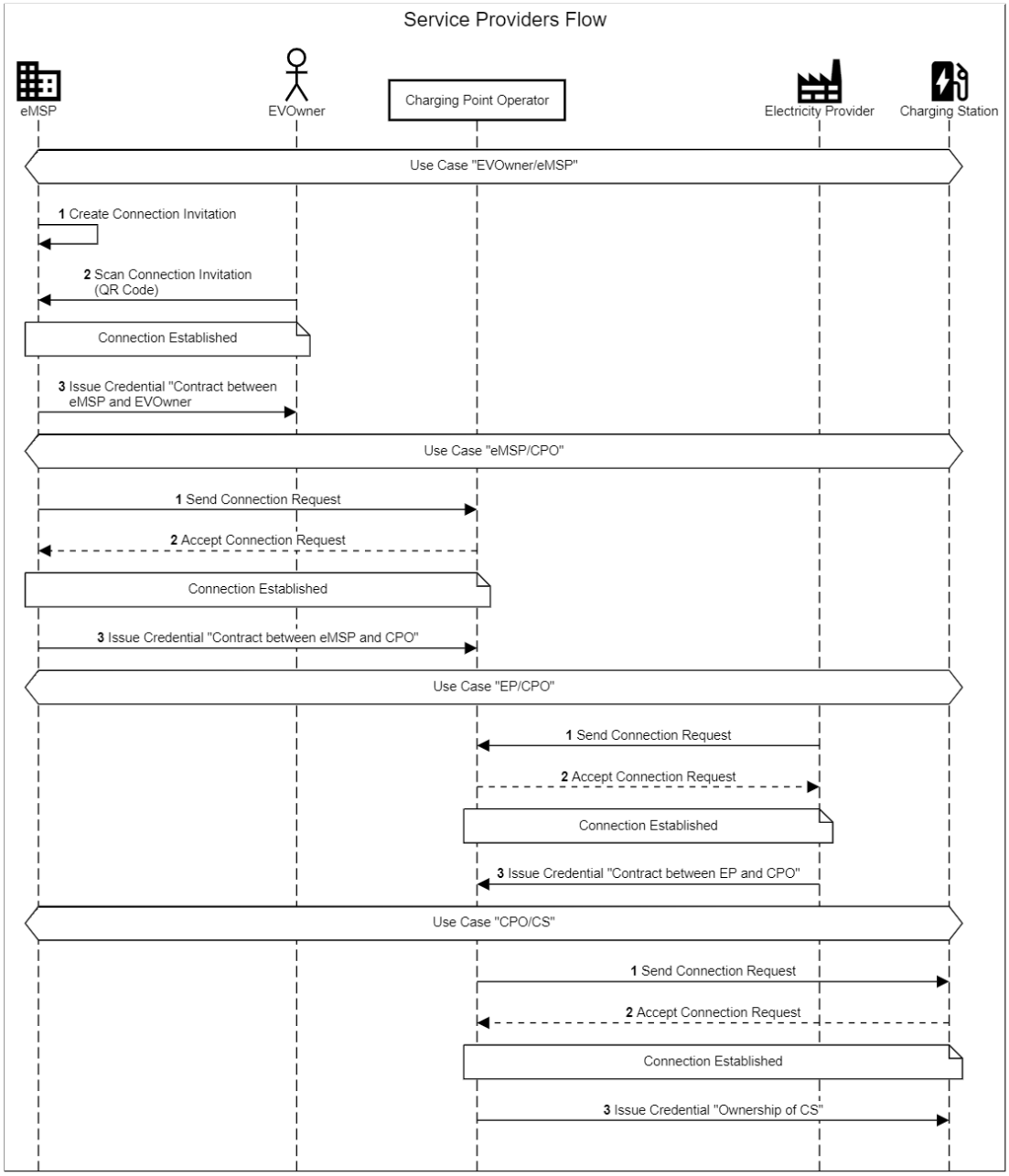

Figure 12. Service Providers Flow System Sequence Diagram

EV Owner and EV Interactions with SSI The **EV Owner and EV Interactions** flow is ported over to the new system with the usage of the interactions marked with the **Green Lines** in Figure 11. This is the first part of the system that requires credential verification. It starts with the EV being issued a VIN by the OEM through means of a credential. After that, the EV is meant to be registered by the local TA (for example the RDW in the Netherlands) and should be given a credential to attest this process. The EV Owner will need to confirm with the TA that it owns the EV it wishes to register; this could be perceived as a credential to add in the future to the system, so in order to simplify this portion of the system an assumption is made that the EV Owner is able to present a credential to this effect. After this, the TA issues a credential to the EV Owner with matching information that can link the EV Owner and its EV (for example the registration ID present in both credentials).

Before making this analysis, there are a number of assumptions that need to be made:

- *The OEM is the party that creates the vehicle, hence it can manually operate and deploy the EVs agent and establish a connection to it.*

- *The EVOwner has a proof of purchase that is sent to the TA which entitles it to register the EV in their name.*
- *The EV Owner and the EV will hold the same registration ID in their credentials given out by the TA, in order to link the two.*

Table 12 highlights the parties and their respective roles (Issuer, Holder, Verifier and/or Trusted Party) in this particular part of the use case.

Role / Party	Issuer	Holder	Verifier	Trusted Party
EV		✓		
EV Owner		✓		
TA	✓		✓	
OEM	✓			✓

Table 12. Roles of the parties involved in the "EV Registration and EV Ownership Registration" use case

In this Information Flow, the steps of the communication are represented in the System Sequence Diagram in Figure 13:

1. EV: The flow commences with the EV sending a connection request (previously generated) to the OEM agent.
2. OEM: The OEM receives the invitation and accepts it, establishing a pairwise-DID connection between the two agents.
3. OEM: The OEM agent now issues a "Vehicle VIN" (Credential #1) credential to the EV agent.
4. EV: The EV agent now wishes to connect to the TA agent, sending a connection request to the latter.
5. TA: The TA agent receives the connection request and accepts it, establishing a pairwise-DID connection between the two agents.
6. TA: The TA agent now requests the "Vehicle VIN" credential from the EV, which was issued previously by the OEM (Credential #1). It requests the VIN attribute and whether the expiration date is greater than the current date.
7. EV: The EV receives the presentation request and generates a proof presentation using the credential issued by the OEM earlier.
8. EV: The EV now sends the proof presentation to the TA agent.
9. TA: The TA agent receives the proof, verifies it and proceeds to send an "EV Registration" (Credential #2) credential to the EV.
10. TA: The interaction between the TA and the EVOwner should begin by the TA generating a connection invitation in the form of a QR Code using the HTTP endpoints provided by the agent.
11. EVOwner: The user scans the QR Code using the Trinsic Wallet App, and accepts the connection invitation. After this point the two agents are connected via a pairwise-DID channel.
12. TA: An event which is not contemplated in the PoC is the agent holding a credential to attest proof of purchase of the EV to register it. But in an ideal scenario the TA agent requests a "Proof of Purchase" credential from the EVOwner.
13. EVOwner: The EVOwner receives the presentation request and generates a proof presentation with the "Proof of Purchase" credential (which is an assumption and is not contemplated in this PoC, as previously stated).
14. EVOwner: The EVOwner now sends the proof presentation to the TA agent.
15. TA: At last, the TA agent issues an "EV Owner Registration" (Credential #3) credential to the EVOwner.

During the "EV Owner and EV Interactions" flow the mentioned agents should receive the following credentials in their wallets:

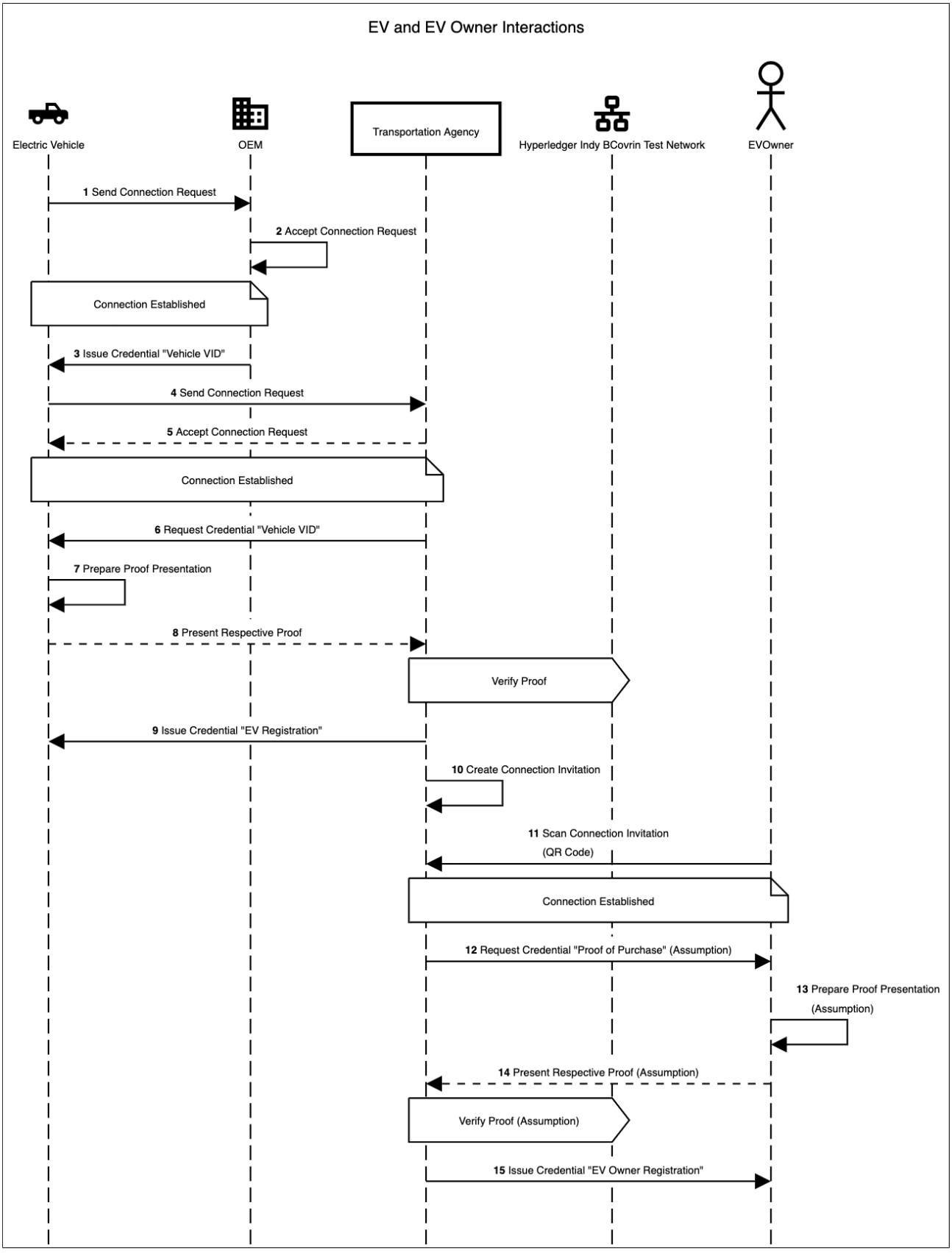

Figure 13. EV and EV Owner Interactions System Sequence Diagram

- EV Agent Wallet
    1. Credential (#1) with Vehicle VIN from OEM
    2. Credential (#2) with EV Registration from TA
- EV Owner Agent Wallet
    1. Credential (#3) with EV Owner Registration from TA

Charging and Billing Flow with SSI The most important part of this system is exactly demonstrating how to improve the privacy of the act of charging the EV; additionally, the system with SSI aims to tackle one of the key issues with the previous architecture: to provide means for the users to know the kWh rate, hence giving the opportunity to deny charging if the rate is too high according to the users desire. Additionally, it is also important to introduce in the system a way for the user to know how much they have spent after each charging session, and keep a record of that. For this, the proposed architecture aims at the end of each transaction to give power to each Charging Station agent to issue a credential to the EV Owner with a "receipt-like" credential, that can be used later to prove claims and also to the eMSP to prove that a specific user has charged its vehicle (seen with the **Red Lines** in Figure 11). In order for EV Owners to know the rate at which the EV will be possibly charged, the Charging Station agent shall communicate with the CPO agent to obtain this information, and pass it on as a DIDComm Basic Message[56].
Before making this analysis, there are a number of assumptions that need to be made:
The connection between the Charging Station and the EV is simplified since the process to send information from an EV to a CS falls out of the scope of this study. The reason is that the domain-specific protocols are not easy to integrate in this solution and need to be better assessed in the future.
Table 13 highlights the parties and their respective roles (Issuer, Holder, Verifier and/or Trusted Party) in this particular part of the use case.

Role / Party	Issuer	Holder	Verifier	Trusted Party
Charging Station	✓		✓	
EVOwner		✓		
Electric Vehicle		✓		
eMSP		✓		✓
Transportation Agency				✓

Table 13. Roles of the parties involved in the "Charging and Billing Flow"

In this use case, the steps of the communication are represented in the System Sequence Diagram in Figure 14.
The steps are explained in detail in the following list:

1. CS: The interaction should begin by the EV Owner arriving at a Charging Station, that will generate a connection invitation in the form of a QR Code using the HTTP endpoints provided by the agent.
2. EVOwner: The EVOwner scans the QR Code using the Trinsic Wallet App, and accepts the connection invitation. After this point the two agents are connected via a pairwise-DID channel.
3. CS: The Charging Station agent will then request the credential attributes from the EVOwner that prove they hold a contract with an eMSP (Credential #4).
4. EVOwner: The EVOwner receives a credential presentation request on the Trinsic Wallet App, and assuming it agrees to disclose the information regarding "Contract ID", "eMSP Company Name" and use ZKP to prove that the "Expiration date of the contract" is greater than the current date, the Trinsic Wallet App will automatically fetch all the relevant attributes from the stored credentials and generate the proof.
5. EVOwner: The Trinsic Wallet App then submits this proof and sends it back to the CS agent, with a reference to the Credential Exchange ID created by the CS agent in Step 3.

[56] https://github.com/hyperledger/aries-rfcs/tree/527849ec3aa2a8fd47a7bb6c57f918ff8bcb5e8c/features/0095-basic-message

6. CS: The CS agent receives the proof and verifies it, checking the revocation status of the presentation.
7. CS: The same process of Steps 3-6 is repeated for the credentials that attest the EV Ownership from the EVOwner, previously issued by the TA (Credential #3).
8. CS: In case the EVOwner has a valid contract with an eMSP, the CS will proceed to contact the CPO agent and request the price per kWh for a user contracted under that eMSP, since this can vary depending the contracts signed between the CPOs and the eMSPs. Otherwise the charging session is rejected.
9. CPO: The CPO agent will receive the message from the CS agent and reply accordingly with a charging rate.
10. CS: The CS will receive the message and present this information to the EVOwner on the CS dashboard.
11. CS: The CS will now connect to the EV in order to compare the credential attributes from the EV Owner and the EV and validate they are indeed linked.
12. EV: The EV agent will accept the connection request from the CS (Note that this process is an assumption and needs to be studied in depth in the future work).
13. CS: The same process of Steps 3-6 is repeated for the credentials that attest the EV Registration coming from the EV, issued by the TA (Credential #2).
14. CS: The values presented by both the EV and EVOwner are compared, and if they match, the charging session starts. Otherwise the charging session is rejected.
15. CS: At the end of the charging session the CS issues a credential to the EVOwner and the respective eMSP with a receipt of the charging session (Credential #7).

