MSE | MASTER OF SCIENCE IN ENGINEERING

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts Western Switzerland

Master of Science HES-SO in Engineering
Av. de Provence 6
CH-1007 Lausanne

Master of Science HES-SO in Engineering

Communication and Information Technology

Application of Self-Sovereign Identity in the Physical World

Interaction between Peers with a Mobile Wallet

Written by

Michel Sahli

Under the supervision of
Prof Sylvain Pasini
HEIG-VD

In collaboration with
Adnovum Informatik AG

Version 1.0

17.02.2022

adnovum

Contents

List of Figures

1 Abstract

Self Sovereign Identity (SSI), a new generation of digital identity, is emerging and experiencing hype around the globe. It gives privacy and control over personal data back to its users, who will be able to store their identity on their smartphone and use it instead its analog counterpart.

Governments and private companies hope to find in it a technology that will enable long overdue digization of analog documents such as ID cards, passports, driver's licenses, etc. Digitization of verification and official identity digitalization are one of the last steps toward greater automation and simplification of processes. Applications to the government will be possible without having to go to a counter, and services requiring residency or identity verification could be simplified without adding multiple days of delay.

SSI as a whole is still immature and needs work on its standards and implementation. Current contributors are focused on the online experience of how to integrate SSI into their services.

This thesis presents a prototype Android mobile wallet based on the latest SSI standards and the highest security standards among SSI wallets, integrating SSI into the physical world. Such an integration would enable the use of SSI in daily interactions. SSI users would thus be able to share verifiable claims about themselves with individuals or machines. As a result, SSI users would, for instance, be able to leave their analog wallets at home and rely on SSI to prove claims about their identity, their capacity to drive and so on. Additionally, the integration of SSI into the physical world would limit the amount of information that is shared by verifying only selected information rather than, for instance, a complete ID card. The exposure of personal information would thus be minimized.

This applied research paper explains the SSI core standards and proposes a complete solution to implement SSI into the digital and physical world. Finally, it implements an interactive mobile application that can be used to prove verifiable claims to other mobile applications.

2 Specifications

This chapter explains the context of this thesis, then defines its tasks and objectives and lastly outlines the initial planning.

2.1 Context

Over the past decade, peoples's perception of security and privacy has evolved to become more critical of the type of data service providers collect about them. In recent years, legislators in the EU and in Switzerland strengthened the protection of personal data from third party harvesting by introducing the General Data Protection Regulation (GDPR) or amending the Federal Act on Data Protection (FADP) to contribute to the protection of individual's data. That being said, such laws do not stop centralized Identity Providers (IdPs) from correlating users' behavior and using their personal data in a verifiable and traceable manner. It is usually a difficult task for the average population to assess the security and privacy standard of online services. In most of the cases, users rely on trust without any relevant evidence.

In recent years, a new concept has emerged that puts users back in control of their data. Instead of relying on a centralized IdP, the users themselves become their own IdP. They manage and prove their digital identity and have the full control over it. This new concept is called Self Sovereign Identity (SSI) and is based on the interaction of three key entities:

- Holder: User who holds onto their personal verifiable data and can prove its authenticity to others.
- Issuer: Authority that issues authentic and verifiable data to Holders.
- Verifier: Entity that can verify the authenticity of the Holder's verifiable data.

Holders are at the center of SSI solutions. They have a mobile wallet application on their smartphone, which is the entry point to the SSI ecosystem. Their digital identity is stored in it, and the Holders control what data is shared with others. Trusted Issuers have to verify the Holder's authenticity and can issue the digital identity to the respective Holder. Verifiers may be online services, Holders or even Issuers and can verify the authenticity of the digital identity received from Holders (see Chapter 3.2).

For instance, a seller of age-restricted goods must verify that the users are old enough to buy such goods using as little personal information as possible. The features of today's mobile wallets are mostly restricted to identity storage, passwordless authentication with online services and in some cases secure chats between peers. Online services must also have an SSI wallet; these are called cloud wallets. Cloud wallets differ from mobile wallets in that they are running on a publicly reachable server rather than on a smartphone.

2.2 Objectives

The main objective of this thesis is to develop a mobile wallet application based on SSI solutions. The mobile wallet application thus includes the basic features to authenticate itself, receive digital identity fragments and verify validated personal data between people in the physical world.

As a result, such a mobile wallet application enables its users to prove their verifiable identity tokens between peers without a central service provider.

2.3 Tasks

The development of a mobile wallet application may be divided into the following tasks:

- Analysis of the concept of Self-Sovereign Identity and accumulation of Know-How.
- Analysis and definition of wallet application requirements with mobile-to-mobile presentation of proof.
- Conception of the wallet application with a security and privacy by design approach.
- Implementation of the wallet application with the physical presentation of proof feature.
- Evaluation and test of the wallet application.

This thesis could also contain the following points if time permits:

- (Optional) Development of additional Wallet features, for example:
  - Secure messaging between peers

Important infrastructure components required by the wallet application (for example to issue digital identities), will be provided by an internal team.

2.4 Planning

The Gantt diagram in Figure 1 shows the division of work over the whole duration of the project.

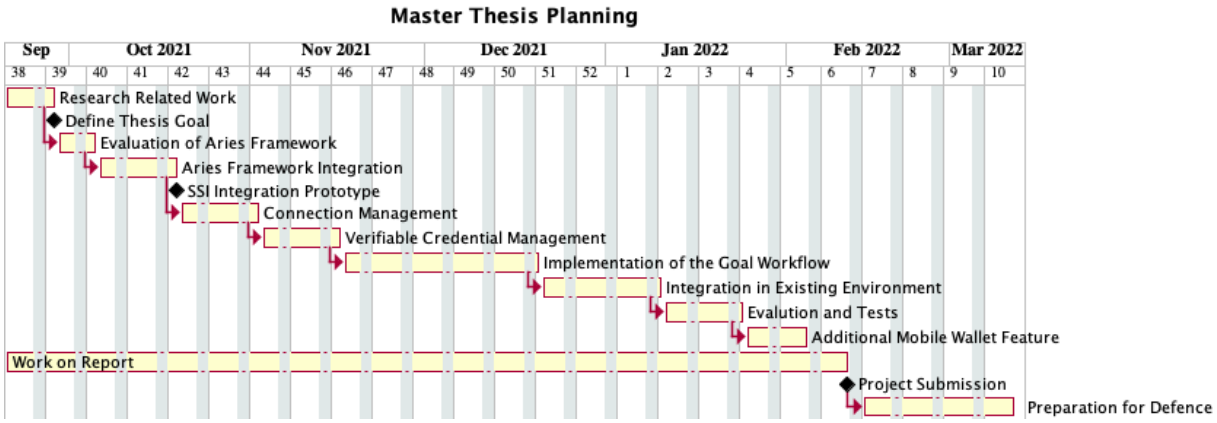

Figure 1: Planning

2.5 Document Structure

This document involves the following nine chapters:

Chapter Name	Description
Introduction To SSI	Explanation about the base SSI components necessary to understand the thesis
Background and Related Work	Research on SSI related standards and available mobile wallets
Use Case	Detailed use case definition and real-world examples
Analysis	Theorical analysis of compatible SSI components to implement the use case
Conception	Comparison of existing SSI frameworks to fulfill and build a mobile wallet
Implementation	Description of structural implementation of the mobile wallet implementation
Issues	Collection of issues encountered during the master thesis
Evaluation	Evaluation of the mobile wallet implementation and future improvements
Conclusion	Conclusion of the Master thesis results and further work

3 Introduction to SSI and Digital Identity

In the early days of interconnected computers, nobody knew who you were and what you were connecting to. As more people started to use IT systems, it became clear that users needed to be identified. Various methods and protocols were used, without any success, to patch an identity layer over the existing infrastructure.

3.1 History of Digital Identity

Kim Cameron defined seven hypotheses in his publication "The Laws of Identity"[1] that systems should implement to achieve a future-proof identity layer:

1. *Identity systems must only reveal information identifying a user with the user's consent.*
2. *The solution which discloses the least amount of identifying information and best limits its use is the most stable long-term solution.*
3. *Digital identity systems must be designed so the disclosure of identifying information is limited to parties having a justifiable necessity.*
4. *A universal identity system must support both discoverable identifiers for use by public entities and private identifiers for use between private entities.*
5. *A universal identity system must channel and enable the interoperability of multiple identity providers.*
6. *The universal identity metasystem must define the human user to be a component of the distributed system integrated through unambiguous human-machine communication mechanisms offering protection against identity attacks.*
7. *The unifying identity metasystem must guarantee its users a simple, consistent experience while enabling separation of contexts through multiple operators and technologies.*

Based on these rules, the representation of digital identity evolved throughout various phases:

An initial solution was for each digital service to create its own identity silos. Each digital service needed the user to create an identifier and had the authority over it. But it lacked in user controls and had poor data privacy concerns, as identity providers were able to collect and sell personal information. A trusted authority, such as an e-mail provider, could lock out a user and prevent them from using all of these systems linked to it. In addition, users had to manage dozens of identities (see Figure 2). A user needed a separate account for every online service they were using. For example, a user needed to create three different accounts with different security requirements to access their banking, insurances and online shopping services. Finally, the user's login credentials, which are often identically set for multiple services[2], could be stored in an insecure manner.

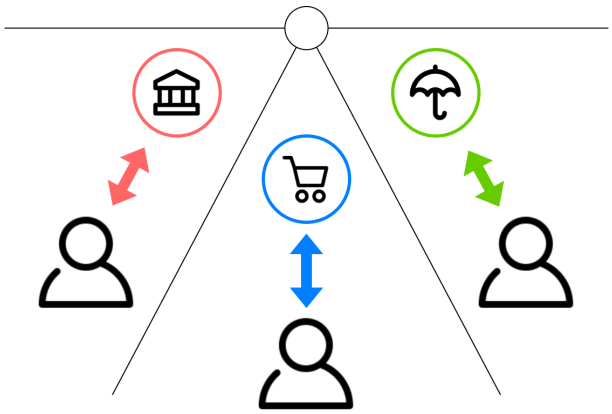

Figure 2: Centralized Identity Provider

The next advancement was the centralization of identities also known as federated identity providers. It resolved the issue to manage multiple identities by the users but it did not solve the problem of the power of the individual authorities which used these identities. As in Figure 3, each user has one account on the federated identity provider and can use it on all services that implement the identity provider.

It resolved the issue to memorize a large quantity of passwords and create individual accounts for each service. But it came at the cost of privacy. As every login goes through the federated identity provider, it became possible to collect personal data but also to correlate the online behavior of their users.

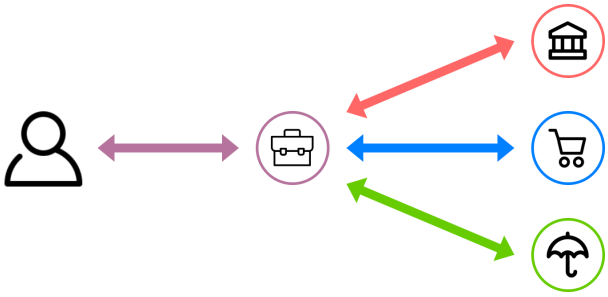

Figure 3: Federated Identity Provider

In a next phase, it was mostly focused to make the digital identity more user-centric. Users could choose with whom they share their identity and make the whole process more interoperable. Multiple new protocols like OAuth[1], OpenID Connect (OIDC)[2], and FIDO[3] were defined to greatly increase integration between systems. But this phase failed to give more control to the user and remained as centralized as

[1]OAuth is the industry-standard protocol for authorization
[2]OIDC is a simple identity layer on top of the OAuth 2.0 protocol.
[3]FIDO is a collection of open and free authentication standards to help reduce the world's reliance on passwords.

the last phase.

This leads us to the concept of SSI as the fourth phase as described by Christoph Allen in 2016[3]. The SSI concept is a recent user centric identity concept where digital identities belong to the user itself and they have authority over it. Users manage all their digital identities and use them to authenticate themselves on online services that also integrate SSI as in Figure 4.

The user does no longer have to worry about online behavior correlation and account creations as the identification is based on already existing digital identities trusted by the online services.

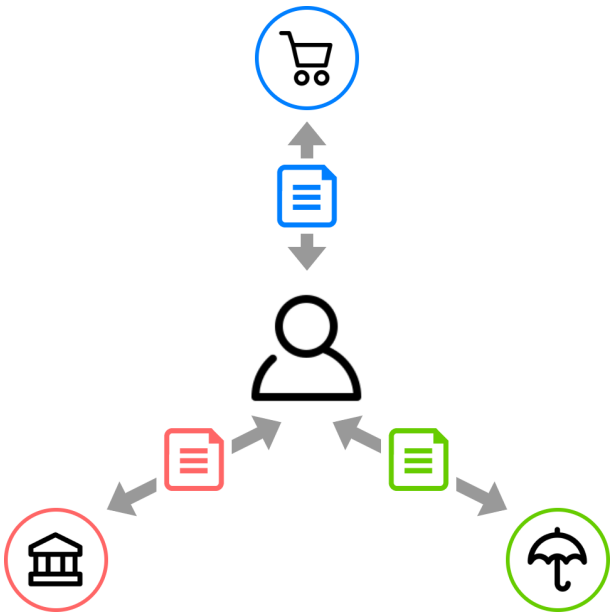

Figure 4: Decentralized Identity Provider

3.2 The SSI Concept

The SSI concept involves digital identities that the users can control themselves. Such digital identities are called Verifiable Credentials (VCs). Each user has their own collection of VCs and can prove their identities to others without using a third party. In Figure 5, Alice asks Bob for his name. In response, Bob searches for a VC with his name and sends it to Alice[4].

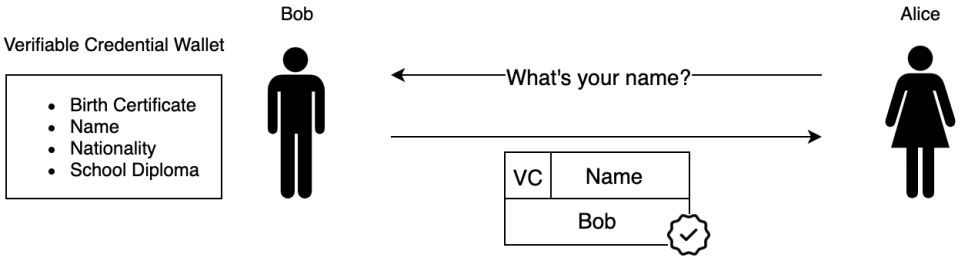

Figure 5: Bob Sharing His Name as Verifiable Credential

More generally speaking, entites called Issuers create VCs and guarantee their attributes. Issuers store and manage their public information and cryptographic keys for authenticity verification purposes on a decentralized registry (also called ledger) which is accessbile to all.

Users like Bob in Figure 5 receive VCs from Issuers. Bob can share the VCs with whomever he wants. Verifiers who receive Bob's VC can verify the authenticity of his identity themselves by requesting a cryptographic key from the decentralized registry and validate with it VC's proof signature. In other words, Verifiers do not need to contact the Issuer to verify Bob's credentials[4].