During the "Charging and Billing" flow the mentioned agents should receive the following credentials in their wallets:

- EV Owner Agent Wallet
    1. Credential (#7) with Receipt of Charging Session from Charging Station
- eMSP Agent Wallet
    1. Credential (#7) with Receipt of Charging Session from Charging Station

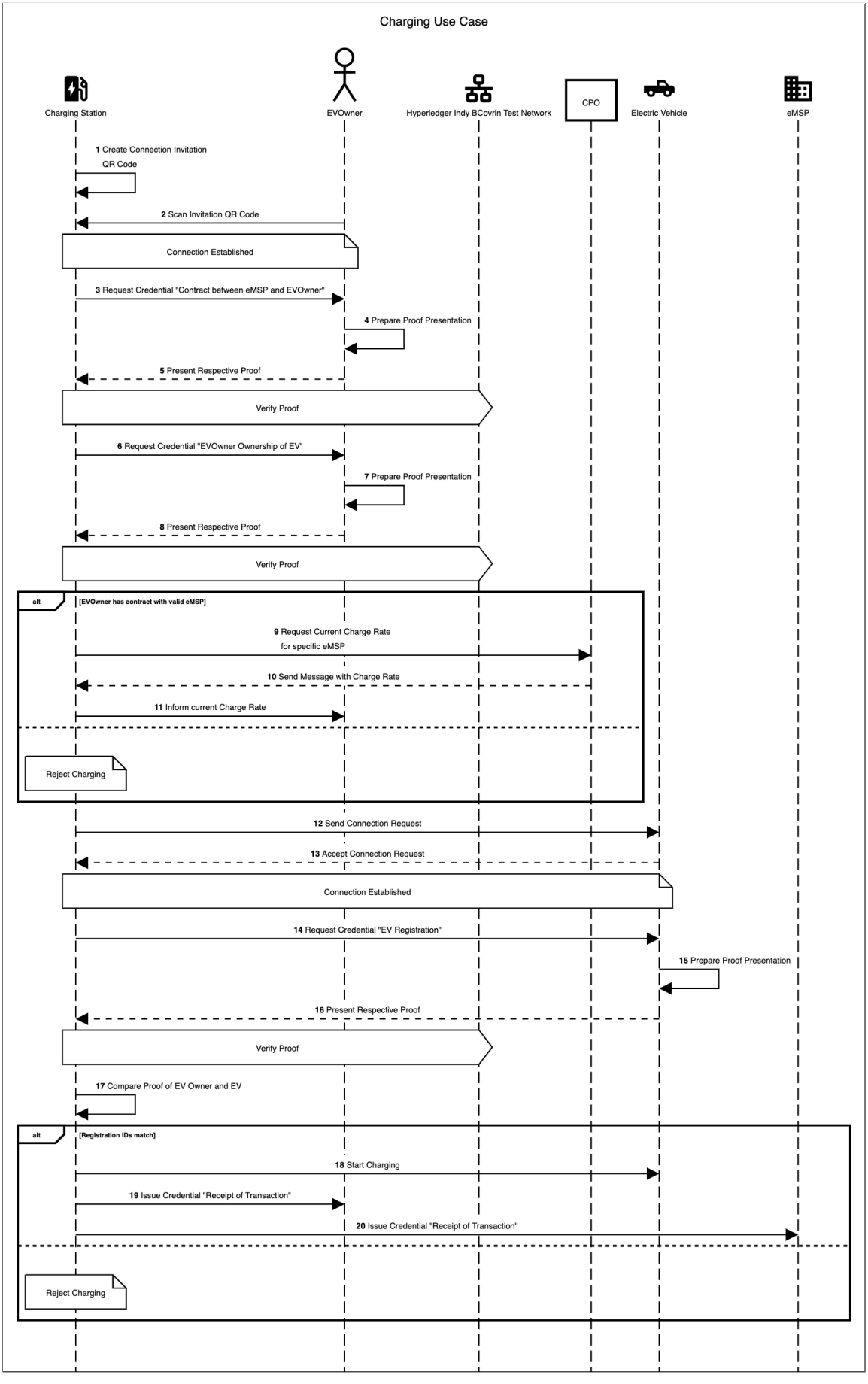

Figure 14. Charging and Billing Use Case System Sequence Diagram

5 Implementation and Deployment

In this chapter the implementation details for a proof-of-concept prototype of the proposed solution, as well as the deployment procedures for it are discussed, with in-depth explanations of the components of the system and the responsibilities of each script. The source code discussed here can be found in a GitHub repository[57] with more comments on the structure of the source code.
A list of the software that was used for this implementation and their respective versions can be seen in Table 14.

Software	Version	Repository
Operating System	macOS Big Sur 11.3	-
Docker Engine	20.10.5	https://docs.docker.com/engine/install/
Tails Server	-	https://github.com/bcgov/indy-tails-server
Aries Cloud Agent - Python	0.6.0	https://github.com/hyperledger/aries-cloudagent-python
Trinsic Wallet Agent	3.2.1	-
pgrok	3.2.0	https://github.com/jerson/pgrok
NestJS	7.5.0	https://github.com/nestjs/nest
AngularJS	12.0.4	https://github.com/angular/angular

Table 14. List of the utilized software, and respective versions and repositories

5.1 Implementation Details

A number of components were needed to facilitate agent deployment, ledger tests and also software deployment. The system follows the structure presented in Section 4.2, where an Agent, a Controller and a Webpage are present for most of the different entities' agent. In Figure 15 the prototypes' architecture is presented, with all the implemented components and how these communicate with each other. The agents can connect to each other through the DIDComm protocol (explained in Section 2.2.3). Whenever a REST endpoint is hit on the NodeJS backend, it triggers a REST call to the agents available endpoints (provided by the ACA-Py implementation) to perform specific actions (start a connection, issue a credential, request credential verification process, etc). The two agents in purple are special since they are acting as mediator agents for both the EVOwner and the EV agents, in red and orange respectively. The EVOwner agent and its mediator are provided by the Trinsic Wallet App, while the EV Mediator agent is an ACA-Py agent that is configured manually to act as a mediator. The Webpages for each of the agents interactions were created using the Angular Framework and provide means contact the backend or the agents directly and have them execute certain actions related to the use case.

5.1.1 Agent Configuration

The parameters used to start-up the agents play a significant role in the necessary tasks needed to be performed by the controller (and in turn the developer). Within the content found on the ACA-Py repository, there are a great number of deployment parameters that can be set on startup. Here the configuration details of the agents are discussed and rationalized. For more information on each of the parameters it is recommended to look at the official startup parameters documentation[58].
One of the most important things to mention in these parameters is the fact that for the sake of simplification and given that this implementation was created with the goal of being a proof-of-concept, multiple "auto-*" parameters have been set to true. This allows the agent to automatically perform certain actions, without the need of a controller. Although this helps demonstrate the functionalities of the agents, in a production level application it would be better to manually implement these actions. In cases of mediation, if this connection is meant to be established on startup (which is the case with this proof-of-concept), then the *open-mediation* and *mediator-invitation* parameters needs to be set on the mediator and the mediated agents, respectively.
It is also relevant to mention that the agents have pre-configured webhook functionalities, that allow for a better interaction with the agents. After assigning a *webhook-url* (which in theory should be assigned to the same endpoint as the controller that is running), it is possible to listen to specific webhook URLs and extract information about the different steps in the

[57] https://github.ibm.com/Filipe-Capela-CIC-Netherlands/SSI-IoT-PoC
[58] https://github.com/hyperledger/aries-cloudagent-python/blob/main/DevReadMe.md#about-aca-py-command-line-parameters

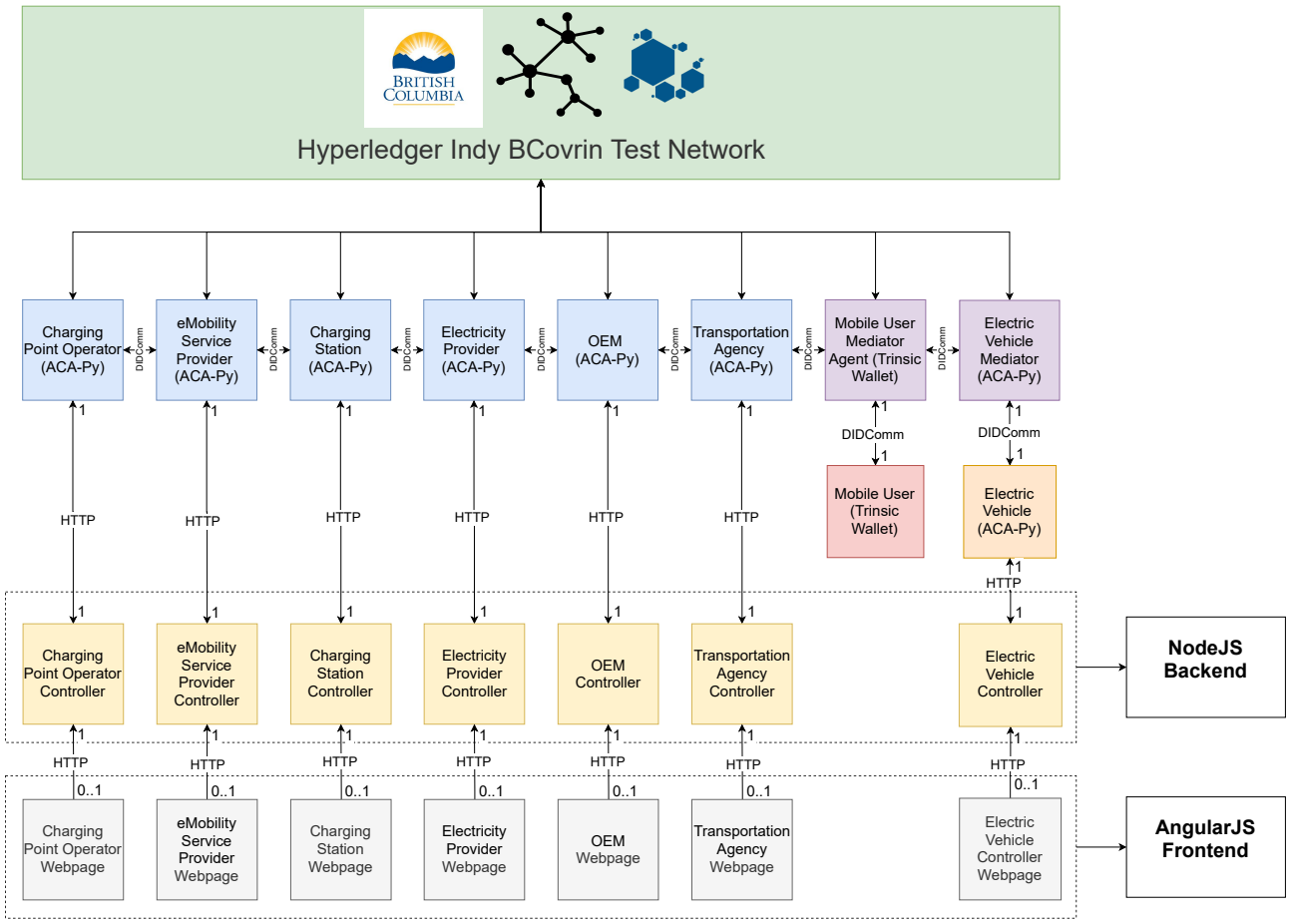

Figure 15. Implementation Components Architecture and Protocols

communications, and from that perform custom logic using the controller. One example of that is the basic messages exchanged between the CS and the CPO agents to obtain the charging rate fee (as depicted in Figure 14 of Section 4.3.3).

5.1.2 Adopted Aries RFCs

All standards used in this solution follow the Aries RFCs Specifications (which have been marked as ACCEPTED at the time of writing) found in the appropriate GitHub repository[59]. These RFCs are considered to be the standard for communication between Aries agents. The most relevant RFCs that explain how the agents can establish connections, send messages, issue credentials, verify credentials, be mediated, are highlighted below with a link to the appropriate RFC:

- **Aries RFC 160: Connection Protocol** - In order for an agent to establish a connection with another, the communication flow follows the Aries RFC 160: Connection Protocol[60].
- **Aries RFC 095: Basic Message** - When an agent sends a message to another agent, it utilizes the Aries RFC 0095 - Basic Message[61].
- **Aries RFC 0036: Issue Credential Protocol 1.0** - The first iteration of the Issue Credential Protocol is used to explain the flow of messages needed to issue credentials from one agent to another. This protocol is still utilized in this proposed solution despite the introduction of version 2.0 of this RFC, since the Trinsic Wallet App only supports versions 1.0 of the protocols mentioned in this section[62].

[59] https://github.com/hyperledger/aries-rfcs/blob/master/index.md
[60] https://github.com/hyperledger/aries-rfcs/blob/master/features/0160-connection-protocol/README.md
[61] https://github.com/hyperledger/aries-rfcs/blob/master/features/0095-basic-message/README.md
[62] https://github.com/hyperledger/aries-rfcs/blob/master/features/0036-issue-credential/README.md

- **Aries RFC 0037: Present Proof Protocol 1.0** - This is the sibling RFC to the RFC 0036, since this utilizes the concepts explained in the latter to explain how a Holder can prove to a Verifier (remembering the roles explained in Section 2.2.2) any claims[63].
- **Aries RFC 0453: Issue Credential Protocol 2.0** - The second iteration of the Issue Credential Protocol is used when an ACA-Py agent wishes to issue credentials to other ACA-Py agents. This version replaces the (indy-specific) filtration criteria with a generalized filter attachment to align with the rest of the messages in the protocol[64].
- **Aries RFC 0454: Present Proof Protocol 2.0** - As a second iteration of the Present Proof Protocol 1.0 protocol, this one is used between presentation requests between ACA-Py agents, since both support the usage of this protocol[65].
- **Aries RFC 0046: Mediators and Relays** - In order to guarantee mediation between agents, used in the Electric Vehicle agent implementation, the communication flow between mediator and mediated agents is explained in detail in RFC 0046: Mediators and Relays[66].
- **Other Relevant RFCs** - Other RFCs are implicitly used when developing using the Aries framework, and are underlying to the agents communications. Therefore they have been abstracted from a detailed explanation but are worth revisiting in the official Aries RFCs repository as linked in the introduction to this section.

5.1.3 Application Logic

Since the application relies on multiple parties, it is important to explain how the flow between the *frontend - backend - agent* works, with the help of some use cases. The agents expose REST API endpoints which are used to trigger operations according to the agents documentation, as explained previously in Section 5.1.1. The controllers are all present in the same NodeJS project, which essentially has different endpoints, depending on the action that is meant to be performed, as in any REST API application. To provide a better understanding of how the application works, a small diagram in Figure 16 contains the call stack between the different components in the use case of the Charging Station issuing a credential to the EVOwner at the end of the charging session.

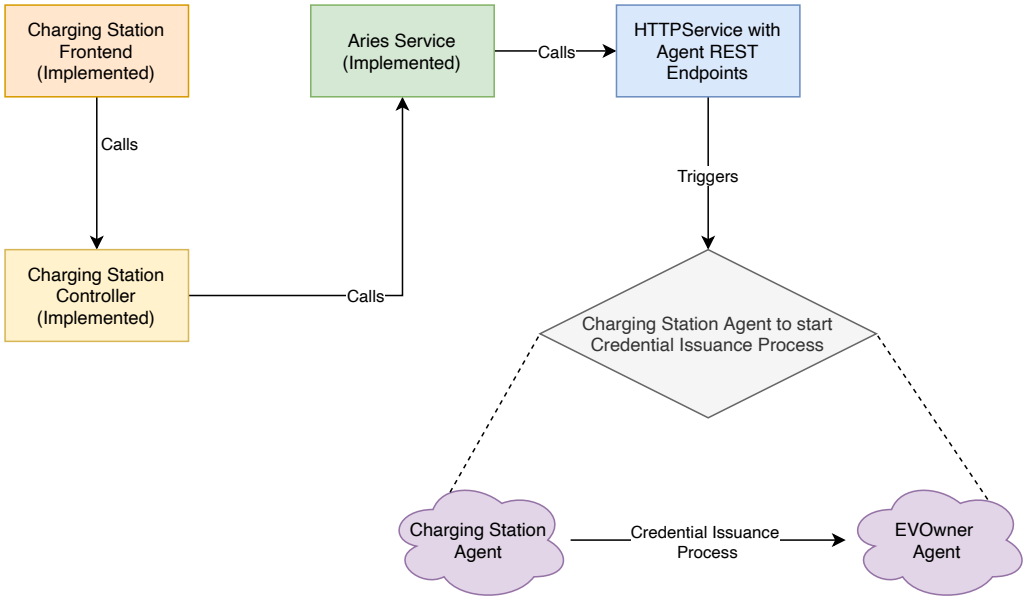

Figure 16. Call stack to Start Credential Issuance Process

Starting off, the user triggers the action to issue the credential by pressing the button on the UI "Issue Receipt Credential". This will trigger the AngularJS function *issueCredential* (as seen below) that will call the Angular Service to hit a NodeJS REST endpoint that is tailored for that event (function *issueCredentialWithFields*).

[63] https://github.com/hyperledger/aries-rfcs/blob/master/features/0037-present-proof/README.md
[64] https://github.com/hyperledger/aries-rfcs/blob/master/features/0453-issue-credential-v2/README.md
[65] https://github.com/hyperledger/aries-rfcs/blob/master/features/0454-present-proof-v2/README.md
[66] https://github.com/hyperledger/aries-rfcs/blob/master/concepts/0046-mediators-and-relays/README.md

```
1  issueCredential(): any {
2    this.coreService.issueCredentialWithFields('cs', this.connectionId, 'receipt', this.
       eMSPContractID);
3  }
```

```
1  public issueCredentialWithFields(endpointName: string, connectionId: string, credentialEndpoint:
       string, contractID: string): any {
2    try {
3      return this.http.post(this.backendUrl + '/' + endpointName + '/issue-credential/' +
     credentialEndpoint, {"connectionId": connectionId, "eMSPContractID": contractID}, this.headers).
     toPromise();
4    } catch (err) {
5      console.log(err);
6    }
7  }
```

The code below is the controller code on the NodeJS backend that is listening to the endpoint hit previously by the frontend call (*issue-credential/receipt*). Lines 4-10 represent the object that contains the attribute values for the receipt credential after a charging session. The object *credentialOffer* in line 14 represents the object that is sent to the issuer agent, with information such as the connectionID (line 16) to the (to-be) holder of said credential. Other relevant information like the schemaID, credential definition ID and issuer DID (lines 21-23) are also present in this object. At last, the object is ready to be sent to the appropriate agent endpoint that handles the issuance of credentials. For this, the controller calls the implemented service (line 27) to perform the HTTP POST request that are used to interact with the agents.

```
1    @Post('issue-credential/receipt')
2    public issueCredentialReceipt(@Body("connectionId") connectionToEVOwner: any, @Body("
     eMSPContractID") eMSPContractID: any): Promise<CredentialExchange> {
3
4        var credentialValues = [
5            "0000000001",
6            "00:30:00",
7            this.currentChargingRate,
8            "4€",
9            eMSPContractID
10       ]
11
12       const credentialPreviewArray = this.credential7schema_attributes.map((key, i) => ({ name:
     key, value: credentialValues[i] }));
13
14       const credentialOffer = new CredentialProposalRequestV1(false,
15           "Charging Transaction Receipt",
16           connectionToEVOwner,
17           new CredentialPreview('issue-credential/1.0/credential-preview', credentialPreviewArray)
     ,
18           {
19               dif: { some_dif_criterion: '' },
20               indy: {
21                   schema_id: this.credential7schema_id,
22                   cred_def_id: this.credential7credentialDefinitionID,
23                   issuer_did: this.csDID
24               }
25           });
26
27       return new Promise(resolve => {
28           this.ariesService.sendCredentialV1(this.api, credentialOffer).pipe()
29               .subscribe(responseSend => {
30                   resolve(responseSend);
31               }
32               )
33       }
34       )
35   }
```

After the service is called, the service is programmed to make use of the agents REST endpoints and have them perform the desired tasks, in the code below it is possible to see the generic implementation of how to have an agent start the issuance of credentials (with the V1 protocol version). After this point, the agent will handle the rest and the credential will be sent to the receiver agent.

```
1         public sendCredentialV1(endpoint: string, credentialProposalRequest:
      CredentialProposalRequestV1): Observable<CredentialExchange> {
2             return this.httpService
3                 .post(endpoint + "issue-credential/send", credentialProposalRequest)
4                 .pipe(map((response: AxiosResponse<CredentialExchange>): CredentialExchange => response.
      data))
5                 .pipe(
6                     catchError(e => {
7                         throw new HttpException(e.response.data, e.response.status);
8                     }));
9         }
```

5.2 Deployment Details

In this section the deployment details of the prototype will be highlighted, with instructions on how to replicate the behaviour of the system and what conditions to bear in mind when deploying the system from fresh.