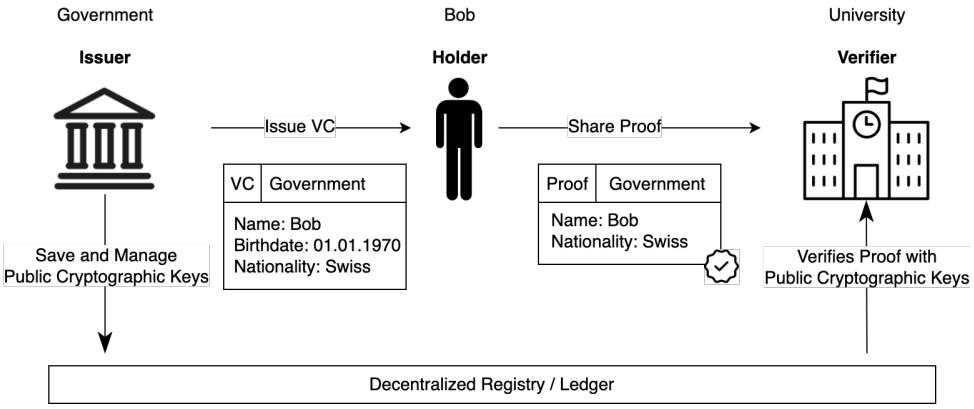

Figure 6: Bob Enrolling for University

In Figure 6, Bob receives a VC from the government which proves his name, birthdate and nationality. Bob can now enroll at a university with this digital identity. In such a case, the university might only needs their student's name and nationality. Bob can meet this request by sharing only selected attributes from his VC. The university can then verify the authenticity of the VC without having to contact the government.

More generally, people like Bob who receive VCs are called Holders. Holders are responsible for their VCs, and they can use them to prove their digital identity. Service providers, in this case the university can verify the VC from a holder and trust that the shared information is correct and reputable.

3.3 Why are Traditional IdPs Problematic

Traditional IdP and federated IdP are centralized and store the user's credential and personal information. It is easy for an IdP to correlate the user's behavior by analyzing the information that they have about them, including on which site they logged in.

IdPs rely on centralized storage of their user's data and are not safe from possible data leaks. Data leaks become all the more critical as not only the minimum amount of data, but most of the time the entire data collection is exposed (credit card, home adresse, social security number). As recently as June 2021, the e-mail addresses, geolocation, phone numbers, full names and more information of over 700 million LinkedIn users was collected[5].

Finally, when logging in over a federated IdP, the service provider has no guarantee that the information provided by the federated IdP is authentic and thus reliable.

3.4 Why Do We Need SSI

Relying solely on digital data is one of the bigger challenges that have not yet been solved. To prove one's identity, one would, for instance, scan an ID card or a driver's license. There is no immediate digital identity that could be used instead.

Today, many centralized platforms rely on the e-mail address and phone number their users provide during the registration. This method is fallible for two main reasons: first, the information provided might not be authentic; and second, an e-mail adress or phone number can be taken over during an attack using the recovery feature in combination with Sim Swapping[6].

VC solves these problems. The government can issue digital identities with the same value as hard copy identification documents to people who were identified in person or through other trusted identification processes.

Trust Over IP Foundation's website provides further information on how to establish trust over Internet and provides policies and technologies to communicate securely and confidentially[7].

SSI drastically reduces the problems of correlation, data collection and collective data theft that centralized system have. Users are using a unique identifier for each communication and store their own data. Their personal data is secured (with biometrics, PIN, etc) in the mobile wallet and cannot be used without it.

It is true that once the data is shared, it is exposed to the possibility of being collected. As opposed to traditional means, the user knows with whom they shared their information before it is made public.

3.5 Technical overview

SSI is a concept where multiple standards come together to propose a complete digital identity solution.

3.5.1 Decentralized Identifier

A Decentralized Identifier (DID) is an URI that contains a unique identifier as well as a method to resolve it to get more information about the holder of the DID:

```
did:example:123456789abcdefghi

[did]:[example]:[123456789abcdefghi]
|     |         |__ Unique identifier
|     |__ DID Method
|__ URI Schema
```

W3C proposes DID as standard[8]. Everyone can propose resolution methods[9].

As the number of methods increases, it becomes more difficult to implement each method resolution in the final solutions. That is why the Decentralized Identity Foundation (DIF) [10], a foundation that focuses on coordination of technical specification and implementation around the decentralized identity, created Universal Resolver. Universal Resolver takes the task to resolve the DID and respond with the tide information without implementing each method.

The next chapter introduces what information is associated with DIDs and what a user gets after resolving it.

3.5.2 DID Document

DIDs resolve to DID Documents (DID Docs), which are Json data structures that may contain, among other things, the following information:

- public endpoints (`service`) and cryptographic keys (`keyAgreement`) to create a secure connection, which will be explained in Chapter 3.5.3;
- public cryptographic keys to verify proof signatures of emitted VCs (`verificationMethod`);
- more information to prove the authenticity of the DID owner.

The example below shows a DID Doc from an Issuer. The service and key agreement information could be used to create a secure connection, and the verification method could be used to verify signature emitted by the Issuer.

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/jws-2020/v1"
  ],
    "service": [{
    "id": "did:example:123#didcomm-1",
    "type": "DIDCommMessaging",
    "serviceEndpoint": "http://example.com/path",
  }],
  "keyAgreement": [
    {
      "id": "did:example:123#
         zC9ByQ8aJs8vrNXyDhPHHNNMSHPcaSgNpjjsBYpMMjsTdS",
      "type": "X25519KeyAgreementKey2019",
      "controller": "did:example:123",
      "publicKeyBase58": "9hFgmPVfmBZwRvFEyniQDBkz9LmV7gDEqytWyGZLmDXE"
    }
  ],
  "verificationMethod": [
    {
      "id": "did:example:123#key-0",
      "type": "JsonWebKey2020",
      "controller": "did:example:123",
      "publicKeyJwk": {
        "kty": "OKP",
        "crv": "Ed25519",
        "x": "VCpo2LMLhn6iWku8MKvSLg2ZAoC-nlOyPVQaO3FxVeQ"
      }
    }
  ]
}
```

3.5.3 DID Communication

DID Communication (DID Comm) is the secure transportation layer based on DIDs and DID Docs proposed by DIF, which enables a secure end-to-end communication between multiple peers (see Figure 7). To achieve this, each peer needs to know an endpoint to which they can send their communication.

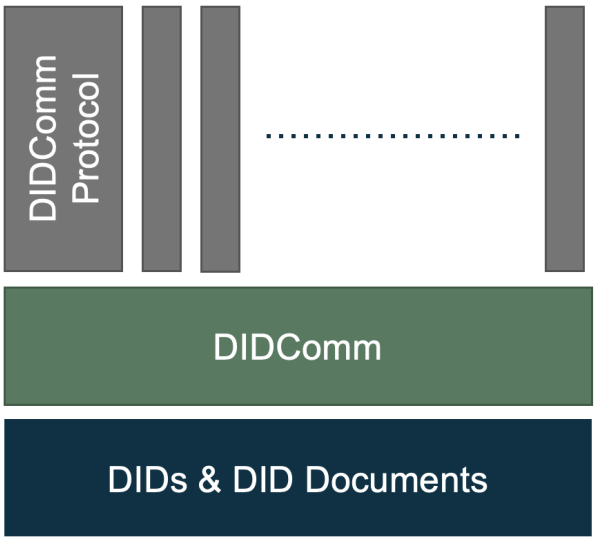

Figure 7: DIDComm Stack

Issuers have accessible public endpoints and public cryptographic keys stored in a DID Doc. These types of DID Docs can be stored in the decentralized registry, for example on Bitcoin[11] or Ethereum[12]. For private communications between individuals, the DID Docs are shared directly.

Each participant uses the public cryptographic key from the DID Doc key agreement to encrypt their messages before sending it to the other party. The other party can then decrypt it with their corresponding private cryptographic key and vice versa to send a response.

3.5.4 Verifiable Credentials

VCs are digital credentials that consist of machine-verifiable personal information that can be shared and verified in a secure and privacy-friendly way.

The VC data model is standardized by W3C[13] and can be used to represent physical documents such as degrees, healthcare related data and ID cards. These documents, which are usually difficult to verify and easily forgeable now are represented digitally and can be verified by anyone.

The following VC example below shows a representation of a degree. The VC's claims are found under the property `credentialSubject`, the VC's Issuer under `issuer` and the proof verification method as well as the proof itself under `proof`.

```
{
    "@context": [
        "https://www.w3.org/2018/credentials/v1",
        "https://www.w3.org/2018/credentials/examples/v1"
    ],
    "type": [
        "VerifiableCredential",
        "AlumniCredential"
    ],
    "id": "http://example.edu/credentials/1872",
    "credentialSubject": {
        "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
        "alumniOf": {
          "id": "did:example:c276e12ec21ebfeb1f712ebc6f1",
          "name": [{
            "value": "Example University",
            "lang": "en"
          }, {
            "value": "Exemple d'Université",
            "lang": "fr"
          }]
        }
    },
    "issuanceDate": "2010-01-01T19:73:24Z",
    "issuer": "https://example.edu/issuers/565049",
    "proof": {
        "type": "RsaSignature2018",
        "created": "2017-06-18T21:19:10Z",
        "proofPurpose": "assertionMethod",
        "verificationMethod": "https://example.edu/issuers/keys/1",
        "jws": "
           eyJhbGciOiJSUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19..
           TCYt5X
          sITJX1CxPCT8yAV-TVkIEq_PbChOMqsLfRoPsnsgw5WEuts01mq-
             pQy7UJiN5mgRxD-WUc
          X16dUEMGlv50aqzpqh4Qktb3rk-BuQy72IFLOqV0G_zS245-
             kronKb78cPN25DGlcTwLtj
          PAYuNzVBAh4vGHSrQyHUdBBPM"
      }
}
```

VCs play a central role in SSI as they represent the user's identity fragments. Users are free to share their identity partially or entirely with others in a verifiable manner[14].

3.6 Technical Representation of a DID Communication

To better illustrate the concept of DID Comm, Figure 6 shows a simplified SSI exchange. A more technical and closer representation of the same exchange is illustrated in Figure 8. The example assumes that the issuer uses a public DID to communicate and issue VCs.

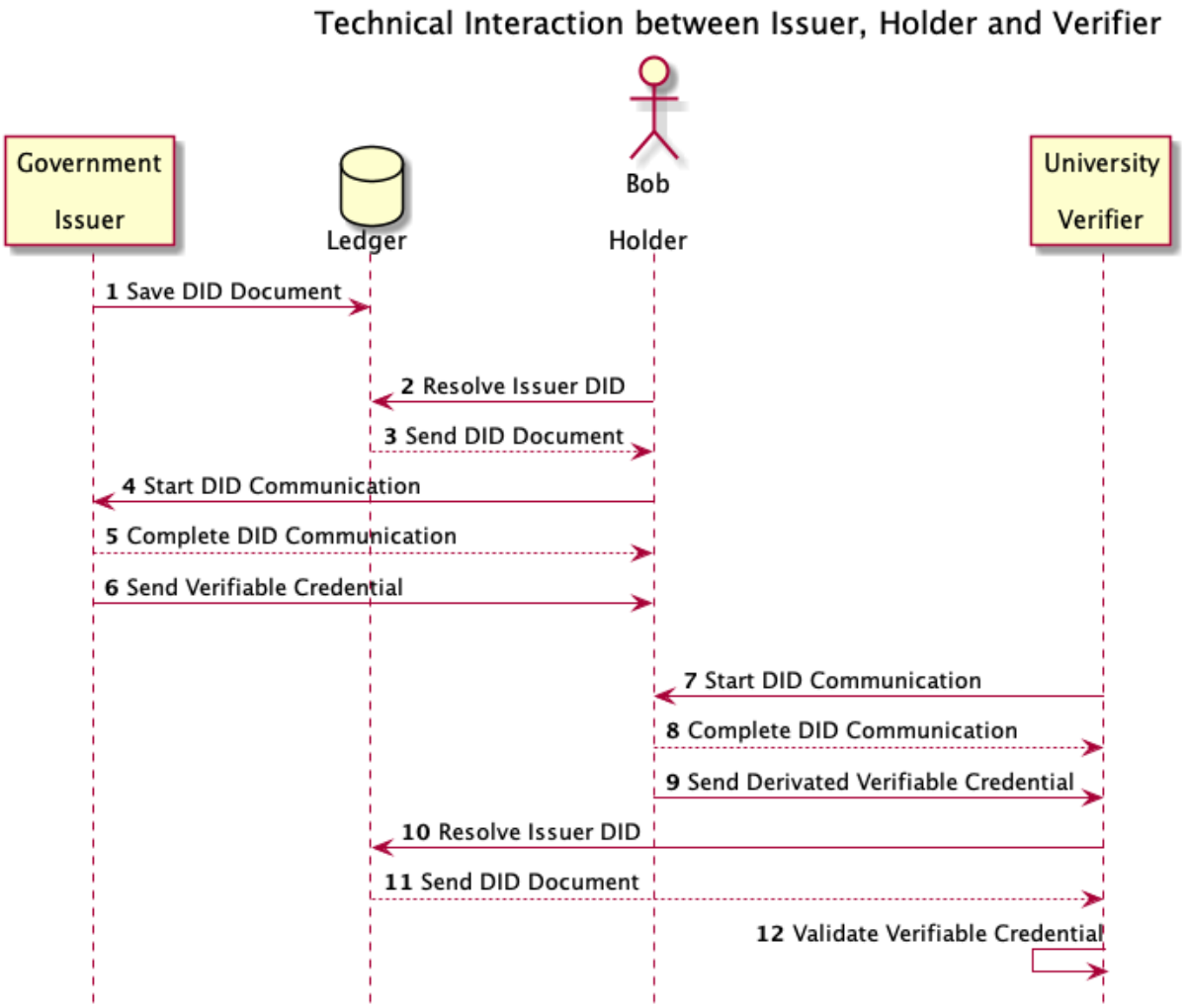

Figure 8: Bob Enrolling at University with DIDComm

1. Before the Issuer can issue VCs, they have to create a DID and save the corresponding DID Doc on a ledger. In this context, the DID Doc contains:
    - endpoint for other users to connect to (`service`);
    - cryptographic keys to encrypt the communication (`keyAgreements`);
    - cryptographic keys to verify the emitted VC proof signature (`verificationMethods`).

2. The Holder wants to connect to the Issuer and will ask the ledger to resolve the Issuer's DID.
3. The ledger sends back the Issuer's DID Doc to the Holder.
4. The Holder can initiate a DID Comm on the Issuer's public endpoint and will have to encrypt the communication with the key agreement cryptographic key. The Holder will also send its DID Doc because the Issuer could not resolve a private DID.
5. The Issuer decrypts the received communication with the corresponding private key and stores the Holder's DID Doc. They can send a response by encrypting the communication with the public cryptographic key from the received DID Doc.
6. The Issuer sends a VC that is signed with the private cryptographic key counterpart of the `verificationMethods` over the secure communication to the Holder.
7. Later on, the Verifier wants to validate the Holder's identity. As neither the Holder nor the Verifier have a public DID, the Verifier will need to send its DID Comm in plain text to the Holder. This is mostly done by way of a QR code.
8. The Holder scans the QR code and completes the DID Comm by using the scanned information to encrypt the communication to the Verifier and sending them the Holder's DID Doc.
9. The Holder sends the derived VC with only the necessary attributes to the Verifier over the secure communication.
10. The Verifier searches the received VC for the Issuer's public DID and asks the ledger to resolve it.
11. The ledger sends back the Issuer's DID Doc to the Verifier.
12. The Verifier uses the public verification cryptographic key to validate the VC and its signature received from the holder.

3.7 Conclusion

This chapter has explained the various historical phases of digital identity and the flaws that accompany them. The next step towards SSI brings many innovations and contrasts by transferring the responsibility for the entire identity management to the user to create a privacy-friendly, user-centric way of identification in the digital and physical world.

The SSI concept reunites multiple mature technical aspects in cryptography and (PKI) management to create a new set of common standards like DID, DID Doc, DID Comm and VC. These specifications will be the common base for an innovative way to manage digital identity.

4 Background and Related Work

This chapter analyzes available SSI solutions, and in particular the technologies that are already used and the application of SSI to the physical world. Based on this, gaps in existing SSI solutions will be identified.

4.1 Introduction to the Architectural Stack & Standards

There are several organizations currently defining standards in multiple SSI areas. The areas can be divided into three components that make up the architectural stack of SSI [15] (see left blue rectangle in Figure 9):

- Verified Credentials / Trusted Data Storage & Exchange
- DID Communication (+ subprotocols)
- DID Resolution (anchoring / registries)

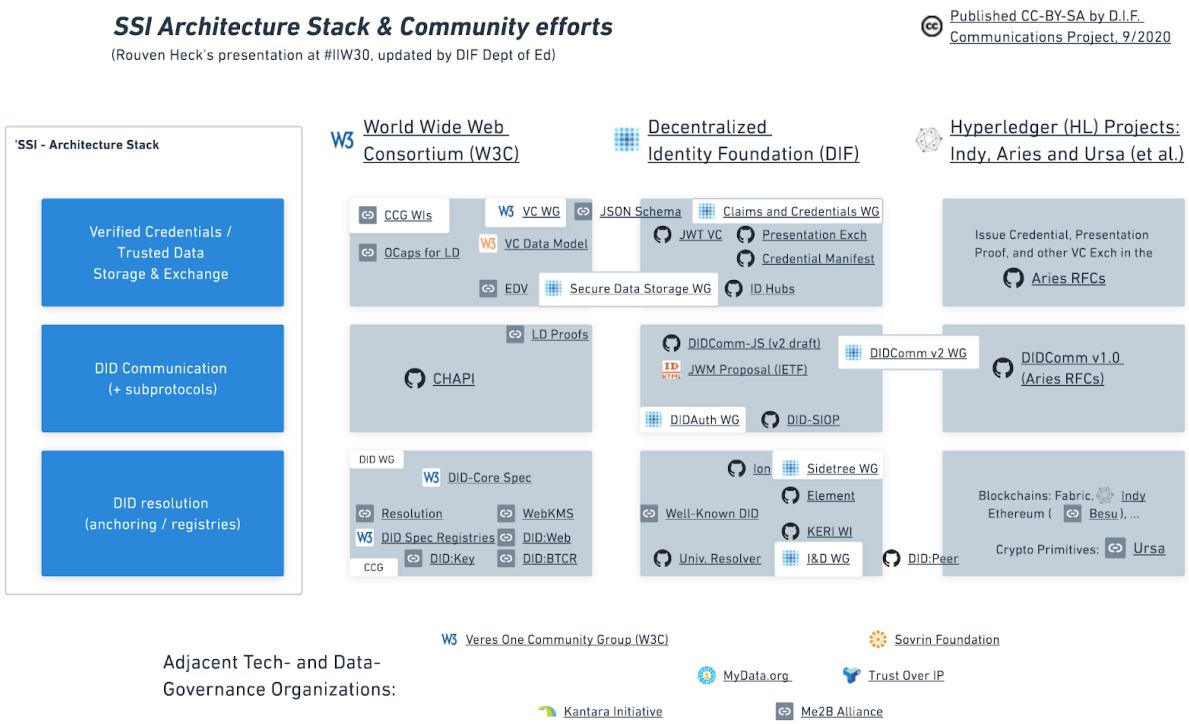

Figure 9: SSI Architectural Stack

Source: DIF Dept of Ed[16]

Each line in Figure 9 represents the standards written by the W3C, DIF or Hyperledger. The collection and interaction of these standards are what define SSI.

The base block of `DID Resolution` alone is necessary to represent how decentralized identifiers work and how critical information like public cryptographic keys have to be structured and resolved. Based on the information available from DIDs, it is possible to define and build a second block to securely establish a communication channel. All standards on that line define how to create an interoperable channel between participants, that do not necessarily support the same protocols. Finally, additional features and protocols can be added on top of the base components. With a secure channel, digital identity data structures, standards for secure storage and exchange between peers can be defined and implemented.

The history of the SSI standards and collaboration between the organization can best be illustrated in Figure 10.

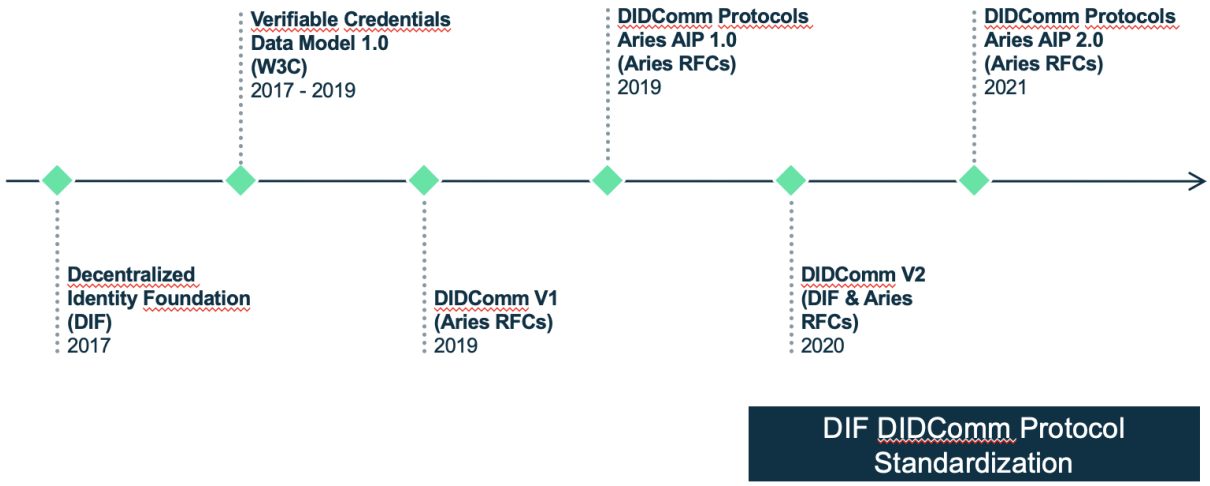

Figure 10: History of Standards proposed by DIF

DIF was created in 2017 to establish an open ecosystem for decentralized identity and ensure interoperability. Shortly after, a new standard for VC was elaborated by W3C. It is one of the first essential standards to represent user identities.

The Aries Project (see Chapter 4.3.1) then defined a secure transportation channel called DID Comm V1 to ensure the safety of the VC. Based on DID Comm V1, the Aries community also implemented various communication protocols. The collection of protocol definitions is called Aries RFCs and it includes protocols to create a DID Comm, exchange VC over a DID Comm, etc.

A year later, a second iteration of DID Comm was launched in collaboration between DIF and Aries. At the time of writing, this is slowly being implemented[17]. Starting 2021, a second iteration of the Aries

RFCs was proposed to update protocol definitions and add new ones. It did not mention DID Comm V2 initially but was integrated into the RFC after some months.

DIF also started to integrate some of the Aries RFCs protocols into the definition of DID Comm V2. Mostly critical protocols like the out-of-band invitation protocol were integrated to start a DID Comm [18].

4.2 Verifiable Credentials

The concept of verifiable credentials with selective attribute disclosure is not new and dates back to the first implementation in 2002[19]. These are called Anonymous Credentials and fit into the VC specification. They provide efficient privacy, protection and selective disclosure or even mathematical proof of the credentials' content (see Figure 11).

The mathematical proof, also called predicates, uses a Zero-Knowledge Proof (ZKP) Algorithm[20] to determine whetever or not a claim is true, based on the holder's credential. For example, it is possible to use predicates to verify if the Holder is within a certain age range without revealing their birthday.

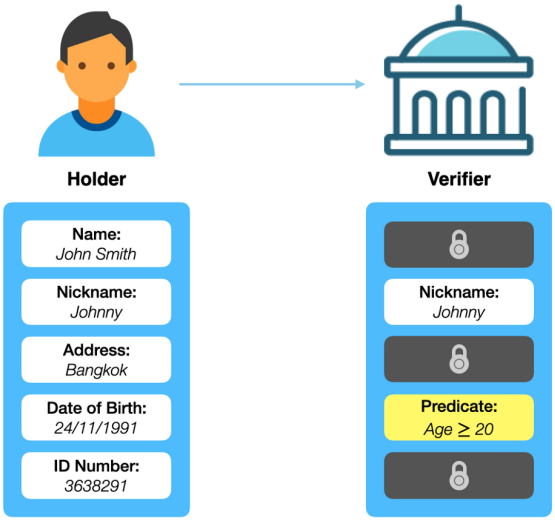

Figure 11: Selective Disclosure with Predicates
Source: Finema [19]

The first application for SSI platform was developed by Evernym and the Sovrin Foundation. It was later donated to the Hyperledger Foundation, which hosts a collection of open-source blockchain technologies. The project was then renamed Hyperledger Indy (more about Indy in Chapter 4.4.1).

VCs are referred to as Anonymous Credentials in Hyperledger Indy. It was the first verifiable credential implementation to be adopted for SSI Proof of Concepts (PoCs). It enabled ZKPs, but had the disadvantage that a credential definition had to be provided on a ledger, proof verification was relatively slow, and proof sizes were relatively large.

At the 31st Internet Identity Workshop (IIW) in Fall 2020, a new solution was announced by Mattr to resolve the previous problems[21]. This solution is based on JSON-LD with BBS+ signature. The first iteration did not support ZKP [22]. The focus of the format was to eliminate the credential definition on the ledger and create smaller proof footprints as well as faster proof verification[23].

The draft for the new solution is beeing submitted on W3C and work is currently underway to add ZKPs [24].

4.3 DIDComm and its Protocols

The first version of DID Comm was implemented as a software RFC rather than a specification. It was created as part of the project Hyperledger Aries, joint development project, which aimed to create SSI frameworks to reduce the complexity of DID Comm and the Aries RFCs. DID Comm evolved through increasing use case context and implementation supported by Aries[25].

Hyperledger Aries then decided to create a second version specification to allow for better interoperability. The open standard is co-sponsored by Hyperledger and DIF. Currently, the DID Comm standards are hosted by DIF.

The main difference between the two iterations as shown in Figure 12, is that the latter is specified to be adopted by everyone instead of being specific to the Aries project.

DIDComm v1	DIDComm v2
Born within Hyperledger Aires	Active DIF Working group
Active production across multiple vendors & codebases; full backwards compatibility with Aries & Indy solutions	In development with a broad community of implementers beyond just Aires implementations
Loosely based on JSON Web Encryption standard	Pursuing full JOSE standards (JWM, ECDH-1PU, etc.)
Special Case handling of Peer DIDs	Uniform DID Method support
Dedicated connection handshaking	Streamlined connection handshaking

Figure 12: DIDComm Version Difference

Source: DIF [25]

The first protocols added to DID Comm was implemented in the Aries project as RFCs, in the same spirit as was the first version of DID Comm. This is why DIF also started to standardize protocols, most of which were directly donated by the Aries project[26].

There are currently two main implementations for using DID Comm that provide a procotol implementation rather than just managing DID Comms:

- Hyperledger Aries
- Veramo (formerly Uport)

4.3.1 Aries

Aries is one of the largest joint development projects that focus on delivering frameworks that abstract the complexity of DID Comm between SSI agents, underlying cryptographic operations, safe storage and DID resolution[27].

The project includes a collection of RFCs on how to interact with others on a DID Comm. There are several frameworks written in different programming languages and each project has their own goals and focus in the implementation of the Aries RFCs[28].