5.2.1 Steps for the deployment

In order to deploy the system, a number of software/utilities are used. The system comprises of 6 components, which are containerized using Docker[67] and orchestrated using docker-compose[68]. These components are present in Figure 17 and contain the following:

- postgres - a container that is running an instance of a postgres database, needed in case of backend persistent storage (if needed, to keep track of connection IDs, schema IDS and/or credential definition IDs).
- docker - this entry contains two instances that make for one tails-server[69], which is used as the revocation registry for the system, this is necessary otherwise revocation cannot be supported. This software stores and makes available tails files for use with Hyperledger Indy.
- aries-cloudagent-containers - this is an entry that comprises all eight agents that were created for this proof-of-concept, each agent is therefore a docker container.
- ssi-thesis-backend - a container that is running a NodeJS backend with the NestJS[70] framework, for the backend of the proof-of-concept.
- ssi-thesis-frontend - four containers that are running multiple instances of AngularJS[71] frontends, one for the main application, and three used to have a better interface of three of the agents states (connections, credentials, etc...).

In order to deploy these containers, the instructions are put together in different bash scripts[72] to automate the deployment of the application.
Given that the agents need to know how to interact with others using their own endpoints (or through the mediators endpoints), in a development environment these endpoints would be *localhost:port*. But, to be able to access them from the outside, it was decided that using a tunneling software would make the solution more elegant and more close to reality. For this purpose, the open-source software pgrok[73] was used. It offers free introspected tunnels to localhost, and this way the agents have their own fixed endpoint.
For instructions on how to install the Trinsic Wallet Agent Mobile Application, please refer to the instructions on Appendix A, where a detailed description on how to setup the agent is provided.

[67] https://www.docker.com/
[68] https://docs.docker.com/compose/install/
[69] https://github.com/bcgov/indy-tails-server
[70] https://nestjs.com/
[71] https://angular.io/
[72] Used to curl specific endpoints to automate connections, issuance and verification of credentials for testing and demonstration purposes.
[73] https://github.com/jerson/pgrok

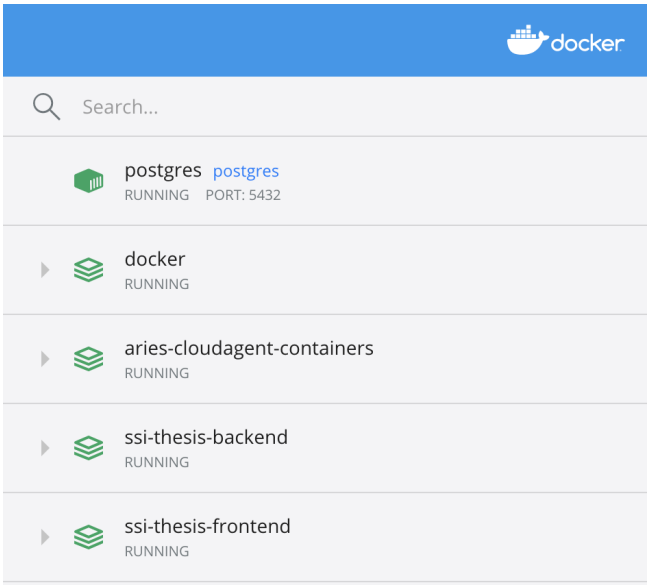

Figure 17. Docker Desktop with running containers

The best source of code documentation is present in the GitHub repository that complements this manuscript. It contains all the necessary information on how to deploy the system and where changes need to be made depending on different scenarios. Despite those instructions, a step-by-step guide will also be provided below.

Step 1: pgrok configuration To make sure the agents have their tunnels active before the start of the application, running the following line will start the tunneling service for each agent. This deployment follows the configuration created that will forward each agent's localhost port to a specific endpoint, exclusive for each agent.

```
1 pgrok -config=pgrok.yml start oemagent epagent evmediatoragent rdwagent csagent evagent emspagent
      cpoagent
```

Figure 18. pgrok in action, tunneling the localhost ports to specific endpoints, depending on the configuration

Step 2: Configuration File This file is crucial for the systems operation. It is located in the backend implementation (*ssi-thesis-backend/src/common/config/DefaultConfig.ts*). This is where the hard-coded information is written to, and from where the controller extracts information regarding:

- Agents endpoints;
- Public DIDs;
- Credential Schema IDs and Credential Definition IDs;
- Connection IDs

Step 3: *deployment.sh* script This script is in charge of deploying the revocation registry (tails-server), the agents, the backend, the postgres instance and the frontends. Each agent is deployed according to a configuration file specific to each agent. There is a particular step that is really important for the deployment of the agents. On startup, the mediator agent for the Electric Vehicle will create a mediation-invitation. This is printed in the command line and needs to be pasted to the *mediator-invitation* field on the EVs agent deployment configuration file (*ev.config.yml*). After pasting the mediatior-invitation to the configuration and pressing enter on the terminal, the deployment will carry on and both the EV and EV mediator agents mediation will be established.

Step 4: *establish_connections.sh* script For the sake of demonstrating the functionalities of the system, the agents connections are setup using a script, and the connection IDs are stored in a file for the agents to communicate between themselves. This script makes HTTP calls to the controller and that in turn makes contact with the agents to first create the connection invitations on the inviting agent, and receive the connection invitation on the receiver agent. Each connection ID that is printed (as seen in Figure 19) needs to be pasted on the correct field on the Configuration File, as explained before.

```
CPO Agent starts connecting to CS Agent
76c75116-4c66-4144-a561-b909c539ddfb

Wait for connection from CPO to CS agent to be established...
5bddcacd-4f03-4597-b0b6-66abf7d042ce
Done
```

Figure 19. How to extract the connection IDs after executing the script

Step 5: *deploy_schemas_and_credential_definitions* script After setting the version number in the Configuration file (field "schema_version"), the task of submitting schemas and credential definitions onto the ledger has been automated. The REST API calls the controller to create the schemas by making the calls to the agents' endpoints, this way the appropriate agent submits these onto the ledger (according to Table 10 in Section 4.3.2). Similar to the previous *establish_connections.sh* script, the shell script outputs the Schema IDs and the Credential Definition IDs that should be inserted into the Configuration file, as presented in Figure 20.

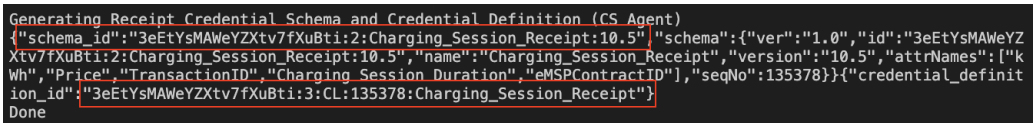

Figure 20. How to extract the schema and credential definition IDs after executing the script

5.2.2 Steps for running the use case scripts

After deploying the system, it is ready to power the use cases described in Section 4.3.3. The tasks that are conducted only between ACA-Py agents have been automated via shell scripts to hit the specific controller endpoints to trigger specific actions. All of the actions that require EV Owner interactions are meant to be used by means of the frontend instances deployed.

Step 1: *service_providers_flow.sh* script This script triggers the "Service Providers Flow" explained in Section 4.3.3, only from the eMSP, CPO, EP and CS perspectives.
It starts with the eMSP issuing the Contract with CPO credential to the CPO.

```
1 #Issue Contract with CPO to CPO from EMSP (CREDENTIAL 5)
2 curl -s -XPOST -H "Content-type: application/json" 'localhost:9229/emsp/issue-credential/cpo' > /tmp
      /output.html
3
4 printf '\nCredential is being issued from EMSP to CPO...\n'
5 sleep 2
6 printf '\nDone\n'
```

Then the CPO issues a credential to the Charging Station to attest its ownership.

```
#Issue Ownership of CS to CS from CPO (CREDENTIAL 6)
curl -s -XPOST -H "Content-type: application/json" 'localhost:9229/cpo/issue-credential/ownership-cs' > /tmp/output.html

printf '\nCredential is being issued from CPO to CS...\n'
sleep 2
printf '\nDone\n'
```

And at last, the EP issues a Contracting Credential to the CPO.

```
#Issue Contract with CPO to CPO from EP (CREDENTIAL 8)
curl -s -XPOST -H "Content-type: application/json" 'localhost:9229/ep/issue-credential/contract-cpo' > /tmp/output.html

printf '\nCredential is being issued from EP to CPO...\n'
sleep 2
printf '\nDone\n'
```

To complete the rest of the use case, the EV Owner performs the steps listed below. Note that visual representations of each step can be found in Appendix B.1:

1. Navigate to the frontend's URL (localhost:4200)
2. Select the eMSP portal (Figure 27).
3. Connect to the TA Agent using the QR Code, scanning it with the Trinsic Wallet App (Figure 28).
4. Press the "Issue Credential" button (Figure 29) and accept the credential on the Trinsic Wallet App (Figure 30).

Step 2: *ev_interactions.sh* script This script triggers the "EV Owner and EV Interactions with SSI" explained in Section 4.3.3, only from the EV, OEM and TA perspective.
It starts with the OEM agent issuing the VIN Credential to the Electric Vehicle.

```
#Issue VIN Credential to EV from OEM (CREDENTIAL 1)
curl -s -XPOST -H "Content-type: application/json" 'localhost:9229/oem/issue-credential/send' > /tmp/output.html
printf '\nCredential is being issued from OEM to EV...\n'
sleep 2
printf '\nDone\n'
```

Then, the TA verifies this credential, and issues another credential to attest the EVs registration.

```
#RDW Verify VIN Credential and Issue EV Registration to EV from RDW (CREDENTIAL 2)
printf '\nRDW is requesting VIN credential from EV\n'
curl -s -XPOST -H "Content-type: application/json" 'localhost:9229/rdw/present-proof/send-request' > /tmp/output.html
printf '\nRequest sent, waiting for reply from EV\n'
```

To complete the rest of the use case, the EV Owner performs the steps listed below. Note that visual representations of each step can be found in Appendix B.2:

1. Navigate to the frontend's URL (localhost:4200)
2. Select the TA portal (Figure 31).
3. Connect to the TA Agent using the QR Code, scanning it with the Trinsic Wallet App (Figure 32).
4. Press the "Issue Credential" button (Figure 33) and accept the credential on the Trinsic Wallet App (Figure 34).

Step 3: Performing the "Charging and Billing" flow This part of the system does not rely on any scripts, but rather just the EV Owner interacting with the frontend and the Trinsic Wallet App.
The steps are already described in Section 4.3.3, but a link between the steps and the screenshots of the frontend and the Trinsic Wallet App will be made below, similarly to what was done for the previous two scripts above.
To complete this use case, the EV Owner performs the steps listed below. Note that visual representations of each step can be found in Appendix B.3:

1. Navigate to the frontend's URL (localhost:4200)
2. Select the Charging Station Dashboard (Figure 35).
3. Connect to the Charging Station Agent using the QR Code, scanning it with the Trinsic Wallet App (Figure 36).
4. Press the "Request Proofs" button (Figure 37).
5. Receive the two presentation requests (Figure 38), and accept the Request for the Contract with eMSP credential (Figure 39) and the EV Owner Registration at the TA one (Figure 40).
6. At the Charging Station Dashboard, press the "Verify Proofs" button (Figure 41).
7. The credentials have now been verified by the agent as seen on the screen, as well as the eMSP Company name is presented on the screen (Figure 42).
8. Now that the Company Name is known, the EV Owner may request the current charge rate pressing the "Obtain Current Charge Rate" button (Figure 43).
9. The price/kWh is presented on the screen (Figure 44).
10. The EV Owner presses "Request EV Credentials" (Figure 45) so that the Charging Station can request from the EV agent a presentation to attest it is the same vehicle as indicated by the EV Owner.
11. After a few seconds, the EV Owner presses "Verify Proofs" to verify if the process has been concluded.
12. The credential from the EV has now been verified by the CS agent as seen on the screen, as well as the registration number is presented on the screen together with the EV Owner registration number (Figure 46).
13. Now the EV Owner may start charging the EV, pressing the "Start Charging" button (Figure 47).
14. The Charging Station will start the charging and stop it right away (Figure 48).
15. The EV Owner can now issue a receipt credential, pressing the "Issue Receipt Credential" button (Figure 49).
16. On the Trinsic Wallet App, the EV Owner will receive a credential offer (Figure 50).
17. The EV Owner accepts the credential by clicking the "Accept" button (Figure 51) and the credential is stored on the Trinsic Wallet App (Figure 52).
18. An overview of the final state of the Charging Station dashboard in the good-case scenario can be found in Figure 53.

6 Evaluation

In this chapter the system designed in Chapter 4 and implemented in Chapter 5 will be evaluated. First the methodology of this evaluation will be documented and then the results are demonstrated.
The evaluation will refer to a particular use case presented in Section 5.1. The evaluation for the general SSI with IoT architecture presented in Section 4.2 will be discussed at the end of this chapter.

6.1 Methodology

The evaluation will be split into three distinct steps:

- Benchmark the "Hyperledger Indy Network" and "Hyperledger Aries Cloud Agent - Python" in order to assess performance-, scalability- and availability-related NFRs;
- Qualitative evaluation of the proposed solution from domain experts;
- Cross-check requirements against the decisions taken throughout the project (and evaluation results).

Benchmark "Hyperledger Indy Network" and "Hyperledger Aries Cloud Agent - Python" This part of the evaluation will be made possible with the help of the development team of Hyperledger Indy, since performance and scalability tests were already made to the network. The results from these tests will be presented to validate the Non-Functional Requirements of the system.
Given that the agents are a core part of the system, being able to know the time they take to perform their tasks is imperative to know if these can be used at scale.

Qualitative evaluation from domain experts In order to measure the validity of the new system, and whether it can actually be considered an improvement to the current implementation (dissected in Section 3.1), a qualitative assessment from domain experts is a vital part of understanding whether some assumptions are valid, feasible or impossible. This evaluation process consisted of:

- Gathering contacts from various domain-specific experts that are related to the targeted use case (Energy, Blockchain, Self-Sovereign Identity).
- Inviting contacts to participate in the evaluation process, by having them sign up via a Google Forms with preliminary assessment questions, to profile the participants.
- Recording a presentation of the project with a small demo to send to the interested participants.
- Sending the interested evaluators the recorded presentation and demo, together with a Google Forms to assess their perspective on certain aspects of the system.

Cross-check requirements For this part of the evaluation, each requirement will be analyzed and discussed, to understand whether in the novel architecture of the system that given requirement is (partially) met, or not.