The current implementation has two major interoperability versions of RFCs collections[29]. They are not backwards compatible and not all frameworks currently support both versions without interoperability issues at the moment[30].

4.3.2 Veramo

Veramo (formerly named Uport) was one of the first implementations of the SSI concept based on Ethereum smart contract. They had multiple PoCs, including one with the Canton of Zug in Switzerland, where the project was later rejected by the residents. This first version was implemented before the DID standards existed[31]. Later developments focused on the W3C standards, which were at an early draft stage[32]. Today, the Uport project is split into two projects in beta. The identity platform has been renamed Serto and the SSI agent has since been renamed Veramo.

The third iteration called Veramo is written in Typescript and proposes APIs for SSI functionalities as well as VC management.

4.4 Identity Network

A DID and a DID Doc are necessary to create a DID Comm, as discussed in Chapter 3.5.3. For Peer to Peer (P2P) connection between wallet holders, this is not a problem, since the necessary information

is directly sent to the other party. The challenge, however, lies in deciding on how to store publicly accessible DID Docs for Issuers, which are necessary to verify that VCs are valid.

This is the identity network's objective. It should allow to store DID Docs and be resolvable with a DID.

There are multiple ways to implement an identity network. Some focus on a specialized implementation of decentralized nodes and others build protocols upon existing decentralized solutions, such as Bitcoin, Ethereum, etc.

4.4.1 Indy

Hyperledger Indy was first implemented by Evernym and later donated to the Hyperledger Foundation[33]. It is a distributed ledger built specifically for decentralized identities[34].

Indy is a permissioned ledger and is composed of two node types:

- Validator nodes coordinate with each other and validate all write operations in the network. The network operator must ensure that these nodes are operated by trusted institutions.
- Observer nodes store a read-only copy of the ledger from the validator pool and serve the read operation from clients.

Indy is currently one of the more popular options to act as an Identity Network. There are several foundations and companies that provide public networks of Indy nodes:

- VON [35]: Network launched by the governments of British Columbia, Ontario, and Canada. These governments also provide a portable development level Indy Node network.
- Sovrin[36]: One of the first networks available.
- Indicio[37]: Popular network with many partnerships.
- IdUnion[38]: European network which could be used for a European identity.

4.4.2 Sidetree Protocol

The Sidetree Protocol proposed by DIF [39] is a specification to create scalable identity network over an existing blockchain.

The Sidetree protocol consists of three components (see also Figure 13):

1. an anchoring system which saves Sidetree operations or writes transactions;
2. Sidetree nodes that observe the anchoring system and replicate data on the Content-Addressable Storage (CAS);
3. a CAS, which is a storage where data is retrieved by its content rather than its location.

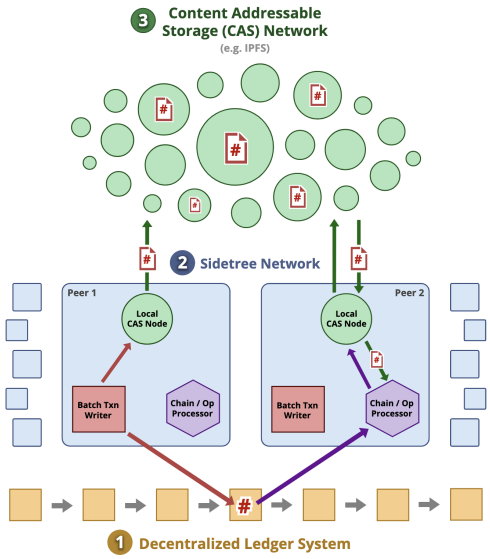

Figure 13: Sidetree Network Topology
Source: Sidetree Specification [39]

There are currently two major implementations of this protocol:

- ION, which is made by Microsoft and uses the Bitcoin blockchain and InterPlanetary File System (IPFS) as CAS; and
- Elements, which is made by Transmute and uses the Ethereum blockchain and IPFSs as CAS.

The Sidetree protocol is permissionless and everyone can start their own Sidetree nodes and publish new DIDs.

4.4.3 Keri Protocol

Key Event Receipt Infrastructure (KERI) is a lightweight blockchain that can be used as an additional layer to an existing ledger. KERI stores key events (key rotation, signing, etc.) and act as a security log[40].

KERI allows an entity to prove control over a DID and brings a thin scalable interoperability to the ledger. The ledger will have fewer complex and costly operations to perform and can bring additional security redundancy.

KERI also has portability features that allow KERI logs to be exported to another ledger with the DIDs in question. This could be useful when a ledger becomes deprecated and the DIDs need to be transferred[41].

4.5 DID Resolution and Management with DIF

With more and more DID methods and different ledgers, the question arieses as to which one developers should implement in their solution.

DIF started two projects to enable future proof DID resolution for clients and DID management for issuers:

- Universal Resolver to resolve any kind of DID methods and retrieve DID Docs; and
- Universal Registrar to create, modify or deactivate DIDs.

Both projects can be operated over REST APIs and can be used to abstract the DID interaction and create an interoperable SSI Ecosystem.

4.6 Mediator

Mobile wallets cannot communicate directly with each other as they do not have a public endpoint with which to initiate a communication. This is why mediators are required to be set up. Mediators have a public endpoint and forward the communication to the mobile wallets.

Every time a user wants to communicate with a mobile wallet, it has to encapsulate the encrypted message into a message to their mediator (see Figure 14). The mediator will receive the address of the receiver and their encrypted message. It will unpack the received message and forward the encrypted message to the mobile wallet receiver.

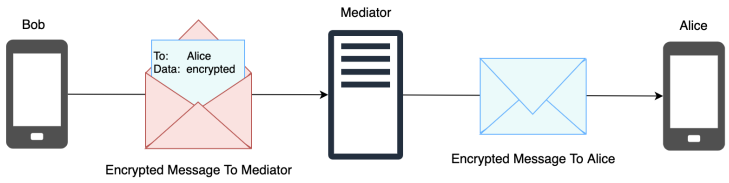

Figure 14: Mediator Forwarding Message

Due to the current way Internet works, mediators play a central role for coordinating communication between client devices and queuing messages when clients are not available. The concept is not new and can be found in many other places like messaging protocols, such as XMPP[42].

Most mobile wallet providers use their own public mediator, to which they can add additional proprietary features to enable push notifications and cloud backups[43].

However, some of these additional propriertary features could potentially pose a risk to data privacy. For instance, additional VCs images stored on the mediator are such features. The Mediator's access

could contain the user's IP address as well as all images that the user downloads. As a result, correlating this information with geolocation could lead to the identification of a user.

4.7 Wallet Implementation

This chapter provides a non-exhaustive list of SSI wallet implementations and their feature set.

All wallet implementations support a common set of features, namely responding to a DID Comm invitation, receiving VCs and responding to proof presentation requests.

Most wallets also depend on an Indy network and must be configured to use the same one as their peer to work properly.

4.7.1 Trinsic

The Trinsic wallet is one of the more feature-rich implementations of the SSI wallets. In addition to the general features, it supports the creation of a connection invitation. This feature allows to create a connection between wallets and to use the exclusive chatting feature that other wallets do not implement.

It also offers push notifications by new requests from others and allows to create local or cloud recoverable wallet backups.

4.7.2 Esatus

The Esatus wallet also supports push notifications and the creation of recoverable wallet backups like Trinsic but only locally.

They are the only implementers to support VC images that are stored on the Mediator. This implementation has privacy issues and is discussed in Chapter 4.6.

4.7.3 Evernym

Evernym, the creator of Indy, also offers an application, but it only supports the general feature set, without any additional feature.

4.7.4 Lissi

Unlike other wallet implementations, Lissi scans multiple Indy networks and makes it simultaneously operable on several Indy networks. Such an approach raises some security concerns, which are explained in Chapter 6.3.3.

4.7.5 Implementation Gaps

All tested mobile applications allow to create connections and receive credentials by scanning QR codes. The wallets are implemented mostly as passive storage and will only react to interaction started by others connected to the wallet. The only exception is the Trinsic wallet which can create a connection invitation and has a chat function with connected peers that also use the Trinsic wallet.

Table 2: Mobile Wallet feature table (State January 2022)

Features	Trinsic	Esatus	Evernym	Lissi
Create Connection	X	X	X	X
Receive VCs	X	X	X	X
Push Notification	X	X	X	X
Present VC on Request	X	X	X	X
Local Backups	X	X		
Cloud Backups	X			
Multi-ledger Support				X
VC Image Support		X*		
Create Connection invitation	X			
Chat function	X			
Downloads on Google Play	1000+	500+	5000+	1000+

X*: With privacy issues

4.8 Mobile Wallet Acceptance Observation

During the observation of multiple wallet providers, one pattern came up multiple times. Currently, most wallets have a low amount of downloads, which usually lies between 500 and 5000 users on the Google Play Store[44][45][46]. Further, these wallets are only used by people that are already familir with the concept of SSI, they are ranked about 4-5 stars on the app stores. Unfortunately, this cannot be verified anymore, as every wallet (except Evernym) disabled their rating on the Google Play Store. But providers that announced their solution and had some media coverage, such as the failed attempt to launch digital driver's licenses in Germany[47], quickly gained over 50'000 users, received a lot of negative feedback and have one of the lowest ranks available, namely ranging from 1 to 3 stars[48].

The reason for this is mostly the lack of usability and stability, which was neglected prior to publishing the application on the app stores.

4.9 Conclusion

Research on related work about SSI made apparent that SSI is a vast concept, which is built on various components that must be coordinated. Further, many questions are yet to be answered and standards are yet to be created. Currently, the standards are evolving fast and it is difficult to keep track of every new proposal and community movement.

That being said, it is possible to conceptualize a working coordination of SSI components and their variations (as explained in this chapter), as well as further elaborating on the opportunities SSI offers

The current state of mobile wallets makes it clear that there is a lack of interactivity between peers, which would enable wide adoption and use of digital identity in the physical world.

To bring SSI to a whole new level, the next chapter will explore ways to use SSI in the physical world and innovate identity validation in a privacy-friendly way.

5 Use Case for Physical Interaction

Last chapter concluded by listing the different possibilities that current mobile wallets offer for digital identity and their gap for wide adoption. Gaps existed for the user interaction with the mobile wallet and the missing features to enable physical interaction. This chapter will focus on possible implementations to make SSI viable for physical identification using their digital identity.

Wallet Interaction between physical near peers is essential for the wide adoption for SSI. Peers can connect securely to each other and verify claims that create new-found convenience and safety. Companies and privates do not have to trust on possible fake or forged physical proofs and can trust tamper proof and verifiable digital credentials.

5.1 Actual Solution

At the time of writing, the common SSI use case is to authenticate yourself by scanning a QR code on the web service you want to access. The user has to accept the connection invitation from the web service first and then accept to share they requested information (see Figure 15).

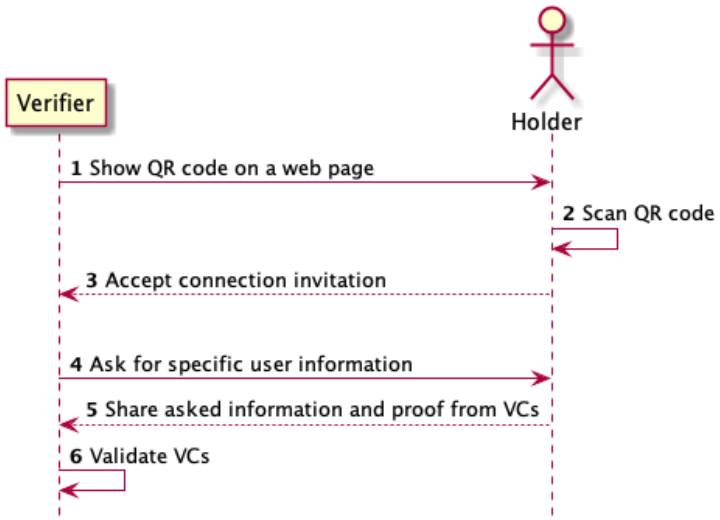

Figure 15: SSI Online Identification

The use case is adapted for online authentication but it is unadapted and not convenient as other alternatives like the actual vaccination certificates or plane ticket where you only scan a QR code on the phone of the holder. These convenient alternative are currently implemented as centralized services and the goal is to implement them with the SSI concept described in Chapter 2.2. But these centralized service comes with multiple disadvantages as the requirement to have one application per proof type, possible data leaks and phishing like attacks to take over accounts.

5.2 Physical Identification

Physical identification is not convenient because it is hard to automate processes around it to make life more convenient for their user. One example could be the work of a police agent that requests the driver license and other physical documents, they need to verify that they are not fake by typing the id number into an application to get the necessary information from a centralized service. The same case with SSI makes the whole interaction more secure as the VC of the driving license is tamper proof and the police agent can rely on only the necessary information needed to do their control.

Cases like the police control can be generalized in a generic use case that is necessary for wide adoption of SSI. It will be the convenient way of proving user's identity to other over a simple means of transportation like QR codes or Near Field Communication (NFC).

A verifier, should it be a machine or a physical person, should be able to ask for proofs and receive the asked proof over easily integrable medium as QR codes, NFC or others (see Figure 16).

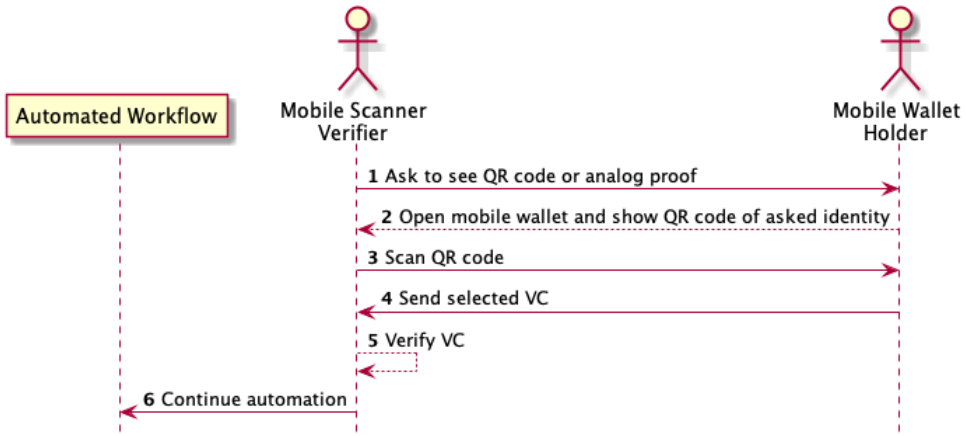

Figure 16: Verifier asking a Holder for their Identity

1. Verifier asks to see QR code
2. Holder opens the wallet and select credentials to show proof
3. Verifier scans QR Code with mobile scanner
4. Secure transaction of credentials
5. Verifier validates the identity
6. Automated processes continue with validation status

The generic case can be used to explain more practical examples of real life situations:

5.2.1 National Identity

A nation resident receives his national identity as VC. The resident can use the VC to identify themselves in a tamper proof way and only share information that are relevant for their use case. This can be useful for public institutions or resident control (see Figure 17) but also private companies that need to do a Know Your Customer (KYC) process.

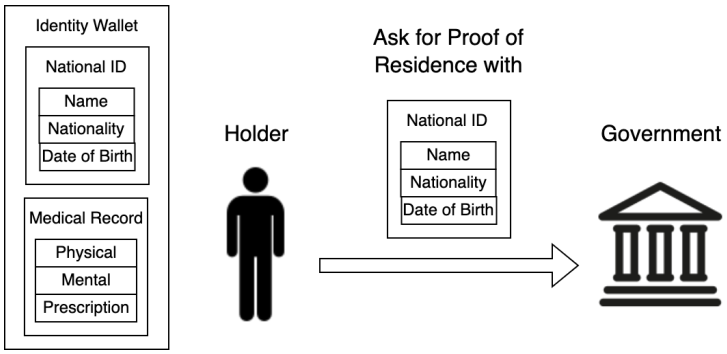

Figure 17: Use Case National Id

5.2.2 Healthcare

A patient receives their medical record as a VC from the Hospital. Each time the person is in treatment, they can share only the required information to the doctor instead of the whole record (see Figure 18). It would benefit the privacy of the patient as there are countries where physical and mental cases are saved together in the same record.

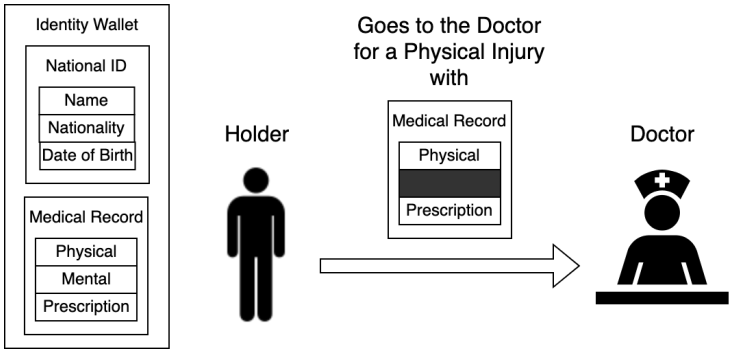

Figure 18: Use Case Healthcare

5.2.3 Pharmacy

A person goes to the doctor and receives a drug prescription on their mobile wallet. The doctor in this case is the trusted issuer and issues a VC to present in pharmacies. The person can go into a pharmacy and present their VC to be able to get their prescriptions (see Figure 19). The pharmacist becomes the verifier and can validate the VC for tamper proof drug prescriptions.

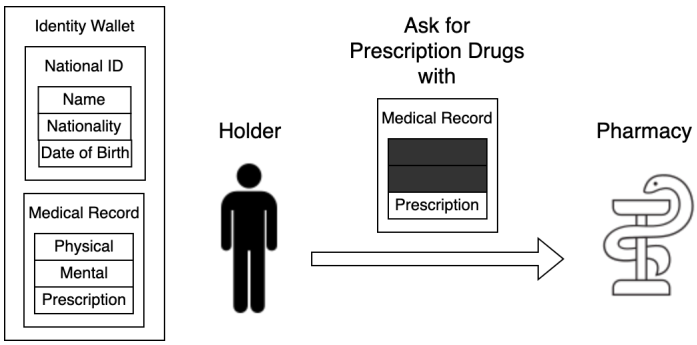

Figure 19: Use Case Pharmacy

6 Analysis

This chapter analyzes the feasibility of the use case and lists the different possibilities with their advantages and disadvantages. It does intentionally not draw conclusions for explored possibilities to act as a reference for future research. Conclusions to selected concepts and frameworks can be found in Chapter 7.

6.1 Technical Use Case Deconstruction

The use case from Chapter 5.2 can technically be represented as in Figure 20.

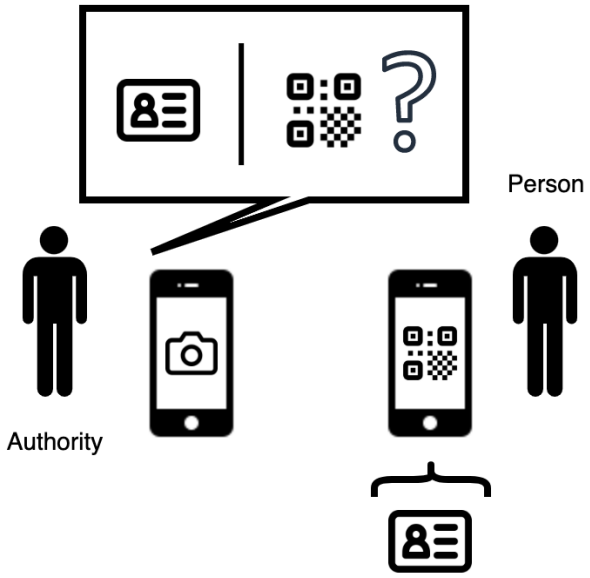

Figure 20: Technical Use Case Deconstruction

Each user has a client device with which they control their identity. On the question about which type of devices they are, we take the following assumption:

Analogically to the physical wallet with their identity, a person has them almost always on them and can use them to identify themselves. To create the same accessibility and broad adoption for their digital identity it will be assumed that the client devices are mobile phones. Other devices like laptops would also be possible but they don't have the same practical size factor and would possibly need an additional optical scanner to be useful.

6.2 Mobile Wallet Variation

SSI Implementation can have two major forms of mobile wallets:

- Local mobile wallet (Edge Wallet)
- Cloud wallet managed from the mobile application (Cloud Wallet)

6.2.1 Edge Wallet

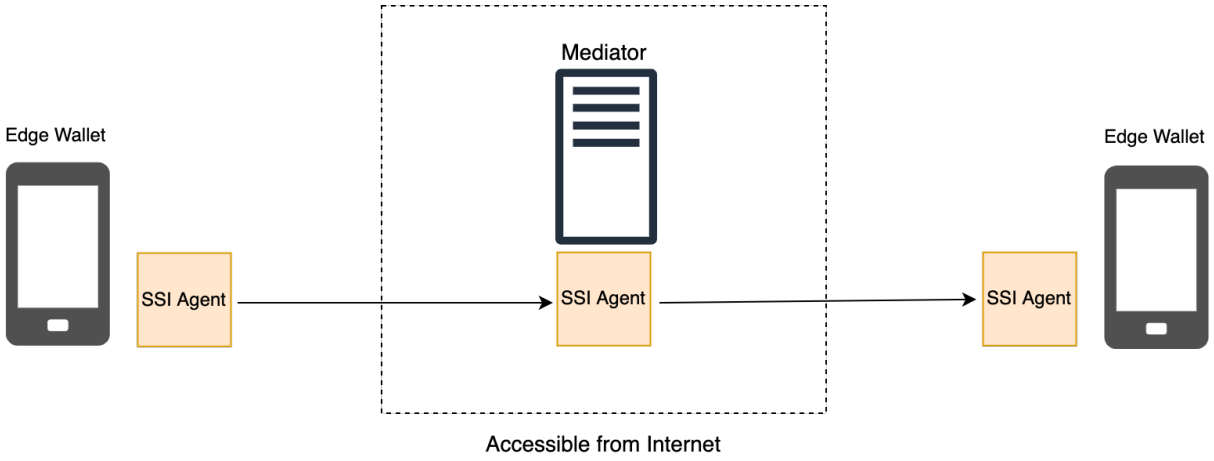

Figure 21: Edge Wallets

Edge Wallets are mobile applications that include an SSI agent. They include most functionalities to be SSI enabled which means that users are managing their digital identity themselves and are truly self-sovereign (see Figure 21).

But there is a major problem with P2P communication due to IP network accessibility restriction on client devices (see Figure 21). To communicate with another client, there is the need for an accessible mediator from the Internet to relay the messages.

There is also the concern about identity backup and restoration. It should be convenient and easy for users to backup and restore their digital identity as they will be responsible for it and there will be no third-party help like in the old fashion way when a user forgot their password.

6.2.2 Cloud Wallet

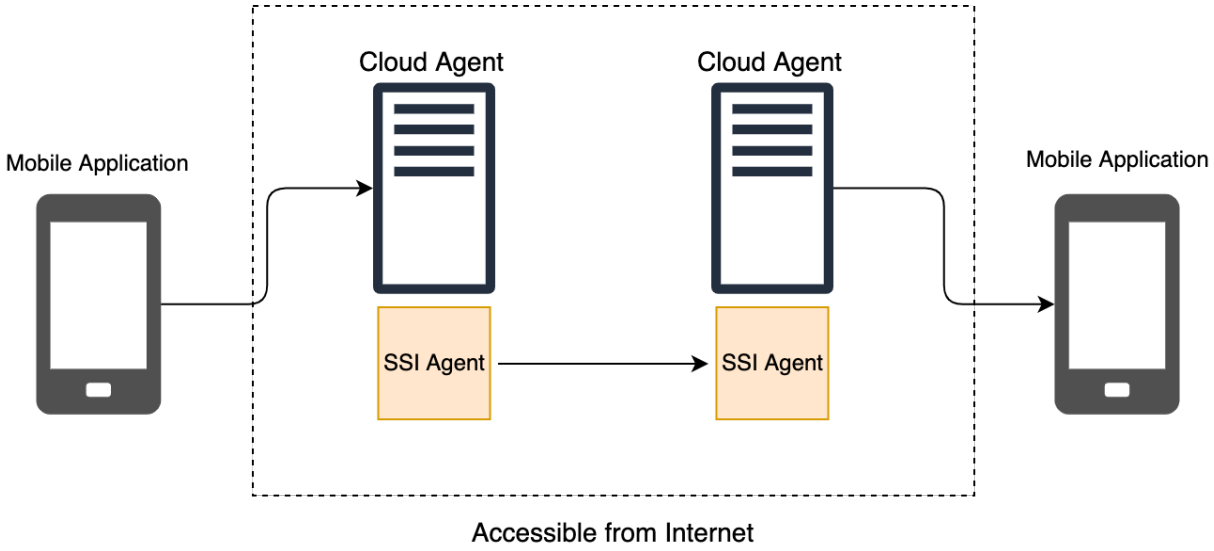

Figure 22: Cloud Wallets

In the other approach (see Figure 22), the SSI agent is hosted on a centralized service and governed by a mobile application.

The main concept of SSI goes lost because the user loses their true self-soverinity and relies on a centralized service to do the transactions.

It has the benefit that the development of the mobile application is simpler as they can abstract away the SSI agent controller role. The need for a mediator is also solved as the cloud agent manage the connection to the mobile application itself.

But there are several disadvantages, the mobile application needs an additional authentication and authorization scheme to access the cloud agent like a login or a certificate. Additionally the solution loses flexibility compared to the edge wallet where the agent can be freely used and configured where the cloud solution is dependent on the centralized implementation and integration.

6.3 SSI Agent

In every case there is the need for an SSI agent, should it be deployed on a cloud server or embedded in a mobile application.

The SSI agent must support to do the following requirements:

- Deployable on cloud servers or embeddable in mobile applications
- Create a DID Comm connection between two peers (preferably didcomm/v2)
- Receive verifiable credentials
- Send verifiable presentations to other peers
- DID Resolution of at least one DID method
- Secure storage of private keys and data
- Backup and recovery process

6.3.1 Veramo

Veramo is a framework written in Typescript and can be either deployed with NodeJs on a server or embedded into a mobile application with the cross mobile React Native framework. It has documentation on which APIs are available but not on the workflows like basic DID Comm between peers. Veramo is currently in public beta[49] and has a flexible plugin system where only necessary components can be loaded.

It offers multiple plugins that can fulfill the requirements:

- `didcomm` gives the capability to communicate between peers with DID Comm v2.
- `credential-w3c` permits the issuance and presentation of VC. They follow the W3C data model standard.
- `did-provider-ethr` supports at least one method for DID resolution for the ETHR method
- `data-store` allows for local storage for the KMS systems and the DID management

Veramo comes from a long path of changing standards (see Chapter 4.3.2) and their new iteration is at its beginning. Currently, it is incomplete and missing documentation of basic interaction makes the feature estimation of the framework more time consuming as it would be necessary to search in community posts or implementations to see if the use case could be fulfilled.

6.3.2 Aries Frameworks

As explained previously in Chapter 4.3.1, Aries is a co-development project with multiple different implementations.

To ease the development and interoperability of these Aries frameworks, there are two major versioned enumerations of RFC concepts and features known as Aries Interop Profile (AIP). Each version includes a set of RFC for Aries agent builders to implement.

AIP 1.0[50] depends heavily on concepts from Indy and misses newer standards for wallet and credential representations.

AIP 2.0[51] updates the existing RFCs to be conform to newer standards, removes the dependencies to Indy, offers clarification to critical mobile architecture like how mediator interaction should be made.

In short AIP 2.0 offers the following advantages:

- Not Indy dependent and no schema definition on the ledger anymore
- Ledgers can be Decentralized Public Key Infrastructures (DPKIs) with public keys and a public endpoint
- Reuse of already existing connections between peers
- Standardization of the mediator coordination
- Based on W3C standards and interoperable with future technologies
- More security and privacy in general

Not all Aries frameworks support both AIP version at the moment of writing. Currently there are five major Aries framework implementations that support different AIP versions.

Table 3: AIP Compatibility of different frameworks (State 23.09.2021)

	AIP 1.0	AIP 2.0
Aries Cloud Agent Python (ACA-Py)	X	X
Aries Framework Go (AF-Go)		X
Aries Framework JavaScript (AFJ)	X	
Aries Framework for .NET (AF-.NET)	X	
AriesVCX (VCX)	X	

The major mobile wallet providers base their implementation on AF-.NET (Trinsic, Esatus and Lissi) which support AIP 1.0. but the community wants to push frameworks into the direction of the second iteration with a common roadmap to AIP 2.0 and interoperability[52].

Currently the only Aries frameworks that say to support AIP 2.0 are ACA-Py and AF-Go. But ACA-Py's version support has a catch. ACA-Py has legacy implementation used in the protocols of the newer AIP version and does not conform exactly to version 2.0[53]. This also means that a ACA-Py agent cannot communicate with a AF-Go agent.

AF-Go implemented a temporary build flag which makes it compatible with the erroneous ACA-Py implementation but makes it incompatible with AF-Go instances built without. This flag is only meant to be used until ACA-Py fixes their issues.

AF-Go is a newer implementation based on the W3C standards and is a pure implementation in Go. It only supports AIP 2.0 and is incompatible with all current mobile wallet because most of them are based on AF-.NET AIP 1.0[52].

Also it is the only solution to date to use AIP 2.0 on mobile because AF-Go supports the compilation to an embeddable library usable for Android and iOS development.

6.3.3 Limitation on Indy Ledger Selection

Due to an early conception mistake on DIDs for Indy nodes, it is not possible to target a specific ledger provider easily without a security compromise.

For the history, Sovrin started as the first Indy provider and proposed to use DIDs with the following format:

```
1 did:sov:123456789abcdefghi
```

The problem is that there are multiple networks of Indy nodes available today and their DID are indistinguishable. SSI agents have to load a file with ledger configuration named Genesis file beforehand.

Some mobile wallets like Lissi used a workaround where they loaded multiple Genesis files and check if the VC schema exist on the ledger or not. This could seem clever but it is not scalable in the long term and it has a major security concern. It is possible to confuse the SSI agent implementing this technique by duplicating the VC schema and load it on another ledger that will be scanned. Depending on the role that the agent has, it would be possible to impersonate digital identities or to bypass authorization steps with fake identities.

A real solution to the problem came afterward as a second generation DID method which is currently in the specification phase[4]. Addresses on Indy base ledger will be represented by a new DID method `indy`. The (+DID)includes a namespace component that enables resolving a DID to a specific Indy network:

Sovrin MainNet ledger

```
did:indy:sovrin:123456789abcdefghi
```

Sovrin StagingNet ledger

```
did:indy:sovrin:staging:123456789abcdefghi
```

6.4 Backup and Restoration

Backup and Restoration processes are important for the wide adoption of SSI. Analog to a physical wallet, losing a wallet is a huge time consuming procedure that involves fast revocation requests to governments, banks, insurances and other important institutes because someone with the lost wallet could use the identity of the owner.

SSI can drastically improve the process by allowing to backup and restore the user credentials. In the best-case scenario, a user can simply restore their backup without any consequences and in the worst-case it is similar as if they lost a physical wallet.

The actual backup concept used by wallet providers and viewed as best practice is inspired from the cryptocurrency world and consist of a set of words (frequently 12 or 24 words) to remember in the right order. They are also called recovery phrases and are used to generate a private key to encrypt a backup. The users can store it securely themselves or the mobile wallet provider can propose to store it on the cloud.

There is one major consideration to take into account and it is the case that someone gets possession of a backup and the recovery phrases. An attacker with both information can impersonate the original owner to verifiers and will need to revocate all their VCs and ask them again (similar to the analogy with the physical wallets).

[4] https://hyperledger.github.io/indy-did-method/

6.5 Trust Anchor

Trust anchor are authoritative entities that are vital to ensure the legitimacy of someone's or something's identity. Most people trust anchors from the root CA certificates that are included in operating systems and browsers to ensure that a website is secure and is what it pretends to be[54].

In the case of SSI, the most important identity are the public DID of issuers. An issuer will issue information about an entity as a VC. Any verifier must request the resolution of the DID and verify that the VC's proof is valid with the public key of the issuer. The verifier knows that the VC is valid but does not know if they trust the issuer with the provided information.

To fulfill the trust issue, verifier will need to whitelist some issuer DIDs that are trustworthy to them. That could be a database created by an entity that does certify the identity of companies like the GLEIF[55], a linked domain challenge where the issuer needs to have control over a domain name or an internal selected list.

But it is not the only issue, there is also a trust issue between the holder of VCs and verifiers. Holders could be phished by wrong verifiers to leak their personal information. The solution to that issue could be similar to the solutions for verifiers with exception of the internal selected. An internal list of selected trustworthy verifiers would be impossible to manage without removing interoperability and the possibility of VCs to prove their owner.

7 Conception

This chapter lists decisions based on the discoveries of the last Chapter 6 and describes the new concepts needed to extend SSI to the physical world.

7.1 Mobile Agent

Considering both options, an edge wallet or a cloud wallet, the choice goes to an edge wallet primary because of the added flexibility, privacy and security.

Cloud wallets defeats a bit the purpose of a user-centric wallet where the holder is in control because the data of the user are centralized on one platform. Edge wallets need an additional mediator to communicate with each other but a cloud wallet would need additional security measures between the mobile wallet and the cloud wallet.

Scenarios like direct connections over Bluetooth or other future transport mode would optimally be used by edge wallets who do implement the communication protocol internally.

Edge wallets give the users more control over their digital identity and a more feature complete SSI mobile wallet.

7.2 Agent Framework

Comparing advantages, features and disadvantages, the choice will be to go with one of the Aries Frameworks. This choice is motivated by the experience from internal colleagues and the fact that Veramo is still in public beta[49]. Veramo also misses important introduction documentation such as the creation of a DID Comm between two peers and their message routing[56].

To the selection of Aries Frameworks, the choice comes done to any framework that implements AIP 2.0, because the first iteration is seen as deprecated as it is too dependent on Hyperledger Indy and does not use the revisited protocols with support of standards defined after the creation of AIP 1.0.

During the redaction of this thesis, the only available Aries framework that supports AIP 2.0 is AF-Go. It supports all necessary requirements, has an active developer community with regular fixes and new features. The repository averages between 2 to 4 pull requests per week[57].

The framework should also be embeddable in a mobile application as explained in Chapter 7.1. AF-Go supports it over the gomobile[58] project which creates java and objective-c bindings. The bindings can be used in a mobile application to call the exported go functions.

7.3 Identity Network

There will not be a concrete choice for an identity network as the thesis focuses on the mobile wallet. Nevertheless, thanks to the DIF universal resolver, it will be possible to support the most common DID methods. The Aries Framework can resolve unknown DID by calling the universal resolver REST API and continue validation with the fetched verification method and public key.

7.4 Mobile development

Android and iOS are supported mobile platforms from AF-Go.

This thesis will focus on Android development for the following arguments:

- The author has experience in Android development.
- Debuggability and deployment for demonstrations are easier to do on Android.
- The author has multiple Android smartphones available to deploy and test.

Cross-mobile frameworks like React Native or Flutter would also have been possible but there were multiple arguments why native development was considered:

- Hardware Security Module (HSM) and other native components are easier to access and use with native development.
- Data storage management is easier on native without having to rely on too much plugins.
- Native has better performance and better background tasks support than cross-mobile solutions.
- Native has better stability, debugability and has better long-term support.

The development will take more time as two separated applications will be necessary but critical application like mobile SSI wallet need the extra security that native can access and will feel more responsive when developed natively.

7.4.1 Fingerprint Support and the Minimum Android API

The minimum API level for the Android application will decide which versions of Android will be able to run the application. A good compromise is to use the lowest possible API that support critical functions that are needed in the application.

In the SSI wallet, a crucial feature will be the fingerprint authentication as it will enable to encrypt and decrypt data with the cryptographic key in the Android Keystore. The feature was added to Android 6.0 (API 23) and the current Android distribution statistics show that it would run on 94.1% of devices (see Figure 23).

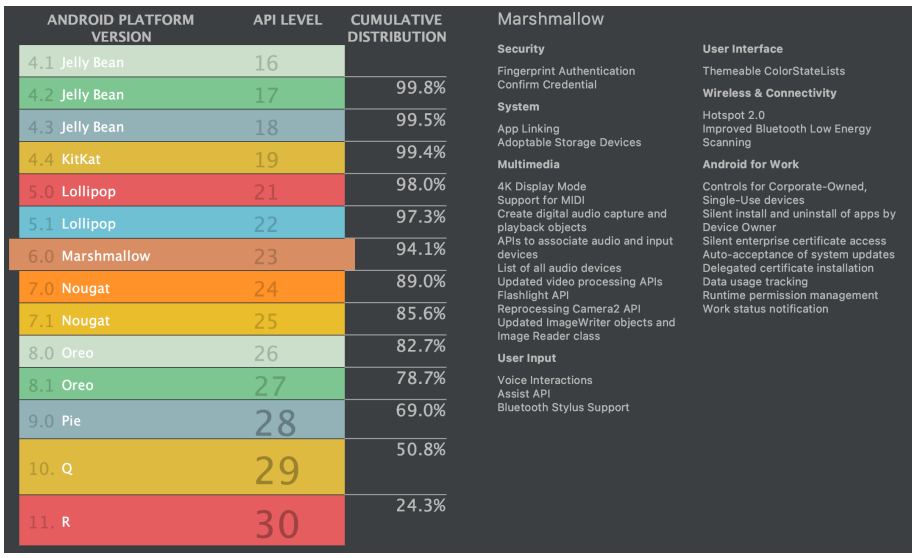

Figure 23: Android Distribution Statistics December 2021

The Android distribution statistics can be found in Android Studio[59].

7.5 Aries Protocols

Choosing to go with the Aries Framework means to understand which protocols are necessary to implement the use case from Chapter 5.2.

In summary, the mobile wallet must be able to:

- Req. 1: Set up a secure connection between itself and another Aries agent.
- Req. 2: Receive VCs from issuer
- Req. 3: Present VCs to verifiers
- Req. 4: Validate VCs presented by holders

All explained protocols come from the Aries RFCs and are implemented in AF-Go. The sub-chapter explains how they work and how they can be used to fulfill the requirements above.

7.5.1 Setup a Secure Connection

A secure DID Comm can be established over the out of band protocol with an embedded didexchange connection invitation. The out of band protocol is a wrapper for other Aries protocols that need a secure connection between peers. The most basic utilization is to encapsulate the invitation payload for the connection protocol didexchange[60].

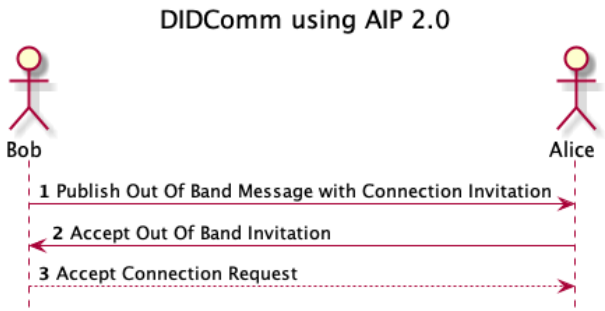

Figure 24: DIDComm Using AIP 2.0

As presented in Figure 24, Alice receives the connection invitation from Bob over traditional of transport medium (Qr Code, E-Mail, etc.). Alice accepts the out-of-band invitation and send confirmation to Bob. Finally, Bob has to approve the connection request to complete the connection to Alice.

7.5.1.1 Mediator Implication

The simple case of secure connection does only work if both peers have a public endpoint. Edge wallets who do not have one, need to connect and rely on a mediator to receive messages from other peers. This means that the first step for an edge wallet is to create a secure connection and register itself to a mediator.

AIP 2.0 added two mediation coordination protocols which standardize how edge wallets have to register themselves to a mediator[61][62]. A mediator will queue and forward any messages meant for their registered edge wallets.

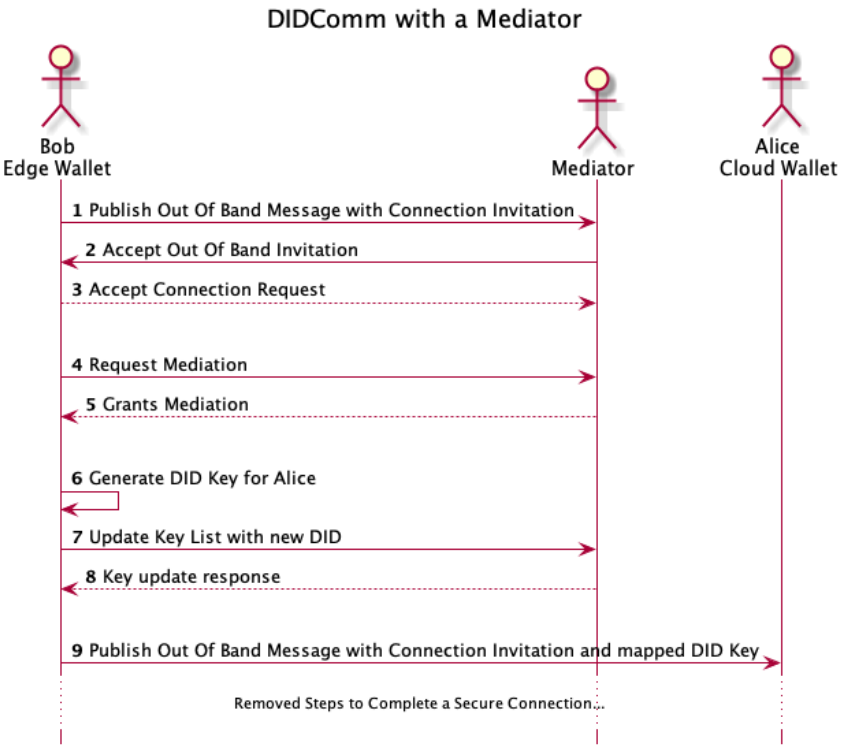

Figure 25: DIDComm with a Mediator

As shown in Figure 25, Bob starts to set up a secure connection between the mediator and himself.

After the setup, Bob request the mediator for mediation. The mediator grants mediation for Bob and sends back the public endpoint and a routing key to include in connection invitations.

At that point Bob will need to call the mediator each time, he creates a new DID to update the DID Key list on the mediator. Without this call it is impossible for the mediator to know where to forward the messages to Bob if he uses a new DID for each of his other peers.

Bob can then use the received public endpoint and the routing key to invite Alice to a secure connection.

Alice does not need to do the same steps as she has a cloud wallet with a public endpoint.

7.5.2 Receive Verifiable Credentials From an Issuer

A wallet must be able to accept verifiable credentials from issuers to have any utility at all. This is why there is also a protocol on how to issue credentials in the Aries RFC[63].

This protocol needs an existing connection between the peers (Req. 1) or the first message must be wrapped in an out of band message.

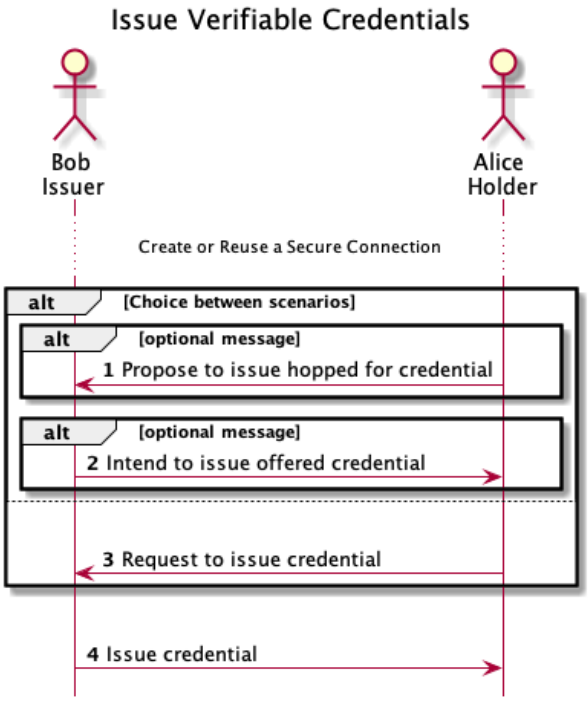

Figure 26: Issue a Verifiable Credential

As shown above in Figure 26, the issuing can start from both peers. Bob, the issuer can offer a credential to Alice but Alice can also propose to receive a credential she hoped for. If neither of these two optional message is sent, it is Alice the holder to request the credential to Bob.

After one of the two paths, Bob will issue the credential to Alice.

7.5.3 Present Credentials to Others and Validate Them From Others

An edge wallet will need to present the credentials that it received from issuers and also be able to validate the ones from other peers. This process is called a credential presentation and involves the transmission of a Verifiable Presentation (VP). A VP is nothing less than a wrapper around VCs and some additional metadata and signatures about the presentation[64].

This protocol needs an existing connection between the peers (Req. 1) or the first message must be wrapped in an out of band message.

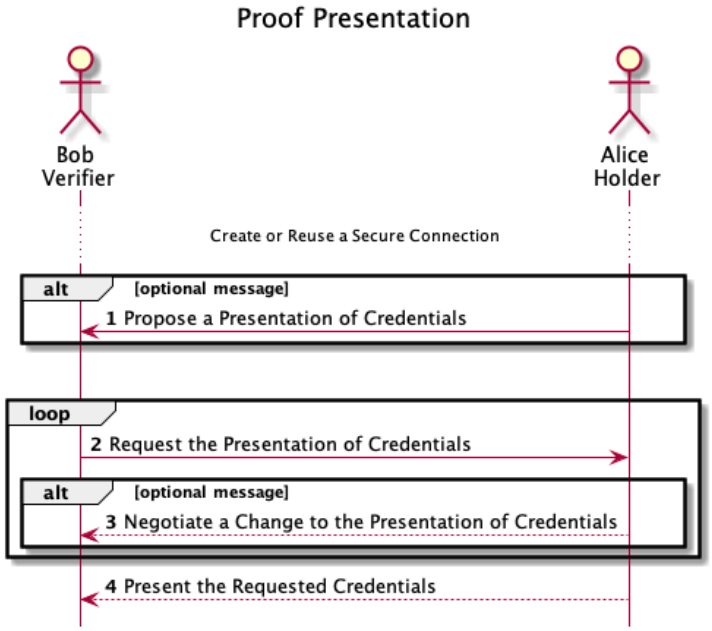

Figure 27: Presentation of Credentials

Alice can optionally start by proposing the presentation of her credentials that she selected.

Bob, the verifier can answer the proposition or directly send a request for specific credentials from Alice. Alice can optionally try to negotiate the requested credentials but at the end, Alice needs to send the requested credentials back to Bob.

7.6 Presentation Workflow for the Physical World

The Aries RFC protocols offer a bit of flexibility and some conception choices are necessary to implement the use case in the physical world. Setup a secure connection and receive VCs do not change but there is a choice on who starts the presentation process after a successful connection.

The QR codes used in this workflow will only represent a communication invitation for more flexibility among peers. The two following choices different themselves on who is responsible on initializing the presentation workflow, a verifier wallet which is possibly only used as a scanner or a generic personal wallet.

7.6.1 Initiated by the Verifier with Specialized Application