6.2 Benchmark "Hyperledger Indy Network" and "Hyperledger Aries Cloud Agent - Python"

Hyperledger Indy Network evaluation As mentioned before, assessing the capability of the network that is the backbone of the proposed solution is a major part of understanding if it can handle the proposed use case requirements. Investigating performance tests made to the network landed on an issue on the Indy's JIRA repository. Issue **INDY-1343 - Prove production stability of an Indy network**[74] arose from the following need: *"Before encouraging people to use the Sovrin network for live loads, we need to prove that it will be stable under conditions similar to production use."* Sovrin's name has been highlighted previously in the Related Work chapter (Section 2.4.1), and corresponds to the production level network created by the Sovrin Foundation.
The tests had mixed conclusions, with certain scenarios underperforming while others achieving the expected TPS throughput. After these tests, three more issues were lifted on the project's JIRA repository related to testing the performance of the network, INDY-1388[75], INDY-1607[76] and INDY-2214[77]. Table 15 contains information on the issues

[74] https://jira.hyperledger.org/browse/INDY-1343
[75] https://jira.hyperledger.org/browse/INDY-1388
[76] https://jira.hyperledger.org/browse/INDY-1607
[77] https://jira.hyperledger.org/browse/INDY-2214

Issue #	Title	Status
INDY-1343	Prove production stability of an Indy network	✔
INDY-1388	Prove stability under a DOS of an Indy network	✔
INDY-1607	Proof of stability under load	✔
INDY-2214	Repeat: Prove production stability of an Indy network	-

Table 15. Issues on Indy's JIRA Repository related to benchmarking the network

regarding network stability and their current status. It is worth noting that the latest issue (INDY-2214) has yet to be resolved, since December 2019. A detailed description of the tests can be found in a Google Sheets shared by the development team, that highlights their findings[78] (Available on 13th June 2021).
Although arguably the JIRA or the project's GitHub repositories are usually the best source of information regarding benchmarking tests, the best source to validate the current state of the Hyperledger Indy network comes directly from the Sovrin's foundation network status dashboard[79]. Figure 21 demonstrates the network status from 12th June 2020 to 12th June 2021, regarding various metrics. The most relevant information regards the read and write availability, that achieves an outstanding **99.999% read and write availability**.

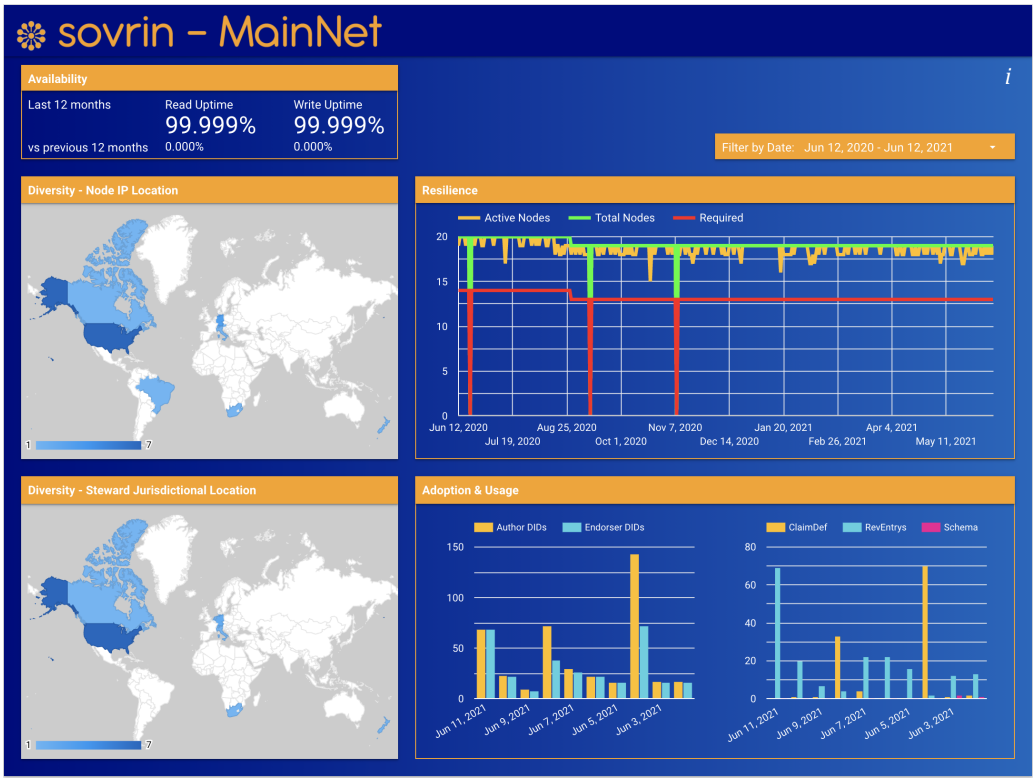

Figure 21. Sovrin Network (Hyperledger Indy nodes) metrics from 12 June 2020 to 12 June 2021

Hyperledger Aries Cloud Agent - Python evaluation In order to assess the performance of the agent implementation, two different methods were used: Use the performance benchmark tests provided by the development team; and measure the times when conducting actions with the agents in the proof-of-concept implementation.
For the first, in the original Aries Cloud Agent - Python repository the developers included a basic *performance.py* script that deploys two agents, and has one issue a predefined number of credentials to the other. These benchmarks are therefore a good indication of the times taken to issue credentials from one agent to another, and can vary depending on whether there is mediation between the two agents or not, and also if the credentials are revoked after they are issued. This latter also indicates the feasibility to revoke credentials, something that was not explored in-depth with the proof-of-concept.

[78] https://docs.google.com/spreadsheets/d/1DTjDsLSysFBiKU-9z4-IzunJk4wEy44hE_PGZYxnN_8/edit#gid=1813415708
[79] https://sovrin.org/ssi-metrics-dashboards/

Run #	Revocation	Mediation	Count	Average Credential Times
1	✗	✔	1000	0.23s
2	✔	✗	1000	0.23s
3	✗	✗	1000	0.23s
4	✔	✔	1000	0.26s

Table 16. Overview of the different attempts

For the second set of benchmarks to the agent implementation, the interactions of the agents will be timed manually to assess the performance requirements. The commands used to execute each of the runs, as well as more details extracted from the runs are listed in Appendix C. In this appendix the computer specifications are also listed for reference.
Looking at the results found in Table 16, it is possible to see that an ACA-Py agent is able to issue credentials to another agent in an acceptable time, averaging 0.23s in three of the runs and 0.26s in the last. Note that these results are limited by the fact that the agents are running in the same piece of hardware, which may lead to an improvement in the times. Still, the results are 10 times faster than the time of interactions of the current system's validation periods, as listed in **Requirement NFR-2.4** - *The process of validation the client's credentials should not take longer than the original method (designed for 2 seconds, with the worst case scenario being 30 seconds)*. For this reason, it is possible that the times may increase in a production environment, but not exceeding the optimal range of values.
Besides these tests made to the credential issuance process, the actions needed for the charging process were also manually timed. When performing the charging process, the "Request Proofs" action from the EV Owner took on average less than 2 seconds. Note that this step occurs between an ACA-Py agent and a Trinsic Wallet agent. For the process of the "Request EV Credentials" from the EV agent, this process also takes less than 2 seconds on average. This process involves the CS agent sending the request for the proof, the EV agent generating the proof, sending it back to the CS agent and then it being verified. At last, the "Issue Receipt Credential" process from the CS to the EV Owner's Trinsic Wallet agent also takes on average less than 2 seconds, but depends on the internet signal at the time of receiving the credential proposal on the Trinsic Wallet App dashboard.

6.3 Qualitative Evaluation from domain experts

In this section the results from the evaluation provided by the domain experts will be presented to extract results that can validate the requirements.

6.3.1 Preliminary Form Results

In the preliminary form shared with potential participants to the evaluation, the questions mostly related to profile questions, to understand the area expertise of the evaluators and how well they are aware of the charging process of EVs. A total of six (6) people responded to the form, confirming their willingness to evaluate the proposed solution. Note that some of the participants had expertise in more than one area as demonstrated by Table 17.

Field of Expertise	Count
Blockchain	4
Energy	3
Self-Sovereign Identity	1
Architecture	1
EV Automotive	1

Table 17. Expertise of the Evaluators

From the six respondents, the follow data was extracted:

- 3 out of 6 responded to be able to "teach someone else" about blockchain.
- 2 out of 6 responded to be able to "teach someone else" about Self-Sovereign Identity.

- None of the respondents own an EV, but more than half (4) of them are knowledged in the process of charging an EV. Out of those 4 people, 4 are aware of (most of) the parties in the process, 3 know how to charge an EV, and only 2 of them are aware of the current payment model.

6.3.2 Results from experts' evaluation of system

After gathering the profiles of the six evaluators, a video containing a pre-recorded slide presentation explaining the project was shared with the latters. In this video it was included part of the proof-of-concept's demonstration of how an EV Owner is able to charge its EV making use of Verifiable Credentials to attest the claims explained in the flow present in Section 4.3.3.
This expert evaluation was mostly used to assess the usability requirements as well as the business value of the novel architecture. From the six (6) evaluators that agreed to answer the preliminary form, the following results arose:
When asked to prioritize "Ease of Use", "Data Privacy", "Cost Transparency", "Availability" and "Integration with Mobile Applications" in an EV Charging Network from the least important (1) to the most important (5), these were the results:

- Availability and Cost Transparency both obtained 23 points, making these two the most prioritized quality attributes;
- Ease of Use obtained 22 points, close to the first two quality attributes.
- Data Privacy and Integration with Mobile Applications scored 14 and 8 points, respectively, demonstrating that these two are not the highest priority for the evaluators.

The questions below constitute the questionnaire made to the evaluators. Each of them was given an ID.

- #1 - *How understandable is the presented solution, from a user and business perspective?*
- #2 - *How do you think the solution compares with the current implementation of the network?*
- #3 - *With this solution, would you feel confident your personal data is protected?*
- #4 - *With this solution, would you consider the process of charging the vehicle more transparent?*
- #5 - *Do you consider the employed payment model user-friendly?*
- #6 - *Which feature(s) from this solution did you appreciate the most?*
- #7 - *Which feature(s) would you like to see added to this solution?*
- #8 - *Do you have any other comments on the presented solution?*

From the aforementioned questions, the more relevant finding were:

- 4 out of 6 believe that the solution is "Very easy to understand" from a user perspective while 3 out of 6 thought the same from a business perspective.
- 4 out of 6 believe that the new solution is much better than the current implementation of the network.
- At least 6 out of 6 "Agree" with question #3, with 3 of them selecting "Strongly Agree".
- The same results from question #3 were obtained for question #4.
- 5 out of 6 "Agree" to question #5 regarding the user-friendliness of the payment model.
- The feature with the most appreciation was the "Added more transparency with the addition of the kWh price" feature, with 4 evaluators selecting that option for question #6. 3 evaluators also enjoyed the "Removal of Login", 2 selected "Added Vehicle and Owner verification for extra security" and 1 liked the "Removal of external party - Maximising revenue" feature. None of the evaluators saw great value in the "Added receipt-credential to the end of the transaction".

For question #7 regarding possible future features to add, the comments made by the evaluators are found below.

"I know that right now there is a solution to have multiple suppliers per charge poll and users can charge their cars in different location. So an approach is based on you credentials and your supplier contract, this should be easy to identify and get a message saying you are now charging your car with supplier A in this rate."

"Using a unique vehicle identity in the process might give away the owner or driver, compromising privacy. Zero-knowledge might help alleviate that iff the vehicle ID is not anyway shared with the charging station."

"Include price options in case you are not in a hurry to charge and the network can use the optimal moment to charge, although that is not really related to the SSI part."

"try to remove as much steps in the user interaction as possible."

For question #8 regarding additional comments or the overall appraisal of the system, the comments made by the evaluators are found below.

"The overall process is well thought-out and nicely implemented."

"I understand the value of SSI for identifying myself as the owner or the car and the one charging it, however what is the benefit of sharing the receipt as a VC? That is also shared with the energy supplier, right? Or is that to resolve a dispute on the invoice in case the supplier charges you more? Or do you see scenario's where I would need to share that with some other party?"

"Excelent demo showing the way forward for EV charging"

"Although data privacy is not a major requirement for me, I would still prefer to have something that protects me as per your solution than what is currently in the market today"

"Need to think about multiple users of a single vehicle and also Fleet driver use case."

The overall assessment of this evaluation method allowed to understand how domain experts in the area perceive the system, which in turn granted good prospects to fulfill the necessary requirements. All evaluators provided valuable feedback and the consensus was that the solution is an improvement to the previous iteration, providing more transparency to the system and improved the user's experience with the system. The privacy limitations regarded in the comments will be addressed at a later stage, when cross-checking the requirements.

6.4 Cross-check requirements

In this section, the requirements listed in Section 3.2 will be evaluated, and a link to the respective motivation and rationales is traced to a more detailed description that will be found in Appendix D. These motivations reflect whether the specific requirement is satisfied or not using the decisions taken across the document (from technological, architectural or assumptions). For more information regarding the assessment of each requirement, the appendix contains all the necessary information for that purpose. Despite that, a few of the most important requirements will be discussed to highlight the most important decisions.
For both upcoming sections, a table has been made containing a mapping between the RequirementID, the priority of the requirement, the approval status and a link to the respective detailed table on the appendix. Below it is possible to find a legend for the different status symbols and their meaning.

- ✔: Requirement fulfilled
- **-** : Needs further investigation or inconclusive
- ✗: Requirement not fulfilled

6.4.1 Functional Requirements

Table 18 contains the aforementioned table, and it is possible to observe that regarding the Functional Requirements, almost all of the requirements were met, with the exception of one whose priority is a "Could have" and another which is

prioritized as a "Must have".
Starting with the most important requirements that were fulfilled, the ones that add more value to the novel system and eliminate the liabilities presented in the case description in Section 3.1.2 are FR-1.2, FR-1.4, FR-3.1 and FR-5.1. Requirement FR-1.2 reflects on the fact that now the EV Owner is presented with a kWh rate before the charging session takes place, allowing the owner to decide whether to charge the EV at that time. This price is dependant on the contracted eMSP company, given that different companies might obtain better contracts with the CPOs and provide better prices for their clients. Requirement FR-1.4's fulfillment provides a way for the EV Owners to obtain a receipt at the end of each charging session, in the form of a Verifiable Credential. This allows the client to obtain proof of that transaction, with information regarding the full price of the charging session, providing more transparency to the latter. Requirement FR-3.1 addresses the fact that the EV Owner now is forced to prove ownership of the EV in order to charge the vehicle, giving the system an extra layer of security, preventing thieves from charging stolen EVs. Requirement FR-5.1 concerns matters of privacy, since Personally Identifiable Information (PII) should not be written to any publicly accessible repository, to comply with current privacy regulations. This matter is fulfilled with the current SSI implementation, that only writes credentials and private DIDs (PII) to the entity's wallet and not in the DLT itself, complying with this requirement. For the requirements that were not addressed by the system (FR-3.3 and FR-6.1), the rationale behind their absence from the system will be detailed when explaining the limitations of the system in Section 6.4.3.

ID	Priority	Status	Appendix
FR-1.1	Must	✔	FR-1.1
FR-1.2	Should	✔	FR-1.2
FR-1.3	Must	✔	FR-1.3
FR-1.4	Could	✔	FR-1.4
FR-2.1	Must	✔	FR-2.1
FR-2.2	Must	✔	FR-2.2
FR-2.3	Must	✔	FR-2.3
FR-3.1	Should	✔	FR-3.1
FR-3.2	Must	✔	FR-3.2
FR-3.3	Could	✗	FR-3.3
FR-3.4	Should	✔	FR-3.4
FR-4.1	Must	✔	FR-4.1
FR-5.1	Must	✔	FR-5.1
FR-5.2	Must	✔	FR-5.2
FR-6.1	Must	✗	FR-6.1

Table 18. Requirements Verification (Functional Requirements)

6.4.2 Non-Functional Requirements

Table 19 contains the aforementioned table, and it is possible to observe that regarding the Non-Functional Requirements, almost all of the requirements were met, with the exception of one whose priority is a "Must have". Two of the requirements were marked with **"-"** to indicate that they were inconclusive or that they would need further investigation.
From the requirements that were fulfilled, the most important findings would be related to NFR-2.4, NFR-3.3 and NFR-4.3. These assess some of the most important quality attributes listed in Chapter 3: Usability, Security/Privacy and also Availability. Requirement NFR-2.4 reflects on the time taken by the system to verify credentials. Following the evaluation made to the Hyperledger Indy and ACA-Py infrastructures in Section 6.2, it was evident that the credential issuance and verification process is aligned with the time taken in the current non-SSI implementation of the system. For this reason, it was concluded that this requirement was met. Requirement NFR-3.3's fulfillment provides the system with an indisputable way to prevent forgery of credential. Based on the concepts of Self-Sovereign Identity and Verifiable Credentials, each credential held by the EV Owner/client is practically impossible to forge, tamper or spoof, given that it is based on advanced cryptographic methods. Requirement NFR-4.3 addresses the system's availability. The current surveys on the EV charging systems proposed that these systems should guarantee an uptime of at least 97% of the time. After the benchmarks conducted in Section 6.2, it was possible to see that the best example for an Hyperledger Indy network is able to guarantee a 99.999% uptime at a production level, fulfilling this requirement.

A note goes also to NFR-4.1, given that the CSs need to be online in order to communicate to the ACA-Py agents. This introduces a difficult requirement to evaluate, given that this mostly relies on the applied area where the solution shall be used, and its network coverage. Assuming an adoption in the Netherlands, and according to an analysis report conducted by Eurostat[80] in 2018, the Netherlands are the country in the top 28 European Union countries with the greatest network coverage, with an astounding 98% of Dutch households having internet access. From this data, it is possible to foresee that this requirement shall be achieved with relative ease and that is why it was marked with a ✔.
Two of the requirements still need more research to assess their fulfillment, namely NFR-1.1 and NFR-3.2. The first regards the scalability of the system, which heavily relies on the supporting network (Hyperledger Indy). For the second requirement which was tagged with **"-"**, regarding GDPR compliance, the decision to mark it with **"-"** takes in consideration two choices/simplifications made in the current system. These limitations will be detailed in Section 6.4.3.