Initiating the presentation workflow from the verifier makes the holder's wallet less complex as it only has to react to presentation requests. The holder's wallet also directly know which credential they have to present.

7.6.2 Initiated by the Holder with the Personal Wallet

Initiating the presentation workflow from the holder give them the possibility to prove any of their credential to any other wallet. Some communication of the workflow can also be automated as the holder's wallet has more context from initiating the workflow.

It does make the personal wallet more complex but it gives the holder more flexibility and use cases for their wallet.

7.6.3 Workflow Conclusion

To embrace mobile wallets in the physical world, personal wallet should be able to act independently and be interoperable with others. By that means, the mobile wallet will be able to initiate the presentation workflows when the QR code with the communication invitation is scanned. The other scenario will still be possible as specialized application can ignore the presentation request from the personal wallet and send their own presentation requests with their requirements.

7.7 Offline Verification and Alternative Transports

Offline verification is a unique feature that increase privacy and accessibility as there is no need to transmit over infrastructure like internet or to pass over middlemen.

There are multiple alternative transport possibilities with their advantages but also their quirks.

7.7.1 Bluetooth

There is a DID Comm standard over Bluetooth Low Energy (BLE) in work but it was not in the focus for the last 10 months[65]. BLE is chosen over Bluetooth because the latter is a lot more complex and interoperability is also not simple. In contrast, BLE has a smaller range and a lower data rate but is greatly supported by Android and iOS. Currently no Aries Framework has BLE but it is one of the most promising alternatives as there is exceptional support between Android and iOS[66].

7.7.2 NFC

NFC is a convenient technology as it only requires smartphones to be near to each other to exchange data. Due to its short range, it is not really meant to transport complete communication but it could be coupled to another technology to remove their discovery phase (BLE and Wifi Direct).

Unfortunately, it is currently not usable as Apple does not expose the APIs for the necessary peer-to-peer mode of their NFC chips[67].

7.7.3 Wifi Direct

Wifi Direct could be an alternative to BLE and be used for communication transport. Unfortunately, it is currently not usable as Apple does not support Wifi Direct and uses their own protocol peer-to-peer Wifi which is not documented for third parties[68].

7.7.4 QR Code

QR Codes are good to transmit initial information to create connection between smartphones but are cumbersome if multiple messages must be transferred between peers.

7.7.5 Conclusion on Offline Use Case

Offline communication and verification are not trivial to achieve with cross-mobile support. Ideally, an offline verification would start either with a QR code or NFC to start a communication. It is easy to use and can send over information needed to switch to a wireless communication which permits multiple messages to be exchanged without having to scan more QR codes or to have both mobile phones in proximity of the NFC range.

Sadly, the incompatibility with Apple smartphones makes use cases with Bluetooth, NFC and Wifi Direct not viable.

The solution which would work technically would be to start verification with a QR code and to switch to BLE for communication. It is supported by Android and iOS but there is necessary work for the DID Comm standard on that transport channel.

7.8 Database Schemas

There are implicit persistence requirements for the mobile wallet to use it on a daily basis and to fulfill the use case for the physical world. These data are represented as database schemas to ease comprehension.

7.8.1 Aries Framework Go Storage

AF-Go per default does store data in memory on mobile. To make it persistent, they implemented an interface to let developers persist the data of the mobile wallet.

The schema to persist can be represented as shown in Figure 28. There are multiple stores entities that have names and their own configuration. Each store has multiple data as key and value pairs but they can also be queried by tags or the store in which they are stored.

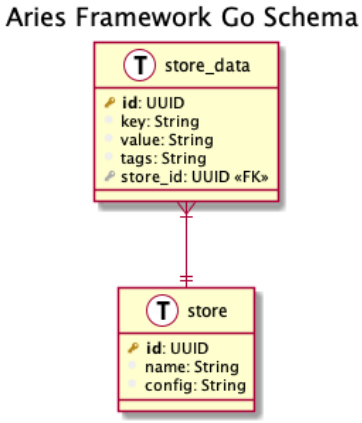

Figure 28: Aries Framework Go Storage Schema

7.8.2 Mobile Wallet History Schema

For accessibility and traceability of user actions and events done by the wallet, it is important to store a history of past events between peers.

All events will save their creation time, their type of events and the connection identifier between the concerned peers (called history_entry in Figure 29).

There are two special cases that inherit from the generic case but need some additional context:

- Events that involve credentials need to store the credential identifier as well (credential_entry)
- Events that send messages need to store who was the sender and the message's content (history_entry)

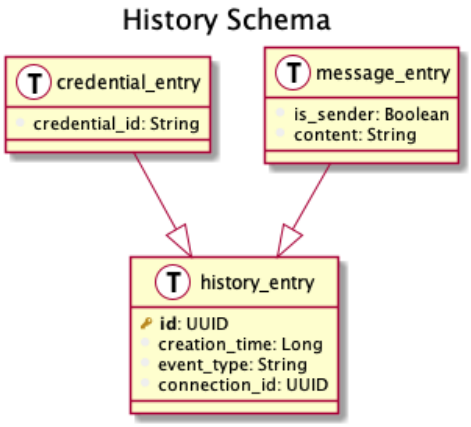

Figure 29: Mobile Wallet History Schema

7.9 Data Encryption

For an application with as critical information as a SSI wallet, it is crucial to encrypt local information about the user's digital identity.

As described in Chapter 6.4, one of the best practices today is to use a recovery phrase to derive the cryptographic key. The recovery phrase can have different entropy-bit values depending of the size of the word list and the amount of word in it.

Entropy bits are the base-2 logarithm representation of the number of possible guesses to find the correct value[69]. The following formula can be used to calculate the entropy bits of a word list of size N and a phrase of K words:

$$Entropy = log_2(N^K)$$

There are word lists available in different sizes available on the website of Electronic Frontier Foundation (EFF). These word lists are specially composed of English words that are familiar and easily spellable based on reading research[70].

The longest EFF word list has 7776 words and the following table shows the entropy bits that result with different number of words in the recovery phrase (only the integral part of the entropy bits are shown).

Table 4: Mapping between the number of words and resulting entropy bits

Words	Entropy bits
3 Words	38 bits
6 Words	77 bits
9 Words	116 bits
12 Words	155 bits
16 Words	206 bits
20 Words	258 bits
24 Words	310 bits

People argue that 128 bits of entropy are enough to secure critical information as it is extremely time consuming[71]. There are theoretical hypothesis that recommends to go with 256 bits of entropy to assure 128 bits of entropy when quantum computers will be available [72]. But as it is currently more important to focus on user convenience, 12 words with 155 entropy bits will be enough.

The recovery phrase is used to generate the cryptographic key with a Key Derivation Function (KDF) to secure the user's mobile wallet and all it's content. The cryptographic key is in the best case stored in a HSM where it is not exportable and the wallet uses the HSM to encrypt and decrypt the user's digital identity.

7.10 Backup and Recovery

Losing a physical wallet can be a complex and long process to recover all cards and identity paper. With a mobile wallet, there is also the same issue but a backup and restore process can be implemented for the user's convenience.

The user only needs to know their recovery phrase and the backup files to restore their digital identity on their mobile wallet (see Figure 30). The encrypted digital identity can be exported to a secure cloud storage as backup. When the user changes their device, the following task can recover their digital identity:

- Download the backup on the new device.
- Generate the cryptographic key from the recovery phrase.
- Import the cryptographic key into the HSM of the new device
- Decrypt and import the mobile wallet data from the backup

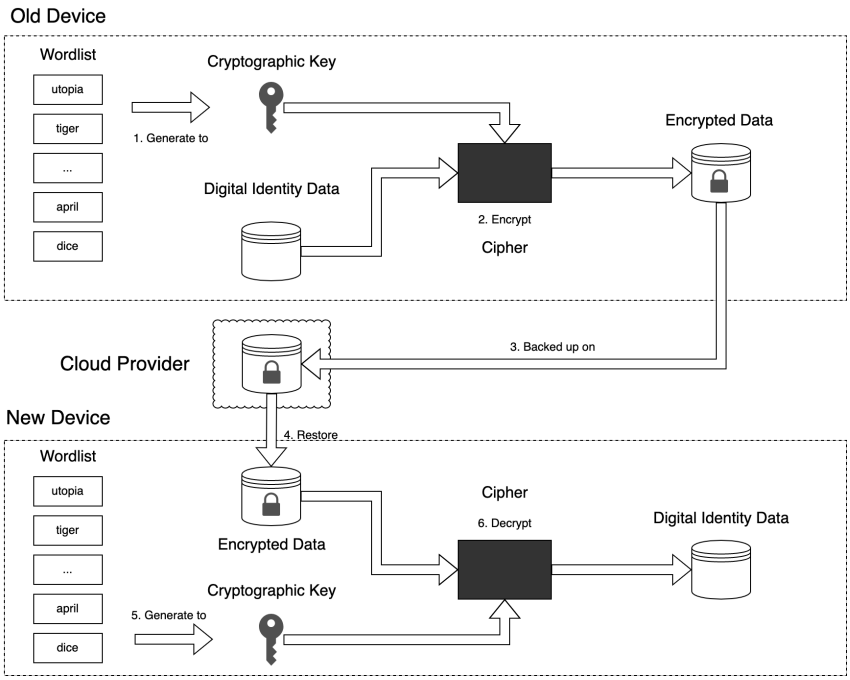

Figure 30: Encryption Concept for the Mobile Wallet

7.11 Conclusion on the Decisions

The design offers a good combination of multiple themes, frameworks and platforms. It ties together data encryption scheme with backup and restore processes and also describes data schemes for data

storage on the Android platform. The Aries protocols implemented in AF-Go also support the Master thesis use case and can be used from the Android platform.

Unfortunately, after inspection, it will not be possible to do offline verification due to the lack of cross-mobile support for wireless peer-to-peer communication and also a lack of contribution for alternative DID Comm transportation layer (other than HTTP and WebSocket).

But the critical decisions made in this chapter come with risks as well:

- Will it be the right Aries Framework to go with?
- Will the mobile wallet be useless without an iOS counterpart and would cross-mobile have been not as restrictive as anticipated?
- Are the design decisions such as the data encryption scheme realizable on the Android platform?

The next chapter will implement the conception decisions and tie together all different components to create a SSI enabled mobile wallet(see Chapter 8).

8 Implementation

This chapter contains interesting technical choices and implementations done to respect the decisions of the conception chapter (see Chapter 7).

Mobile wallet screens are found in Chapter 14.1.

8.1 Architectural Overview

The architectural choice can best be described in conjunction with Figure 31.

The application uses an Activity to manage global UI elements like the top toolbar or the bottom navigation bar. The Activity must also initialize critical components like the navigation components.

Navigating between views is managed by the navigation components[73]. By each view transition, the previous fragment is destroyed and it loads the next fragment.

Each fragment that holds data has a ViewModel companion. The ViewModel itself holds all the view's data and do acts as intermediate between the fragment and other components. Intermediation is necessary to avoid dependencies in the fragment but also for more advanced interaction with services and coroutines[74].

In general fragment and ViewModel do not fetch data but will get it delivered by data streams reactively. They are different available patterns in Kotlin to achieve it and the three used one in the project are:

LiveData LiveData are usually found between fragments and ViewModel. They hold data and can be observed for changes by the fragments. They are also lifecycle aware which means that when a fragment is destroyed, it will automatically remove the observer to avoid leaks[75].

Flow Flows are data streams usually found between services, repositories and ViewModels. They support asynchronous data transformation and will only be executed when they are collected. They have basic operation that can be really useful like flow concatenation where multiple data source with the same object type can be merged together and it will be updated each time one of the data sources emits new values[76].

SharedFlow SharedFlows are similar to Flows with one major difference[77]. SharedFlows are executed even if no one is collection them. In practice it means that is a component start listening to a SharedFlow, it will not receive data emitted before it was listening to it.

The Aries Service does use the SharedFlow internally as an Event Bus[78] to distribute events to all components of the application. One example is the history service. The history service listens to specific events and does save them with the help of the history repository.

And each time the history repository has updated their data, they are emitted back to the listening ViewModels with a Flow.

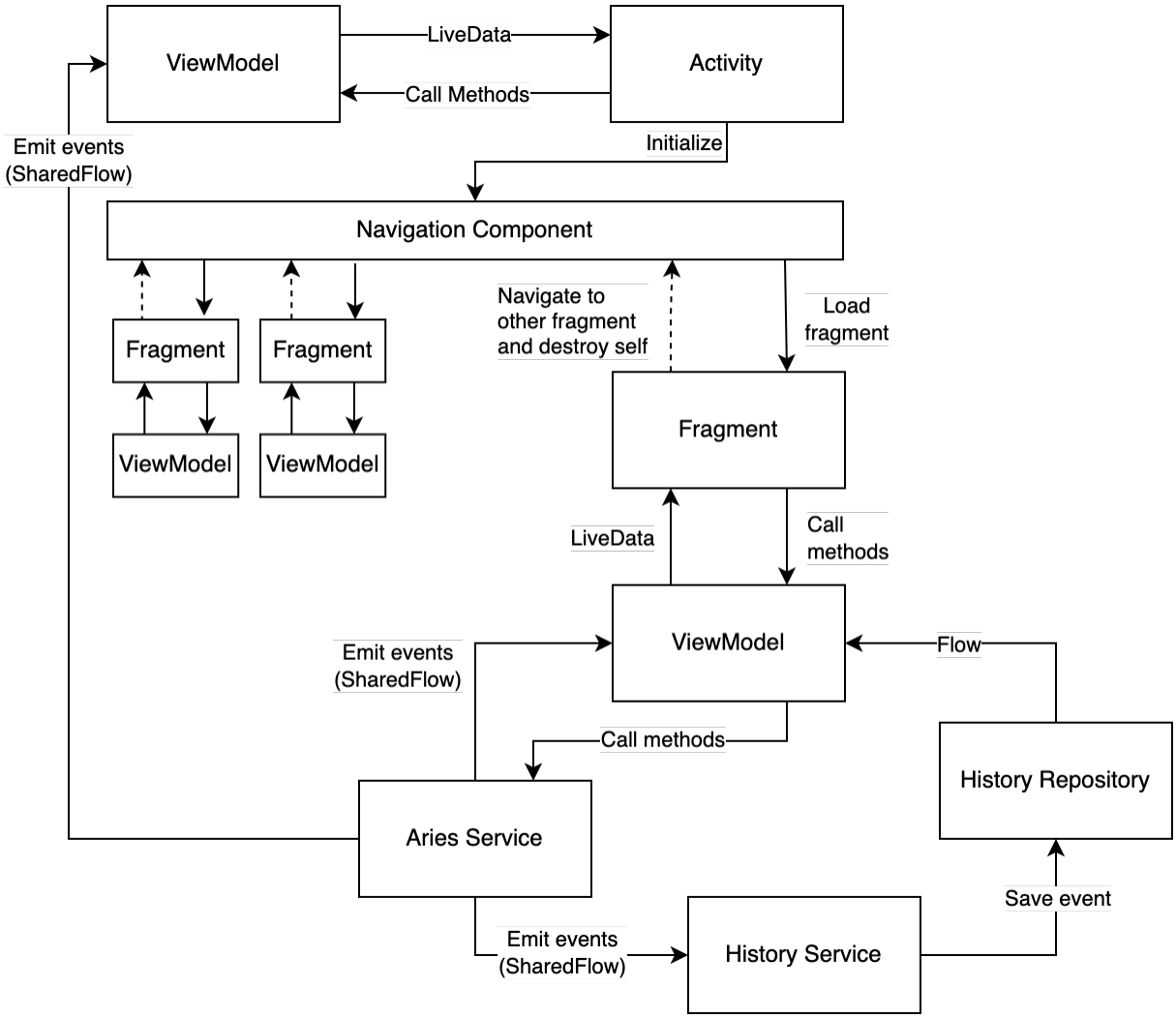

Figure 31: Android Architectural Overview

The application also includes dependency injection globally with Hilt. Hilt is recommended by the Android documentation to increase testability and scalability of android applications[79].

8.2 Integration of Aries Framework Go

The integration of AF-Go is coupled as loosely as possible because of the unknown status of the Aries projects and their standards and protocols that are not definitely defined. It would also help to migrate to another Aries framework if another framework took the implementation lead or the current one

would be deprecated.

The code resides in the Aries package which has a set of useful interfaces and data models (see Figure 32). View models make calls over the `IAriesController` interface and receive events with the `AriesEvent` subclasses.

Internally in the `sdk.go` package, the AF-Go specific controller has its own model definition and storage implementation. It does map the internal models with the public data model when it has to propagate events or return data from AF-Go to the view models.

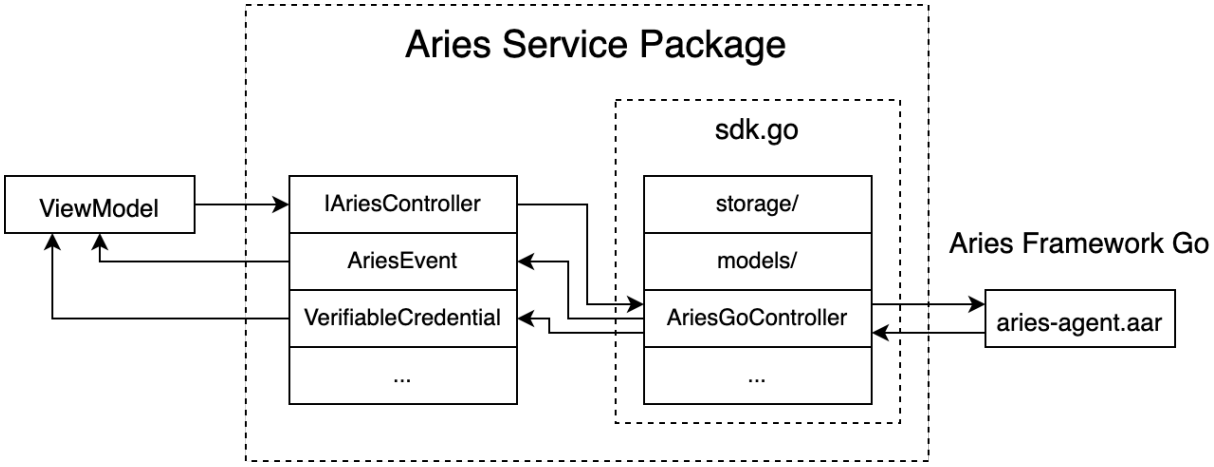

Figure 32: Aries Package

8.3 Data Encryption

The following list of terminology describes the different cryptographic elements used in this chapter:

Table 5: Cryptographic Terminologies

Terminology	Description
Recovery Phrase	List of 12 words known only by the mobile wallet user to generate the master key.
Master Key	Cryptographic key used to encrypt and decrypt the Data Encryption Key (DEK).
DEK	Randomly generated passphrase used to encrypt and decrypt the application's database.

At the first start of the mobile wallet, the user receives 12 words from EFF word list presented in Chapter 7.9. The words are randomly selected from the word list with `SecureRandom`[80] which

generates cryptographic secure random number. The selected 12 words compose the recovery phrase for the user to write down.