ID	Priority	Status	Appendix
NFR-1.1	Must	-	NFR-1.1
NFR-2.1	Should	✔	NFR-2.1
NFR-2.2	Must	✔	NFR-2.2
NFR-2.3	Must	✔	NFR-2.3
NFR-2.4	Should	✔	NFR-2.4
NFR-3.1	Should	✔	NFR-3.1
NFR-3.2	Must	-	NFR-3.2
NFR-3.3	Must	✔	NFR-3.3
NFR-4.1	Must	✔	NFR-4.1
NFR-4.2	Must	✔	NFR-4.2
NFR-4.3	Must	✔	NFR-4.3

Table 19. Requirements Verification (Non-Functional Requirements)

6.4.3 Limitations

In this section the limitations of the system will be discussed, following the discussion on the assessment of the requirements. For some of the requirements in the system, assumptions or simplifications were made, bearing in mind that work is being done to mitigate these assumptions or that the ecosystem is not yet mature enough to handle these features.

EV Owner not requesting information from the Charging Station Requirement FR-3.3 - *An EV Owner could be made aware that the CS belongs to the CPO it claims*, was not fulfilled given that the current implementation of the Mobile agent is very limited and does not allow to request credentials from another agent. This may eventually be achievable if a custom agent is made for this purpose, or if the act of presenting the credential is engaged by the CS. Additionally, and although this requirement would add a bit more transparency to the process, it could have damaged user-friendliness.

Lack of ability to delegate credentials in current agent implementations Requirement FR-6.1 - *An EV Owner must be able to delegate (temporary) possession of its EV to another driver*, was not fulfilled due to the lack of implementation in the current agents of a feature called "Delegated Credentials". Delegated Credentials are a topic which has not been addressed previously, since its implementation may differ from the current idea and prospect. Essentially, delegated credentials are credentials which are delegated to another entity by the holder of that credential. In the scope of the case handled in this study, a delegated credential would allow an individual to have temporary ownership of the EV, giving that individual power to charge the vehicle. This delegated credential should be chained to the original credential, and, in case the original credential is revoked, the delegated credential also loses its validity. "Delegated Credentials" can also go by the name of "Chained Credentials", but the core idea remains the same. This concept has been presented as an RFC in the Hyperledger Aries ecosystem, labelled *Aries RFC 0104: Chained Credentials*[81].

[80] https://www.cbs.nl/en-gb/news/2018/05/the-netherlands-leads-europe-in-internet-access
[81] https://github.com/hyperledger/aries-rfcs/blob/master/concepts/0104-chained-credentials/README.md

Whether this feature would be present in the ACA-Py agents was something that was discussed with the development team at the official ACA-Py rocket chat. The discussion landed on the following response presented in Figure 22 and that can be seen in the following thread[82].

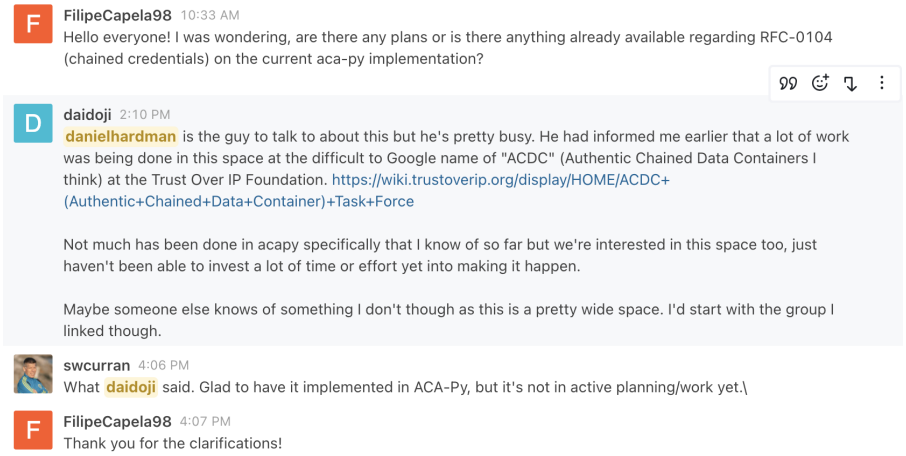

Figure 22. Thread discussing future plans for ACA-Py on the RFC-104: Chained Credentials

Scalability of the Hyperledger Indy Network Requirement NFR-1.1 - *The system must scale according to the growth of the market. Supposing an adoption in the Dutch Market (300k EVs in 2020), the system must be able to support this number of vehicles and the estimated growth (3M EVs by 2030)*, was assessed to the extent possible (as seen in Section 6.2) but the results were inconclusive since they are not updated anymore, not reflecting the current scalability of the Hyperledger Indy Network. Hence it is difficult to understand whether the current implementation of the Hyperledger Indy Network would sustain that many EVs. This was the main reason why this requirement was tagged with the **"-"** tag.

Link between EV/EV Owner and GDPR Requirement NFR-3.2 - *The system must comply with Dutch regulations, including GDPR* was tagged with **"-"** regarding its fulfillment in the novel system, given the assumption made to link both the EV and the EVOwner's registrations at the TA. The method used in this system is oversimplified, since the CS essentially validates the ownership of the EV by matching the credentialIDs in both credentials. This imposes linkability issues that are not GDPR compliant, so this matter needs to be investigated in the future to prevent linkability issues.

Mobile Agent and GDPR concerns Again, regarding Requirement NFR-3.2 - *The system must comply with Dutch regulations, including GDPR* there was another motive why it was tagged with **-**, given that the option to choose the Trinsic Wallet agent has its drawbacks, since this agent is regarded as a *thin agent*. This definition comes from the fact that it stores all of the data (credentials) on the cloud, which is not the best privacy-preserving solution and ultimately defeats the goal of SSI that declares that all PPI should be held by the owner of the data. For this reason, this issue is known, and with a mobile agent that stores the credentials on the device compliance with GDPR would ultimately be achieved regarding this aspect.

Connection of Charging Station and Electric Vehicle In the current implementation of the system, a hard assumption has been made in the way the CS agent connects to the EV. These are manually connected to demonstrate how the request between the agents can occur so that the CS could request proof from the EV regarding its registration with a TA. This assumption was made given that the EV has no way to connect to the CS in an out-of-band fashion (using a QR Code), and the fact that the EV does not make use of a Public DID (for security and privacy matters), which in turn invalidates the possibility for the CS to be the one to engage for the connection with the EV.
Work has been made towards how data can be shared between EVs and CSs, in a video-presentation made by Vector[83], an eMobility-dedicated company that demonstrated the current state-of-the-art protocols, seen in Figure 23. Here it is possible to see the use of the ISO 15118 standard, named *"Road Vehicles – Vehicle to grid communication interface"* which regards how the EV can connect to the grid via CSs, which accounts for bilateral interchange of information

[82] https://chat.hyperledger.org/channel/aries-cloudagent-python/thread/QSkomLPYHQsvzyHS6
[83] https://www.vector.com/int/en/

between the two parties. Given the current state of the standards, it may be a matter of time until the information that is shared includes a connection request from the EV to the CS, which would then power the rest of the use case.

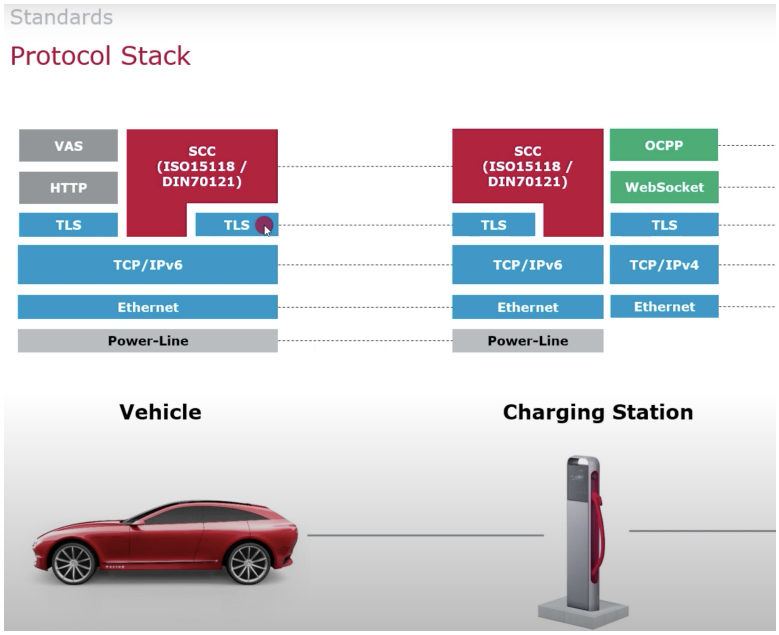

Figure 23. Current Electric Vehicle and Charging Station protocols [84]

6.5 Evaluation of the General Architecture of SSI with IoT

It was evident that the Verifiable Electric Vehicle Charging System as proposed in this work is able to fulfill most of the requirements, with the ones that were not having been provided documentation in order to apply them in the future.
Despite having presented one use case of Self-Sovereign Identity with Internet of Things devices (EVs and CSs) that presents good prospects for adoption, assessing the general architecture presented on Section 4.2 requires a more in-depth analysis, making use of other case studies. For this end, more use cases need to be discussed to give power to this idea. Nonetheless, it was possible to verify that in IoT devices that do not lack computational power and electricity, there are existing implementations of agents and networks that could be used by such devices, like an EV. On top of that, the general architecture presents the necessary components for any Self-Sovereign Identity system that has agents acting on behalf of any entity, whether humans, organizations, or IoT devices.

[84] https://www.youtube.com/watch?v=S_llUuyAcqs

7 Conclusion

In this chapter the conclusions for this study are presented, by doing a summary of the study and its main contributions. This study aimed at responding to the following research question: *"What are the requirements for Self-Sovereign Identity (SSI) to be implemented effectively in an Internet of Things (IoT) environment?"*.

Towards addressing this question, in Chapter 2 background information was provided on the topic of (Digital) Identity, the history of Identity Management Systems through time, and then the fundamentals of Self-Sovereign Identity were presented. The latter included mentions to Decentralized Identifiers, *DIDComm*, Verifiable Credentials, Wallets, Agents, etc. Still in this chapter, the related work was presented, where it was evident that the field of Self-Sovereign Identity with IoT is not yet mature, but provided much information on the advantages of a blockchain-based IdMS. Much of the literature on SSI with IoT provided an analysis on the current state of technologies and solutions available for that purpose, from which a comparison was made, to understand the differences between them.

Chapter 3 narrowed the field of research from SSI with IoT to the automotive domain and more specifically to the Electric Vehicle Charging use case. Here the current state of Electric Vehicle Charging Networks was presented. The use case was analyzed with the assistance of domain experts, from which the major liabilities were extracted, alongside a set of requirements with regards to the necessary settlements between the different parties in order for the owner of an EV to be able to charge its vehicle.

In Chapter 4 the system design was presented, where the architecture of a novel Verifiable Electric Vehicle Charging network architecture was presented, based on a set of technological decisions taken at the beginning of that same chapter that would facilitate the largest number of requirements. Afterwards, the system was analyzed with regards to the changes needed in the original flow of interactions, to accommodate for the SSI-driven approach.

Chapter 5 outlined the necessary steps used to implement the system and how to deploy it. The system consists of a dockerized full stack application that is able to interact with the agents and perform custom actions tailored for the particular use case at hand.

In Chapter 6 the assessment of the system was made, through means of benchmarks, questionnaires made to domain experts and cross-checking the requirements with decisions taken across this study. The overall assessment was positive given that almost all requirements were met, and the ones that were not fulfilled were discussed and their fulfillment should be achievable if the measures listed are addressed in the future. Regarding the evaluation of the general SSI with IoT research question, a step has been taken which makes that prospect possible in a near future, but more use cases need to be investigated to determine more limitations and explore different IoT devices, for example sensors, home assistants, or others.

8 Future Work

In this chapter future work is sketched in order to provide more solidity to the claims made in the tackled use case (Verifiable Electric Vehicle Charging) and to validate the general case of SSI with IoT.

Experiment with other DLTs

Given that the decision to opt for Hyperledger Indy for the DLT was taken largely due to its active community and public availability, in detriment of the IOTA Tangle approach or any of the others listed in Section 2.4, it would be interesting to assess the differences in the implementation whether a different DLT was adopted. Possibly with the IOTA Tangle users would have a more scalable solution, which might address Requirement NFR-1.1 that was marked as inconclusive with the Hyperledger Indy approach.

Investigate with other use cases

As mentioned at the end of Chapter 6, assessing more use cases is crucial to understanding the possibilities of incorporating SSI with IoT. This project focuses on IoT devices that fortunately do not lack any computational power or electricity, which is an assumption that cannot be made for all IoT devices. Therefore it is imperative to study how SSI could be enabled for low-capacity IoT devices to be able to respond with more certainty to the presented research question.

Merge proof requests

Although a small improvement to the system, it would be interesting to assess the feasibility of merging two or more credential requests into a single presentation request, in order to minimize user interactions with the system, as required by Requirement FR-3.4.

Address GDPR compliance and other limitations

One of the pending matters of this project is the fact that, given the fact that it was built on top of a proof-of-concept, privacy concerns in the proposed solution were simplified, leaving room for linkability between the EV Owner and its EV. This has been addressed as a limitation in the Evaluation chapter (Section 6.4.3), and in order to account for these liabilities, it is necessary to create means to link the EV Owner with its EV using a mechanism different than the usage of a unique identifier between the two. A possible approach would be to generate a different number every session so that the link between the two could not be established by the Charging Station, but the feasibility of this mechanism can be a topic for a thesis of its own. The remaining of the limitations also need to be cleared in order to think about applying this solution to a prototype/production level application.

9 Bibliography

[1] C. for International Governance Innovation and Ipsos, "Cigi-ipsos global survey on internet security and trust," 2019.

[2] M. Luecking, C. Fries, R. Lamberti, and W. Stork, "Decentralized identity and trust management framework for internet of things," in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pp. 1–9, IEEE, 2020.

[3] "Internet of Things (IoT) Preliminary Report 2014," standard, International Organization for Standardization, Geneva, CH, 2015.

[4] S. Asiri and A. Miri, "A sybil resistant iot trust model using blockchains," pp. 1017–1026, 07 2018.

[5] S. Terzi, C. Savvaidis, K. Votis, D. Tzovaras, and I. Stamelos, "Securing emission data of smart vehicles with blockchain and self-sovereign identities," in *2020 IEEE International Conference on Blockchain (Blockchain)*, pp. 462–469, IEEE, 2020.

[6] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, pp. 131–164, Dec. 2008.

[7] I. J. Do Van Thanh, "Identity management," *Telektronikk – Strategies in Telecommunications*, vol. 103, no. 3/4, pp. 3–10, 2007.

[8] I. Fernmelde-Union, *Digital Identity Roadmap Guide*. 06 2021.

[9] E. Bertino and K. Takahashi, *Identity management: Concepts, technologies, and systems*. Artech House, 2010.

[10] X. Zhu and Y. Badr, "Identity management systems for the internet of things: a survey towards blockchain solutions," *Sensors*, vol. 18, no. 12, p. 4215, 2018.

[11] S. K. Gebresilassie, J. Rafferty, P. Morrow, L. L. Chen, M. Abu-Tair, and Z. Cui, "Distributed, secure, self-sovereign identity for iot devices," in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, pp. 1–6, IEEE, 2020.

[12] A. Jøsang and S. Pope, "Auscert conference 2005 user centric identity management," 2005.

[13] D. van Bokkem, R. Hageman, G. Koning, L. Nguyen, and N. Zarin, "Self-sovereign identity solutions: The necessity of blockchain technology," *arXiv preprint arXiv:1904.12816*, 2019.

[14] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2019.

[15] "Blockchain and distributed ledger technologies — Vocabulary," standard, International Organization for Standardization, Geneva, CH, 2020.

[16] A. Panarello, N. Tapas, G. Merlino, F. Longo, and A. Puliafito, "Blockchain and iot integration: A systematic survey," *Sensors*, vol. 18, no. 8, 2018.

[17] X. Wang, X. Zha, G. Yu, W. Ni, and R. Liu, *Blockchain for Internet of Things*, pp. 87–136. 11 2020.

[18] S. K. Lo, Y. Liu, S. Y. Chia, X. Xu, Q. Lu, L. Zhu, and H. Ning, "Analysis of blockchain solutions for iot: A systematic literature review," *IEEE Access*, vol. 7, pp. 58822–58835, 2019.

[19] X. Fan, Q. Chai, L. Xu, and D. Guo, "Diam-iot: A decentralized identity and access management framework for internet of things," in *Proceedings of the 2nd ACM International Symposium on Blockchain and Secure Critical Infrastructure*, BSCI '20, (New York, NY, USA), p. 186–191, Association for Computing Machinery, 2020.

[20] Y. Liu, D. He, M. S. Obaidat, N. Kumar, M. K. Khan, K.-K. R. Choo, *et al.*, "Blockchain-based identity management systems: A review," *Journal of Network and Computer Applications*, p. 102731, 2020.

[21] B. Zundel, M. Sporny, G. Noble, D. Burnett, and D. Longley, "Verifiable credentials data model 1.0," W3C recommendation, W3C, Nov. 2019. https://www.w3.org/TR/2019/REC-vc-data-model-20191119/.

[22] Y. Kortesniemi, D. Lagutin, T. Elo, and N. Fotiou, "Improving the privacy of iot with decentralised identifiers (dids)," *Journal of Computer Networks and Communications*, vol. 2019, 2019.

[23] A. Theodouli, K. Moschou, K. Votis, D. Tzovaras, J. Lauinger, and S. Steinhorst, "Towards a blockchain-based identity and trust management framework for the iov ecosystem," in *2020 Global Internet of Things Summit (GIoTS)*, pp. 1–6, IEEE, 2020.

[24] G. Fedrecheski, J. M. Rabaey, L. C. Costa, P. C. C. Ccori, W. T. Pereira, and M. K. Zuffo, "Self-sovereign identity for iot environments: a perspective," in *2020 Global Internet of Things Summit (GIoTS)*, pp. 1–6, IEEE, 2020.

[25] D. Longley, C. Allen, D. Reed, M. Sabadello, and M. Sporny, "Decentralized identifiers (DIDs) v1.0," W3C working draft, W3C, Mar. 2021. https://www.w3.org/TR/2021/WD-did-core-20210302/.

[26] M. Sabadello, D. Reed, D. Longley, M. Sporny, and C. Allen, "Decentralized identifiers (DIDs) v1.0," W3C working draft, W3C, Feb. 2021. https://www.w3.org/TR/2021/WD-did-core-20210222/.

[27] K. T. Nguyen, M. Laurent, and N. Oualha, "Survey on secure communication protocols for the internet of things," *Ad Hoc Networks*, vol. 32, pp. 17–31, 2015.

[28] S. Foundation, "White paper: Self-sovereign and iot," tech. rep., August 2020.

[29] M. Sabadello, D. Reed, D. Longley, M. Sporny, and C. Allen, "Verifiable credentials implementation guidelines 1.0," W3C working draft, W3C, Feb. 2021. https://www.w3.org/TR/vc-imp-guide/.

[30] P. C. Bartolomeu, E. Vieira, S. M. Hosseini, and J. Ferreira, "Self-sovereign identity: Use-cases, technologies, and challenges for industrial iot," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1173–1180, IEEE, 2019.

[31] M. Davie, D. Gisolfi, D. Hardman, J. Jordan, D. O'Donnell, and D. Reed, "The trust over ip stack," *IEEE Communications Standards Magazine*, vol. 3, no. 4, pp. 46–51, 2019.

[32] N. Kulabukhova, A. Ivashchenko, I. Tipikin, and I. Minin, "Self-sovereign identity for iot devices," in *International Conference on Computational Science and Its Applications*, pp. 472–484, Springer, 2019.

[33] M. Sharma and J. Lim, "A survey of methods guaranteeing user privacy based on blockchain in internet-of -things," in *Proceedings of the 2019 2nd International Conference on Data Science and Information Technology*, DSIT 2019, (New York, NY, USA), p. 147–153, Association for Computing Machinery, 2019.

[34] X. Zhu, Y. Badr, J. Pacheco, and S. Hariri, "Autonomic identity framework for the internet of things," in *2017 International Conference on Cloud and Autonomic Computing (ICCAC)*, pp. 69–79, IEEE, 2017.

[35] Y. Cheddadi, H. Cheddadi, F. Cheddadi, F. Errahimi, and N. Es-sbai, "Design and implementation of an intelligent low-cost iot solution for energy monitoring of photovoltaic stations," *SN Applied Sciences*, vol. 2, pp. 1–11, 2020.

[36] P. Dekker and V. Andrikopoulos, "Automating bulk commodity trading using smart contracts," in *2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pp. 52–60, IEEE, 2020.

[37] N. Wang, X. Zhou, X. Lu, Z. Guan, L. Wu, X. Du, and M. Guizani, "When energy trading meets blockchain in electrical power system: The state of the art," *Applied Sciences*, vol. 9, no. 8, p. 1561, 2019.

[38] J. Liu, Y. Xiao, and C. P. Chen, "Authentication and access control in the internet of things," in *2012 32nd International Conference on Distributed Computing Systems Workshops*, pp. 588–592, IEEE, 2012.

[39] K. Salah, "Iot access control and authentication management via blockchain," 06 2018.

[40] C. Lupascu, A. Lupascu, and I. Bica, "Dlt based authentication framework for industrial iot devices," *Sensors*, vol. 20, no. 9, p. 2621, 2020.

[41] IOTA, "The case for a unified identity," iota whitepaper, IOTA, 2020. https://files.iota.org/comms/IOTA$_{T}he_{C}ase_{f}or_{a}_{U}nified_{I}dentity.pdf$.

[42] T. . Environment, "How many charge points will europe and its member states need in the 2020s," tech. rep., Transport Environment, Jan. 2020. https://www.transportenvironment.org/sites/te/files/publications/01 2020 Draft TE Infrastructure Report Final.pdf.

Appendices

A How to Install Trinsic Wallet Agent Mobile Application

The process of installing and setting up the Trinsic Wallet Agent depends on the operating system, with the application being available on both Apple's App Store and also Android's Google Play Store. During this project the Android version was used and that is the version that will be explained below.

- Navigate to the desired Store and download the Trinsic Wallet App. Either Google Play Store[85] or Apple's App Store[86].
- Follow the in-app instructions to setup a wallet.
- Follow the instructions on Figures 24, 25 and 26 to change to the same network as the ACA-Py agents.

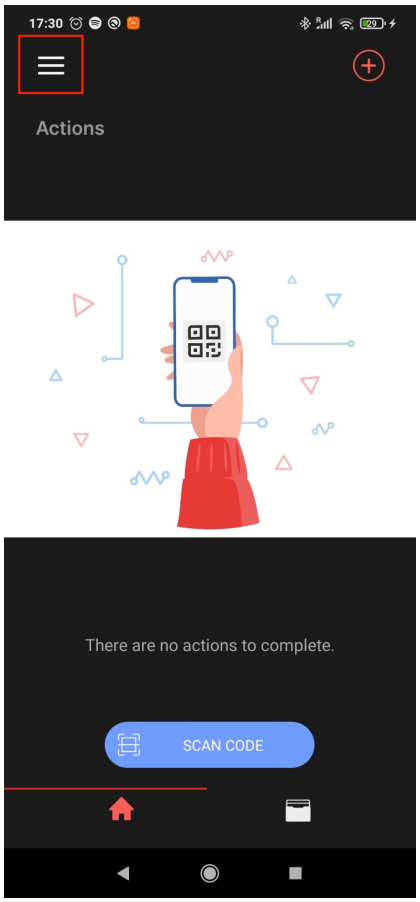

Figure 24. Go to Settings

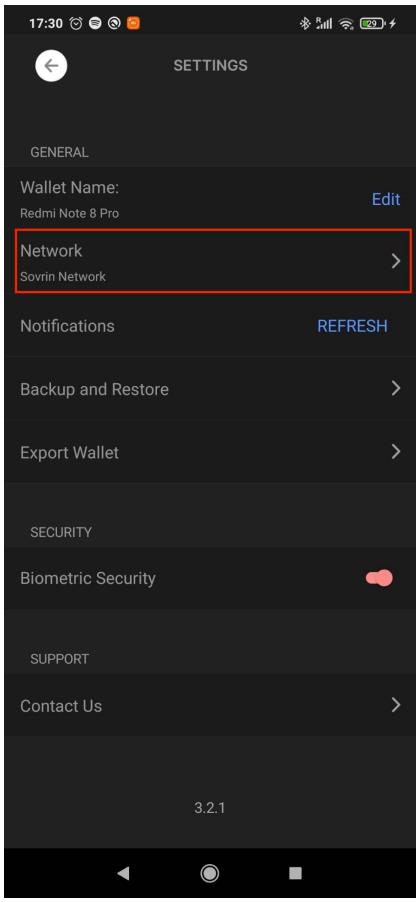

Figure 25. Choose "Network" Option

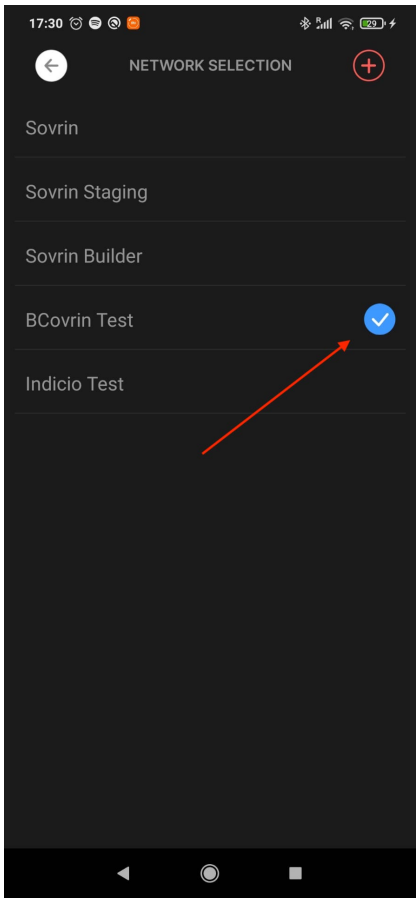

Figure 26. Change it to "BCovrin Test"

[85] https://play.google.com/store/apps/details?id=id.streetcred.apps.mobile
[86] https://apps.apple.com/us/app/trinsic-wallet/id1475160728

B Application Screenshots

In this appendix the screenshots of the frontend as well as from the Trinsic Wallet App on the EV Owner's phone are presented, to illustrate each step of each use-case.

B.1 Service Providers Flow with SSI Screenshots

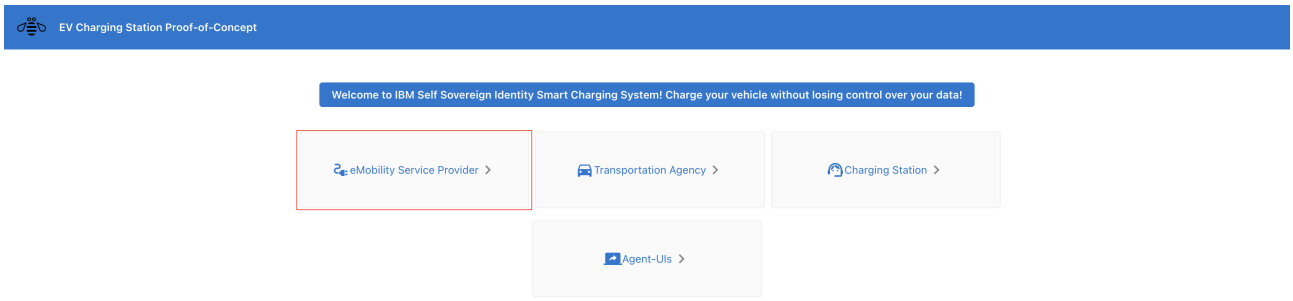

Figure 27. Selecting eMSP portal

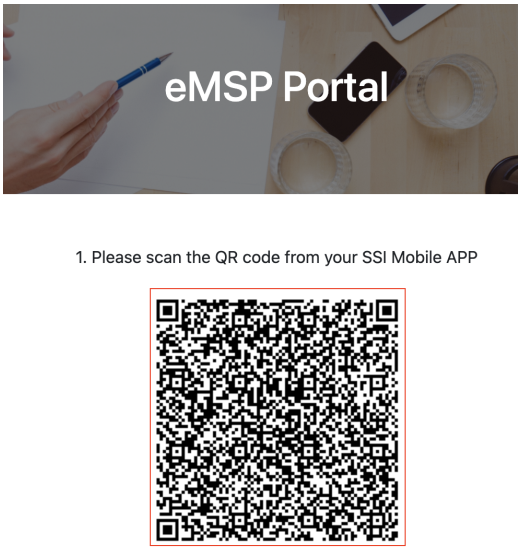

Figure 28. Connect to eMSP Agent using QR Code

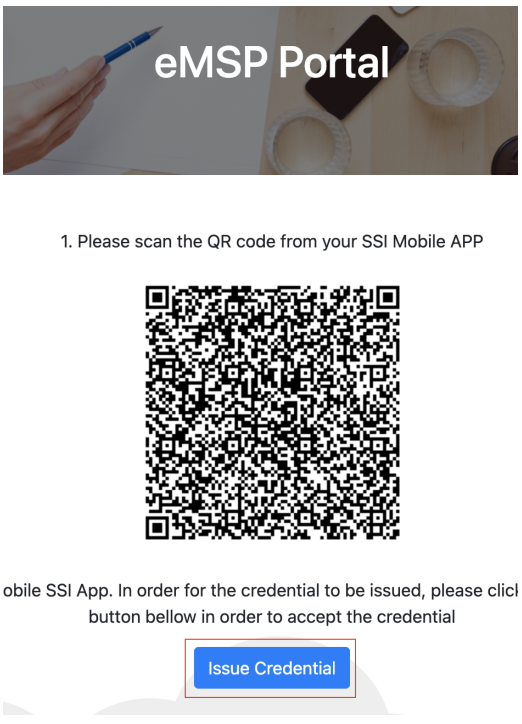

Figure 29. Press the "Issue Credential" button

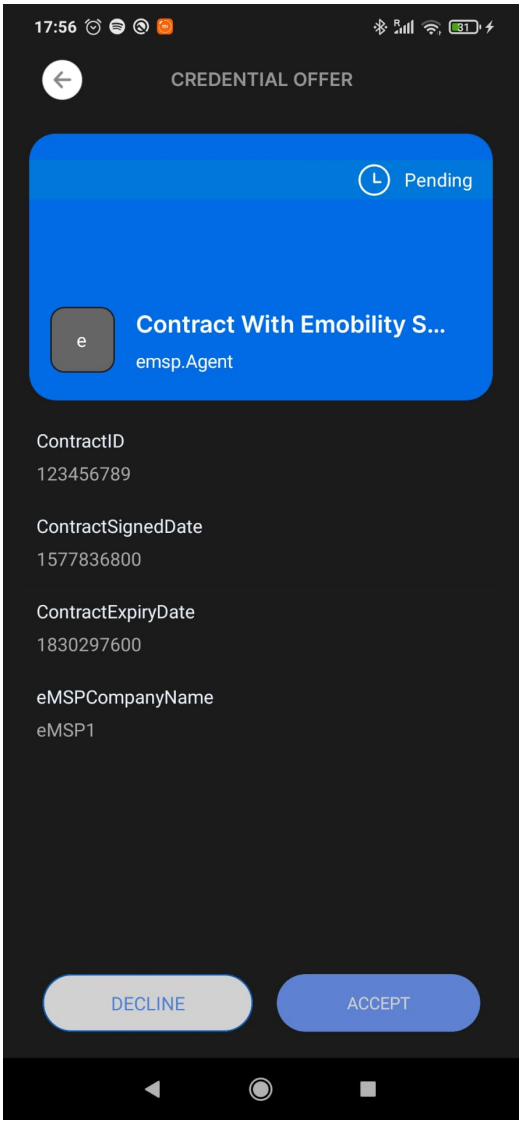

Figure 30. Accept the credential on the Trinsic Wallet App

B.2 EV Owner and EV Interactions with SSI Screenshots

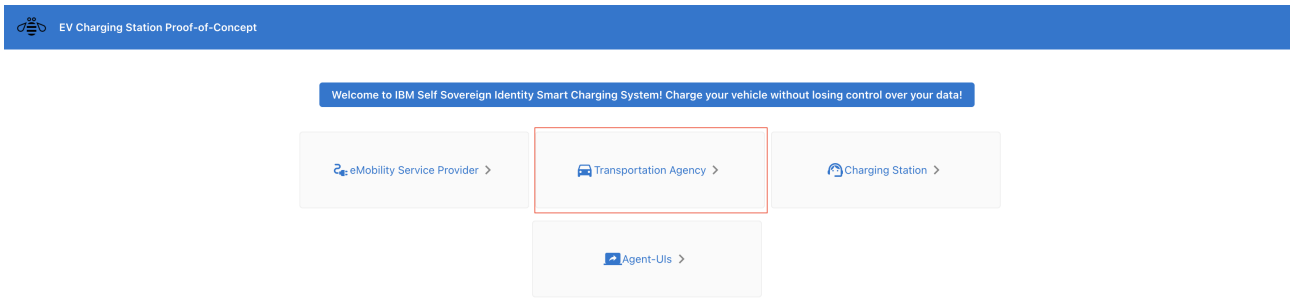

Figure 31. Selecting Transportation Agency portal

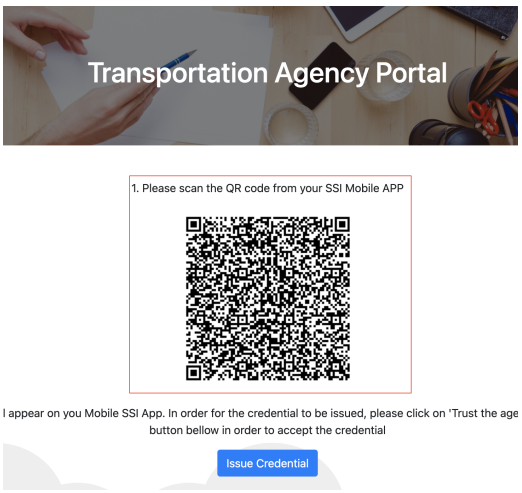

Figure 32. Connect to Transportation Agency Agent using QR Code

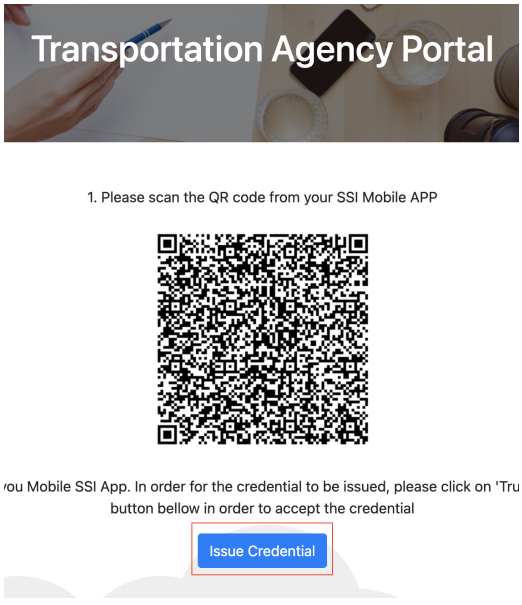

Figure 33. Press the "Issue Credential" button

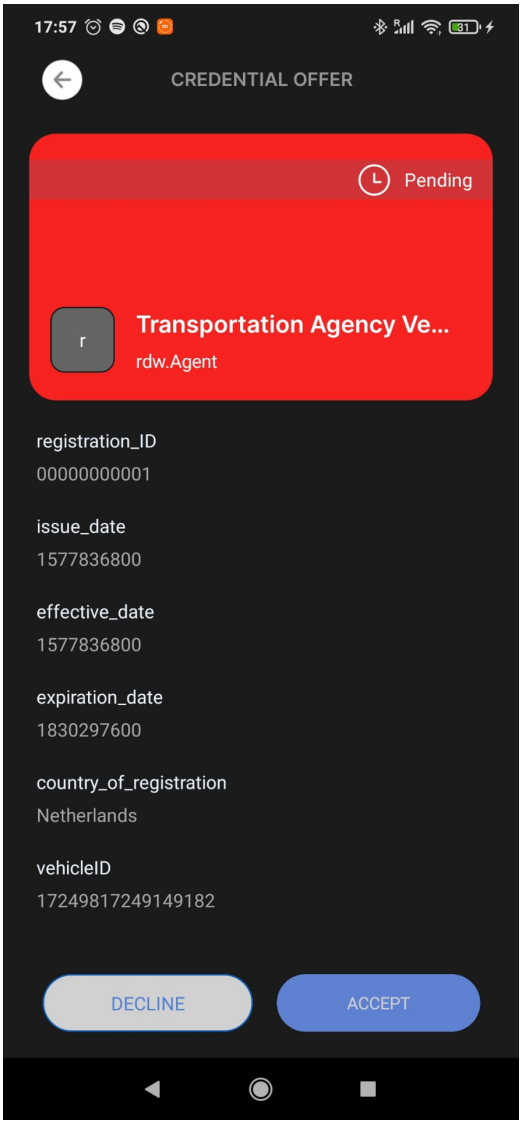

Figure 34. Accept the credential on the Trinsic Wallet App

B.3 Charging and Billing with SSI Screenshots

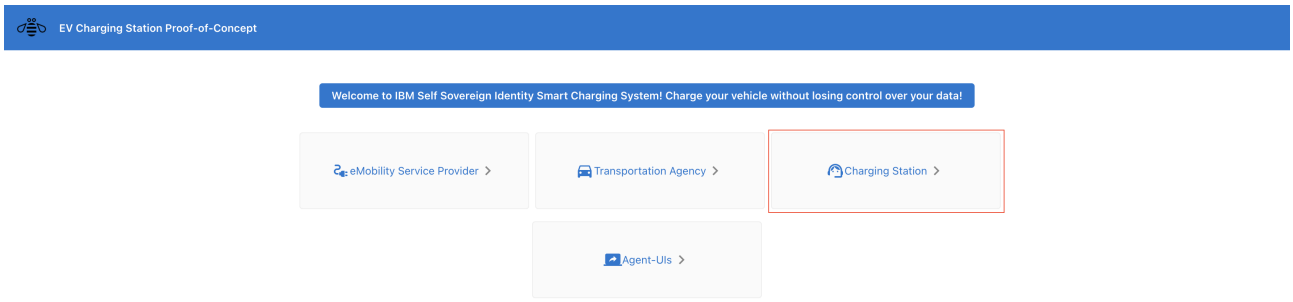

Figure 35. Selecting Charging Station dashboard

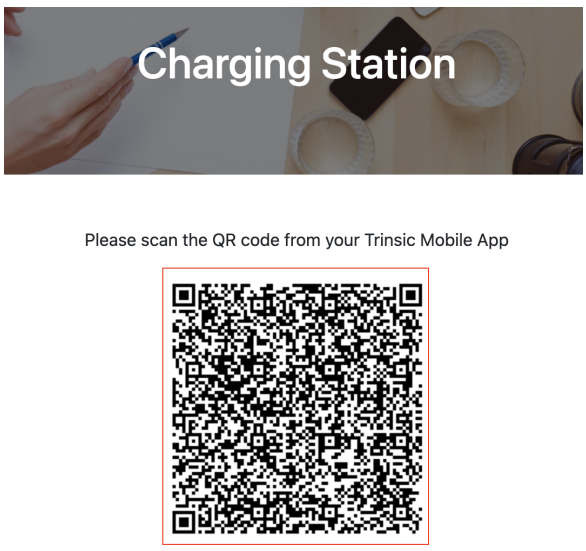

Figure 36. Connect to Charging Station Agent using QRCode

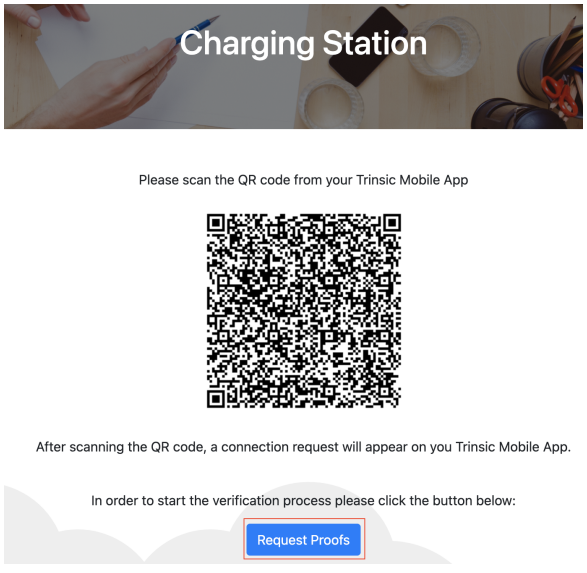

Figure 37. Press the "Request Proofs" button

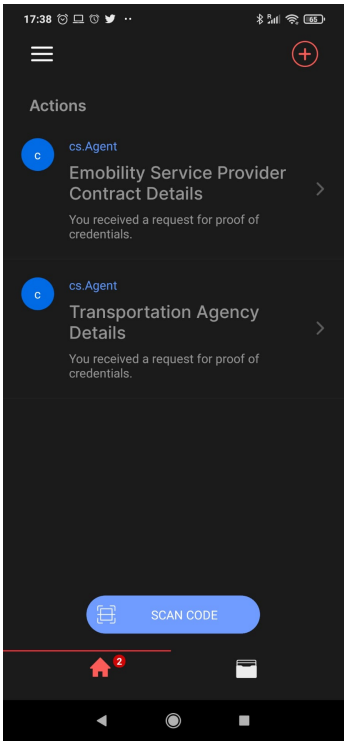

Figure 38. Receive Presentation Proof Requests on Trinsic Wallet App

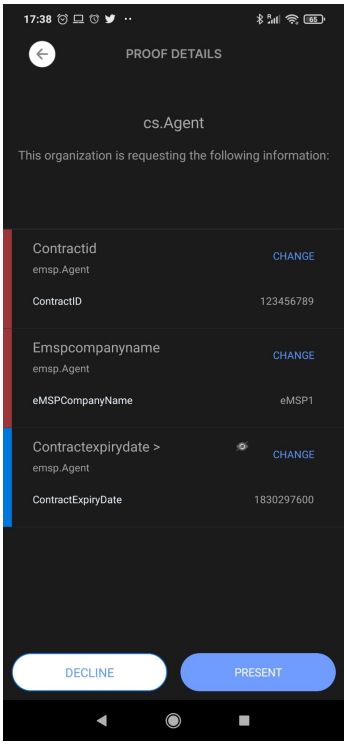

Figure 39. Present the proof for the TA-issued credential

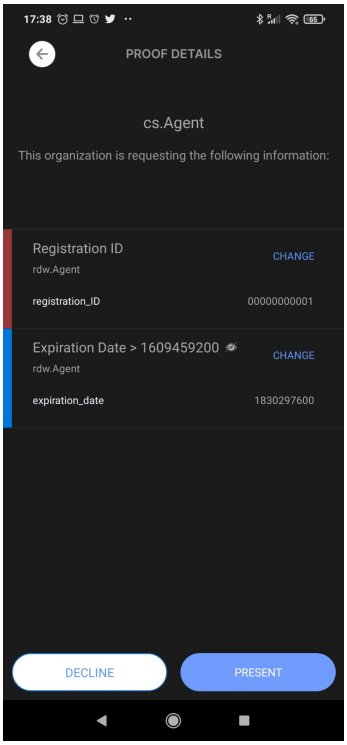

Figure 40. Present the proof for the eMSP-issued credential

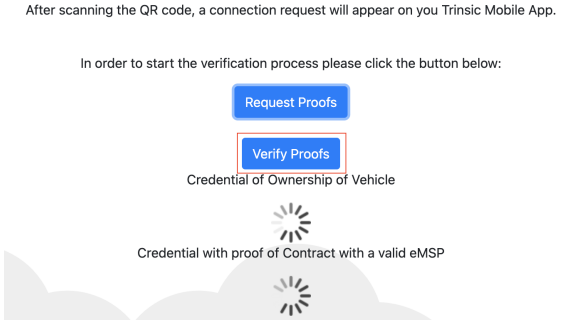

Figure 41. Press the "Verify Proofs" button

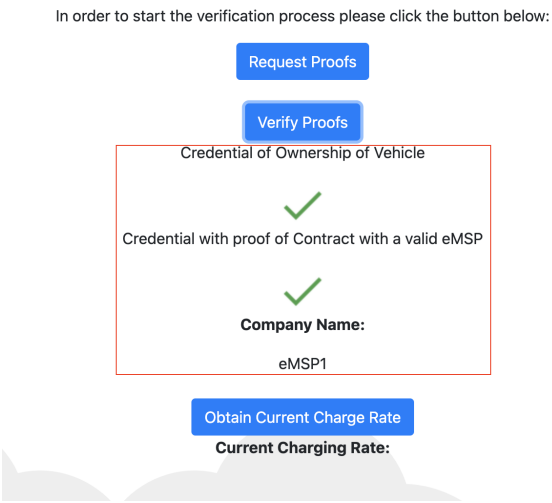

Figure 42. Verify that both credentials are valid on the dashboard

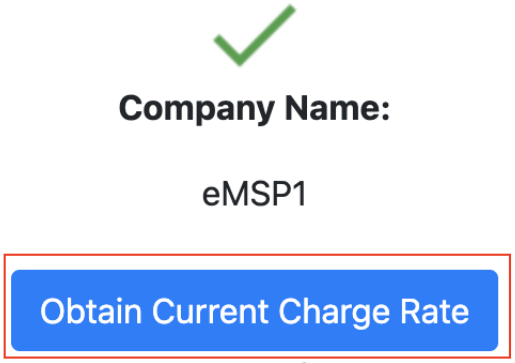

Figure 43. Press the "Obtain Current Charge Rate" button

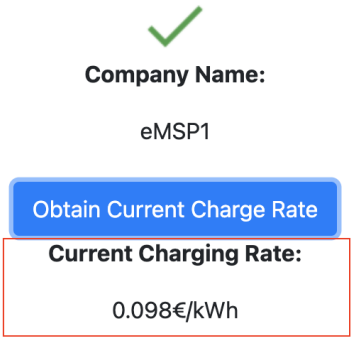

Figure 44. Verify the kWh price on the dashboard

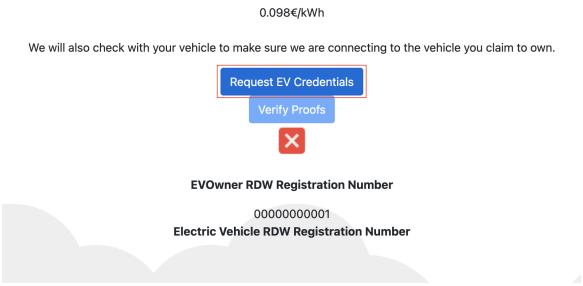

Figure 45. Press the "Request EV Credentials" button

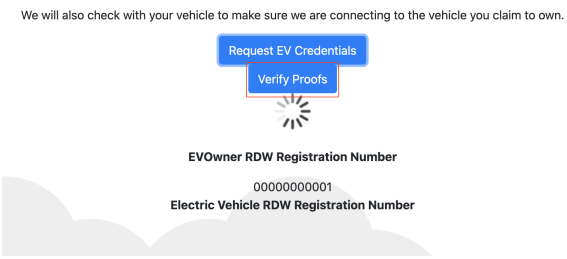

Figure 46. Press the "Verify Proofs" button

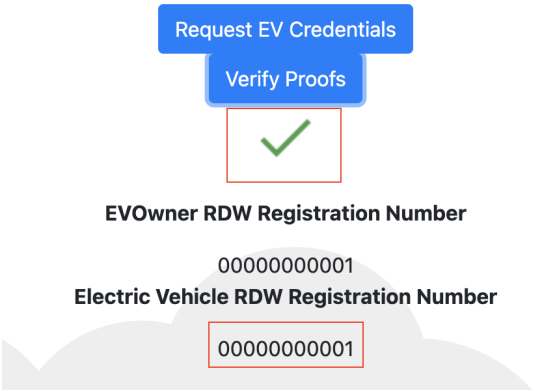

Figure 47. Verify that the credential is valid on the dashboard and that the Registration Numbers match

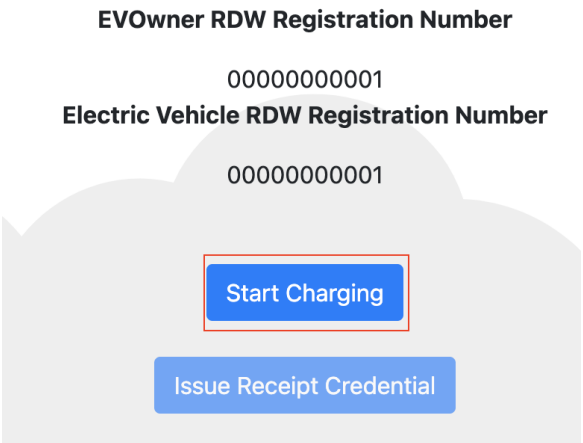

Figure 48. Press the "Start Charging" button

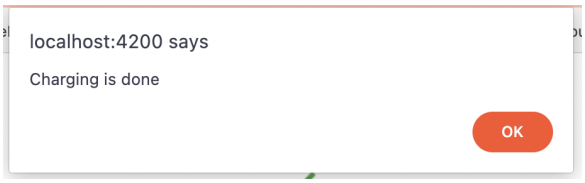

Figure 49. Verify that charging has occurred on the dashboard

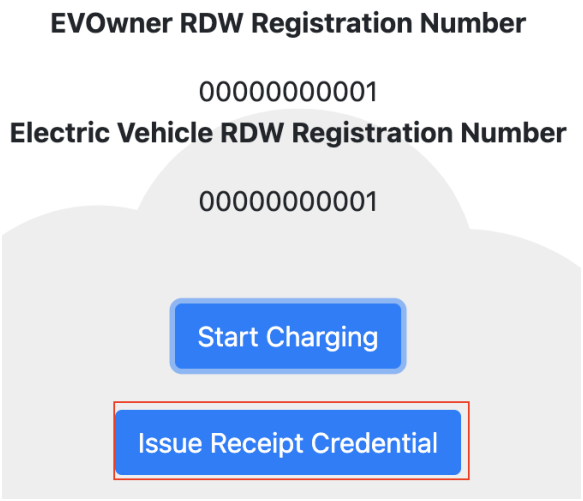

Figure 50. Press the "Issue Receipt Credential" button

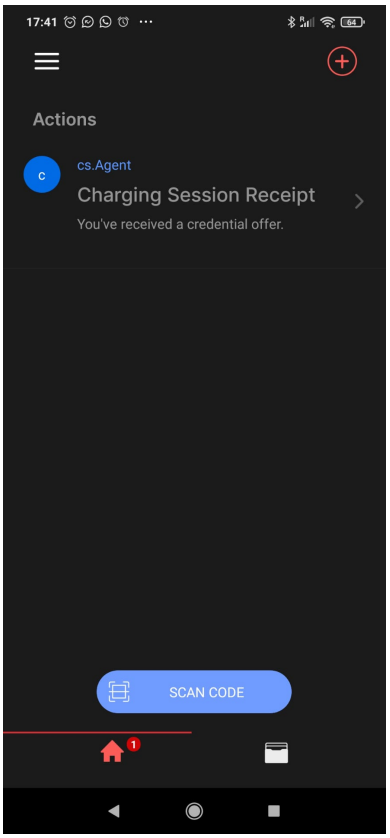

Figure 51. Receive the credential proposal on the Trinsic Wallet App

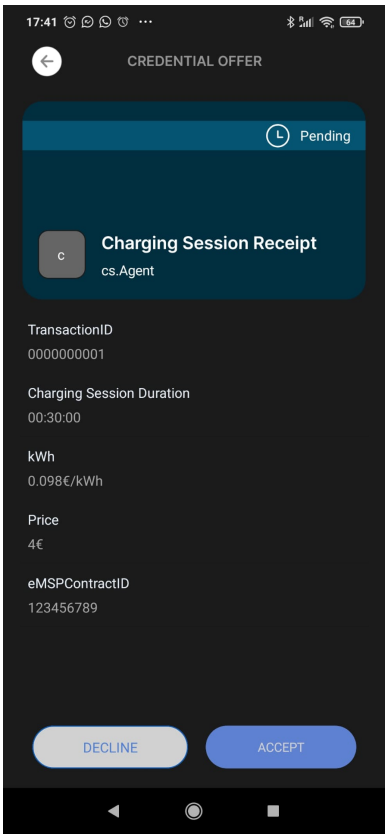

Figure 52. Accept the credential on the Trinsic Wallet App

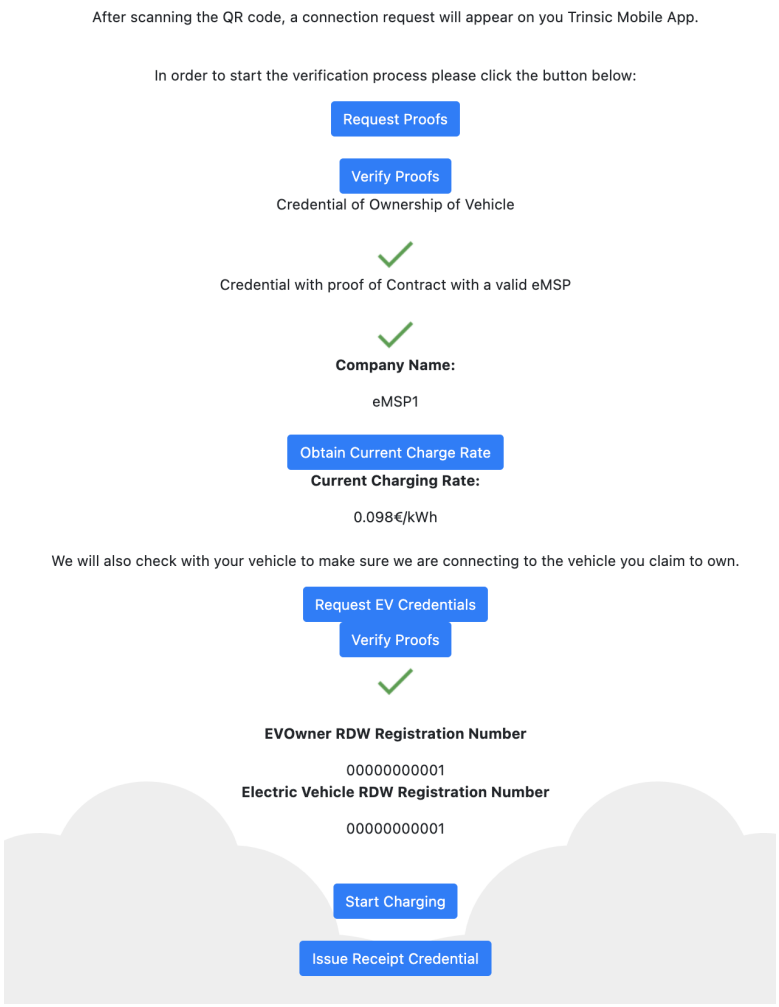

Figure 53. State of the dashboard at the end of a charging session

C Aries Cloud Agent - Python Benchmark Runs

In this appendix the commands used to obtain the average time of credentials are listed, followed by a small paragraph with additional information on connection times, revocation times, time to publish a schema and a credential definition.

- *./demo/run_demo performance –count 1000 –mediation 2>&1 | tee no_revocation_mediation_1000.txt*
- *./demo/run_demo performance –count 1000 –revocation –tails-server-base-url http://host.docker.internal:6543 2>&1 | tee revocation_no_mediation_1000.txt*
- *./demo/run_demo performance –count 1000 2>&1 | tee no_revocation_no_mediation_1000.txt*
- *./demo/run_demo performance –count 1000 –revocation –tails-server-base-url http://host.docker.internal:6543 –mediation 2>&1 | tee revocation_mediation_1000.txt*

All the tests were made using the following laptop specifications:

- Model Name: MacBook Pro
- Model Identifier: MacBookPro15,1
- Processor Name: 8-Core Intel Core i9
- Processor Speed: 2,3 GHz
- Number of Processors: 1
- Total Number of Cores: 8
- L2 Cache (per Core): 256 KB
- L3 Cache: 16 MB
- Hyper-Threading Technology: Enabled
- Memory: 16 GB
- System Firmware Version: 1554.100.64.0.0 (iBridge: 18.16.14556.0.0,0)
- Docker Version: 20.10.5

C.1 No Revocation + Mediation

- Startup duration: 15.14s
- Connect duration: 2.73s
- Publish duration: 14.46s
- Completed 1000 credential exchanges in 229.15s
- Average time per credential: 0.23s (4.36/s)

C.2 Revocation + No Mediation

- Startup duration: 8.06s
- Connect duration: 0.21s
- Publish duration: 21.83s
- Completed 1000 credential exchanges in 232.92s
- Average time per credential: 0.23s (4.29/s)

C.3 No Revocation + No Mediation

- Startup duration: 8.06s
- Connect duration: 0.19s
- Publish duration: 11.66s
- Completed 1000 credential exchanges in 229.45s
- Average time per credential: 0.23s (4.36/s)

C.4 Revocation + Mediation

- Startup duration: 15.68s
- Connect duration: 2.72s
- Publish duration: 10.01s
- Completed 1000 credential exchanges in 263.33s
- Average time per credential: 0.26s (3.80/s)

D Requirement Evaluation

In this appendix, all of the requirements are discussed, presenting a table containing the requirements description, priority, status, arguments to support or contest the requirement and, when applicable, an alternative is briefly introduced to the presented arguments.

FR-1.1	Priority
Priority	*Must*
Status	✔
Problem/issue	The system must allow for an EV Owner to charge its EV at a CS
Arguments	The system still carries the capabilities for a client to charge its Electric Vehicle.
Alternative(s)	-

Table 20. Evaluation of FR-1.1

FR-1.2	Priority
Priority	*Should*
Status	✔
Problem/issue	The rate at which the kWh is being sold to the EV Owner should be presented before the charging starts
Arguments	The current implementation of the system has the CS agent communicate to the CPO agent the company under which the client has a contract with, and receives a price rate back, displaying it to the user on the dashboard.
Alternative(s)	The price per kWh could be sent as a message to the user via the Trinsic Wallet App, instead of being presented on the screen.

Table 21. Evaluation of FR-1.2

FR-1.3	Priority
Priority	*Must*
Status	✔
Problem/issue	The EV Owner must be prompted to accept/reject the price of the kWh
Arguments	Although this is not actively implemented in the solution, given that the kWh price is presented to the user, it can refuse to charge the vehicle by walking out of the latter
Alternative(s)	In the future it would be interesting to have the user have an "accept" or "refuse" button, but that would not change the flow of the system

Table 22. Evaluation of FR-1.3

FR-1.4	Priority
Priority	*Could*
Status	✔
Problem/issue	At the end of a transaction, the EV Owner could receive proof of the transaction with the amount due
Arguments	In the new implementation of the system, the EV Owner receives a credential on his Trinsic Wallet App agent that contains information regarding the charging session.
Alternative(s)	-

Table 23. Evaluation of FR-1.4

FR-2.1	Priority
Priority	*Must*
Status	✔
Problem/issue	A CS must belong to a CPO
Arguments	The Charging Station holds a credential that attests it is owned by the Charging Point Operator.
Alternative(s)	-

Table 24. Evaluation of FR-2.1

FR-2.2	Priority
Priority	*Must*
Status	✔
Problem/issue	A CPO must have a contract with an eMSP
Arguments	The CPO agent holds a credential that attests a contract with an eMSP.
Alternative(s)	-

Table 25. Evaluation of FR-2.2

FR-2.3	Priority
Priority	*Must*
Status	✔
Problem/issue	A CPO must have a contract with an EP to provide power to its Charging Stations
Arguments	The CPO agent holds a credential that attests a contract with an EP.
Alternative(s)	-

Table 26. Evaluation of FR-2.3

FR-3.1	Priority
Priority	*Should*
Status	✔
Problem/issue	EV Owners should prove they are the owners of their EV to a CS
Arguments	In the current implementation of the system, the EV Owner and the EV both have a credential provided by the Transportation Agency that links both by a registrationID, as seen in Section 4.3.3.
Alternative(s)	The fact that the link between the EV and the EV Owner is made with the use of a unique identifier might impose privacy-damaging practices, that need to be evaluated at a later stage.

Table 27. Evaluation of FR-3.1

FR-3.2	Priority
Priority	*Must*
Status	✔
Problem/issue	EV Owners must attest they have a valid contract with an eMSP that the CS's CPO has a contract with
Arguments	The EV Owner is given a credential from the eMSP that attests the contract between the two, and is presented at a Charging Station for verification.
Alternative(s)	-

Table 28. Evaluation of FR-3.2

FR-3.3	Priority
Priority	*Could*
Status	✗
Problem/issue	An EV Owner could be made aware that the CS belongs to the CPO it claims
Arguments	Although the Charging Station holds a credential that attests it is owned by a specific CPO, that credential has not been included in the proof-of-concept, but could be added easily later on, with the support of a better mobile agent.
Alternative(s)	-

Table 29. Evaluation of FR-3.3

FR-3.4	Priority
Priority	*Should*
Status	✔
Problem/issue	The process of setting up the connection and verifying credentials should be executed with the least EV Owner intervention as possible.
Arguments	The process of establishing a connection and verifying credentials has been reduced to a maximum of steps that abide to all of the requirements and does not incur in problem for the users, according to the data extracted from the experts evaluation in Section 6.3.
Alternative(s)	-

Table 30. Evaluation of FR-3.4

FR-4.1	Priority
Priority	*Must*
Status	✔
Problem/issue	An EV and the information regarding its EV Owner must be registered with the TA in order for the EV to be on the roads and to be charged at a CS
Arguments	The EV and its EV Owner are registered at the Transportation Agency and are provided with credentials that attest these claims.
Alternative(s)	-

Table 31. Evaluation of FR-4.1

FR-5.1	Priority
Priority	*Must*
Status	✔
Problem/issue	The system must not write PII in a publicly accessible data repository
Arguments	With the usage of the Hyperledger Indy and the Hyperledger Aries ecosystems, it is guaranteed that no Personally Identifiable Information is written on the blockchain, and that the data is only stored in wallets held by each entity as seen in Section 2.2.8.
Alternative(s)	-

Table 32. Evaluation of FR-5.1

FR-5.2	Priority
Priority	*Must*
Status	✔
Problem/issue	Every credential issued must be verifiable using cryptographic proof
Arguments	With the cryptographic capabilities provided by Hyperledger Ursa to the solution, it is guaranteed that this solution employs state of the art cryptography practices, as seen in Section 2.4.1.
Alternative(s)	-

Table 33. Evaluation of FR-5.2

FR-6.1	Priority
Priority	*Must*
Status	✗
Problem/issue	An EV Owner must be able to delegate (temporary) possession of its EV to another driver
Arguments	In the current implementation of the ACA-Py agents, there is a work in progress to implement delegated credentials, which will mitigate this requirement in a near future.
Alternative(s)	Issue a credential to each individual that will be the driver of the vehicle, which is impractical in real-life scenarios.

Table 34. Evaluation of FR-6.1

NFR-1.1	Priority
Priority	*Must*
Status	-
Problem/issue	The system must scale according to the growth of the market. Supposing an adoption in the Dutch Market (300k EVs in 2020), the system must be able to support this number of vehicles and the estimated growth (3M EVs by 2030)
Arguments	This requirement can be validated with the results obtained in Section 6.2 since the Sovrin network is able to withstand the current load without a loss of information. But for this requirement to be validated more tests need to be performed to the network to adjust the number of nodes needed to withstand such system scale.
Alternative(s)	-

Table 35. Evaluation of NFR-1.1

NFR-2.1	Priority
Priority	*Should*
Status	✔
Problem/issue	The system should be able to kickstart the charging process in a reasonable amount of time (no longer than the time required without VCs, less than 1 minute)
Arguments	The system has been timed in the charging station dashboard, and the process of charging could be started with less than 1 minute, depending on the user expertise with the technology. From the experts evaluation, the user experience has not been affected according to the results on Section 6.3
Alternative(s)	-

Table 36. Evaluation of NFR-2.1

NFR-2.2	Priority
Priority	*Must*
Status	✔
Problem/issue	The system must withstand multiple users accessing the system at the same time, with no loss of information.
Arguments	This requirement is assessed on a theoretical level, since no tests were made with concurrent agents performing the same flow. But, the capability of the system to handle concurrent users depends on the Hyperledger Indy network and not the agents, since each charging station will have one agent each, and therefore only have one session happening at the same time, which has been tested and proven to work. The network can withstand multiple agents writing at the same time as seen by the results on Section 6.2.
Alternative(s)	-

Table 37. Evaluation of NFR-2.2

NFR-2.3	Priority
Priority	*Must*
Status	✔
Problem/issue	The user must be presented with information on how to engage with the CS in less than 5 seconds.
Arguments	As soon as the user arrives at the Charging Station, a call is made to the Charging Station agent to generate an invitation for the EV Owner to connect. This action occurs asynchronously and takes less than 2 seconds to appear.
Alternative(s)	-

Table 38. Evaluation of NFR-2.3

NFR-2.4	Priority
Priority	*Should*
Status	✔
Problem/issue	The process of validation the client's credentials should not take longer than the original method (designed for 2 seconds, with the worst case scenario being 30 seconds)
Arguments	Following the work made on Section 6.2 regarding the benchmarks of the ACA-Py agents and the results, this requirement can be marked as fulfilled.
Alternative(s)	-

Table 39. Evaluation of NFR-2.4

NFR-3.1	Priority
Priority	*Should*
Status	✔
Problem/issue	The system should be constantly updated to mitigate the risk of quantum-attacks, by employing state-of-the art cryptography
Arguments	The current implementation of the Hyperledger Indy and Aries frameworks are based on another project from the Hyperledger ecosystem, Ursa. This project is the cryptographic core of both projects and is constantly updated to provide state-of-the-art security to both solutions.
Alternative(s)	-

Table 40. Evaluation of NFR-3.1

NFR-3.2	Priority
Priority	*Must*
Status	**-**
Problem/issue	The system must comply with Dutch regulations, including GDPR
Arguments	Although blockchain and the information that gets written to the DLT, as discussed in Section 2.2.8, do provide GDPR compliance, two matters on the solution do not. The two latter have been assessed in the Limitations chapter (Section 6.4.3) and hence the requirement has been marked with **"-"**.
Alternative(s)	Discussed in the Limitations chapter.

Table 41. Evaluation of NFR-3.2

NFR-3.3	Priority
Priority	*Must*
Status	✔
Problem/issue	The system must be protected against forgery of identity, so that in the case of theft of identity the malicious party cannot use the system
Arguments	The core principals of SSI and the Verifiable Credentials allow to comply with this requirement, since each credential is cryptographically signed and there is no way to tamper this information or forge it, unless a wallet is taken hostage by a malicious party. But even in these circumstances, biometrics could be used as a Two-Factor Authentication method.
Alternative(s)	-

Table 42. Evaluation of NFR-3.3

NFR-4.1	Priority
Priority	*Must*
Status	✔
Problem/issue	A CS must have an active internet connection to authenticate the EV Owner
Arguments	Although not actively evaluated, in Section 6.4.2 a discussion is conducted to why this requirement can be fulfilled if applied in the Netherlands.
Alternative(s)	Perform more research to assess the current state of charging station connectivity in more regions where this solution might be applied.

Table 43. Evaluation of NFR-4.1

NFR-4.2	Priority
Priority	*Must*
Status	✔
Problem/issue	The systems implementation must allow for as many agent implementations as possible, opting for solutions that favor interoperability
Arguments	This requirement is fulfilled since the goal of Hyperledger Indy and Aries is to provide an interoperable ecosystem that is not vendor-locked. Any agent that complies with the current Verifiable Credentials standards is capable of interacting with the Hyperledger Indy network, and that is why this requirement is fulfilled.
Alternative(s)	-

Table 44. Evaluation of NFR-4.2

NFR-4.3	Priority
Priority	*Must*
Status	✔
Problem/issue	The system must offer a 97% uptime guarantee
Arguments	Following the evaluation performed on the Hyperledger Indy network in Section 6.2, the current best example of an Indy network has more than 97% uptime, which leads to believe that it is capable of handling this availability requirements.
Alternative(s)	-

Table 45. Evaluation of NFR-4.3