The next step is to use a KDF to generate a 256-bit master key (with ~155 bit entropy) with the recovery phrase. Unfortunately, not every Android version support the same KDF versions[81]. The plan would have been to go with a PBKDF2 algorithm but the only ones supported on API 23 are based on a combination of HMAC and SHA-1 which is not recommended anymore[82]. The solution to the issue came from a newer KDF algorithm named Argon2 which won the Password Hashing Competition against 24 other candidates[83]. Conveniently, the developer behind Signal implemented an Android and iOS wrapper to use the C implementation[84].

Theoretically, the generated master key would be imported in a HSM which would manage the encryption and decryption but there are some complications with the minimal supported Android API. Data encryption and decryption are handled internally by Android with the Android Keystore[85]. The Android Keystores has different ways to protect cryptographic key depending on the Android version and available hardware[86]. The weakest protection will only ensure that the cryptographic keys will not enter the mobile application process. When available, the cryptographic keys are bound to secure hardware (Trusted Execution Environment (TEE), Secure Element (SE), etc.). And starting with Android 9.0 (API 28), there is the possibility to use a HSM if it is available on the device[87]. The HSM is superior to the two other choices, because it has its own CPU, storage and a true random-number generator.

The master key will be used to encrypt and decrypt a randomly and securely generated DEK using AES-256. The DEK is a required addition due to the way that the internal SQLite database is encrypted. To encrypt the internal database, there is an SQLite extension library called SQLCypher that requires a passphrase at startup. Using the randomly generated DEK as passphrases, permits the application's data to be secured.

To further secure the access to encryption and decryption functions, the Android Keystore can be configured with user authentication to restrict their access. The most secure way is to prompt for biometrics before each encryption or decryption to assure that it is the smartphone owner that uses the application[88]. It also involves a default invalidation of stored cryptographic keys when new biometrics are added to the android device. Which would require the user to reenter its recovery phrase. It can become inconvenient from a perspective of user experience to only rely on biometrics in case where the sensors are not working properly. The option to invalidate keys on new biometrics would equal to the security of the following alternative.

The alternative way to do user authentication is to use the user's system-wide credentials which are for example used to authenticate the owner on the lock screen. Using this authentication makes the protection possibly weak as short pin or passwords are allowed.

There is unfortunately no way to authenticate against the Android Keystore with an application unique password. It could theoretically be possible by using software based keystores to store the encryption

key to the DEK but it would not benefit from the HSM. The software keystores would be unlockable by either a passphrase encrypted by the Android Keystore or a user password.

In either case, this thesis will restrict itself to use biometrics only, even if it requires users to reenter their recovery phrase when they invalidate their biometrics.

8.4 Backup and Recovery

To recover a mobile wallet, there are three critical pieces to put together (see Chapter 8.3 for more information):

- Encrypted SQLite database containing the user data
- Encrypted DEK to decrypt the SQLite database
- Recovery phrase to decrypt the DEK

Both the SQLite database and the encrypted DEK are stored in the applications user space.

The easiest solution for Backup is to use Google's Autobackup feature[89]. It enables to back up the application's user data automatically and restore them on a new device without any user interaction. But it comes with the following restrictions:

- A Google account must be present on the smartphone (Backup is made to Google Drive)
- Only the newest backup is saved
- Data cannot be larger than 25Mbs

Depending on the amount of credential and possible media content, it should be noted that 25Mbs, could at some point be too small to save user data. If the backup exceeds the data limit, it will simply not make the backup. Additionally, only the newest backup is saved which means that deleted credentials or connections will not be retrieved if the backup was overridden.

This feature is present per default and it will already help users that switch or reinstall Android devices but they will not be able to restore the backup on a possible future iOS implementation.

There is no guarantee for Autobackup to make backups and there should be another convenient cross-mobile solution without impacting the security of the user's data. A theoretically way to achieve it can be found in Chapter 10.3.6.

9 Issues

This chapter lists the issues encountered during the thesis.

Multiple issues needed direct patches made into AF-Go, this is why there is a fork of the project with branches containing fixes[90]. The cumulation of the fixes are merged into the `dev` branch, which is currently used to build the framework and the mobile bindings.

9.1 Interoperability and Transition to DIDComm V2

Starting November 2021, a new working group was started to coordinate development on frameworks for mobile usage[91]. During one of the first meetings, the goal was set to make all framework AIP 2.0 compatible on DID Comm V2. The first milestone should be to implement the out-of-band protocol until 31. December 2021.

The current interoperability can be envisioned on a website managed by the Aries community[30]. Interoperativity tests between multiple Aries frameworks are listed where different framework work together as issuers, holders or verifiers.

This will probably mean that there will be breaking changes in the future and the mobile wallet will also be affected. The changes will mostly affect the mobile SDK and internally used models.

9.2 Aries Framework Go

There were multiple instances during the thesis where blocking problems came up with AF-Go. Some issues have been addressed and others remain unfixed due to missing time to investigate.

9.2.1 Debugability

During the thesis, there were multiple times where it was necessary to debug the framework. Motivation was either that the given parameter was assumed to be correct but it was uncertain how the framework interpret them or times were the framework contained errors.

To be usable from a mobile platform, the framework is prepared by the `gomobile`[58] project. Having a compiled library inside a package with bindings makes it hard to debug properly. There was nothing found during the thesis to set up a source code viewer with debuggable breakpoints.

The used alternative was to use the internal logger to log as much information as possible to find issue sources and to do manual analysis of the supposed faulty code.

9.2.2 Failed Android Builds for 32 Bit Platforms

During the thesis, the compilation of AF-Go did not work out of the box. The compilation would fail without relevant error messages.

Fortunately, it was not a single case, as other users had the same issue[92]. The issue could be bypassed by limiting the build of the x64 platforms.

This issue makes the library incompatible with a limited number of smartphones. In the case of Samsung smartphones, there is only the Galaxy S5 (2014) that supports Android Marshmallow and runs under a 32-bit processor[93][94]. Entry-level smartphones up to 2016 either did not support Marshmallow or switched to x64 platforms.

But the incompatibility had more impact at the start of the implementation phase. The default Android emulator runs on x86 and another x64 Android image had to be downloaded to run the application with the embedded library.

Finally, later on for stability reasons, the development was done completely on hardware without emulation.

9.2.3 Missing and Faulty Documentation

AF-Go proposes some initial documentation to get started but it fast stops there. This is especially true for the REST API and the mobile bindings. For example, the generated OpenApi specification for the REST API does incompletely or even faultily describe APIs.

This made the object mapping generation of AF-Go method parameters and response impossible with existing tools like OpenApi Generator[95]. Communication between the mobile application and the AF-Go library are made in Json. As there were no exposed models by the library, it was necessary to duplicate the model definitions in the mobile application. The models had to be written by hand and the process easily introduced errors because the original models are described in multiple places in the framework. But in the end, the duplicated models made it possible to serialize and deserialize Json parameters and responses in a sane way. And they are only used internally in the mobile SDK.

It does not solve the issue that if some breaking change occurs, it is highly possible that a change will also be necessary in the mobile SDK. But there is currently no other solution other than to replicate the library's models in Kotlin.

9.2.4 Unpredictable Error on Method Calls

There is an issue with the implementation where sometimes the request to the library fails because of an unknown error. The error is triggered by an unknown character passed to the Go Json serializer but the character does not equal to what was provided by the mobile SDK. The calls to the library do work if the method call is repeated after the initial error[96].

There was not enough time to investigate the issue during the thesis but it would make the AF-Go more stable and remove complexity in the mobile SDK that manages this issue.

9.2.5 Sign Presentations

Signing presentations is crucial for the mobile wallet. They have to proof that it is really coming from the trusted DID Comm between them. But during the implementation, the AF-Go method did always result in an error due to missing data. The issue is documented on Github and the problem seems to be hardcoded values instead of the identifier of the key pair necessary to generate the signature[97].

The developers responded that a similar issued existed and a possible fix is underway[98]. Currently, for DID method compatibility, each DID Doc key id is saved into the internal Key Management System (KMS) as is.

A quick patch was made on the AF-Go fork to enable signatures, it could lead to compatibility issues in the future[99]. But to continue work on the mobile wallet, it was a necessary modification. As soon as the real correction is available, the custom patch will be dropped.

9.2.6 HTTP DID Resolver

There are an increasing number of DID methods and one of the easiest solution to resolve them is to use the DIF Universal Resolver. But there are some compatibility and assumption issues that came out after discussion with the AF-Go maintainers[100]. In short, the universal resolver project does some internal verification, validation and data normalization on received data from ledgers. AF-Go expected to receive valid Json-LD data since they implemented the feature, but it seems that the universal resolver behavior has changed since then.

Currently, the resolution works with a patch on the AF-Go fork. It does play with the content type of the request and has more relaxed validation on the Universal Resolver responses and received DID Doc [101]. The actual solution would be investigated in the Universal Resolver and elaborate on a fixed response type. It was not the focus of the master thesis to resolve the issues of the Universal Resolver.

9.2.7 Custom Message Registration

During the implementation of the chat function of the mobile wallet, it was impossible to register a new message service from the mobile application. It came out that it was an issue during the initialization of the framework on mobile.

The issue and a pull request were posted on the Github account and it was later merged into the main repository[102][103].

9.2.8 No Remote Json-LD fetching for mobile bindings

At the start of the thesis, it was not possible to accept Json-Ld files with context definition that were not loaded on the framework at startup. As it was seen as important for interoperability, a pull request was made to add the option to enable it and fetch remote context definition if they are not available locally[104]. The pull request was later merged into the main repository.

10 Evaluation and Testing

This chapter contains information about how the mobile wallet was tested, how its current feature set can be evaluated and what can be done in the future to help move the mobile wallet further.

10.1 Testing in General

As a disclaimer, testing in the mobile application is done manually. This is due to the nature of this project and the ratio between utility and complexity of testing client based applications.

Automated tests help to achieve functional goals and avoid side effects to existing code but this client-sided project relies on a framework that does the complex tasks. The mobile application does the coordination between the framework and the user interface. Automated testing is in this case can become quite complex for mobile applications and there is a high probability of redundant testing.

Due to the current immaturity of SSI, the frameworks that are in active development and the complexity of components to mock up correctly, it makes little sense to automate it even if there is a chance to introduce side effects later on. But it will be coherent to start implementing automated UI tests later when the SSI development becomes more stables and when more UX effort is made into the yet only functional mobile application.

10.1.1 Manual Test Scenarios

The current tests are done manually to ensure the correct implementation of the application. Each test has a list of steps to perform and some test may require two devices.

Possible step actors are:

- **Holder-A:** Device with mobile wallet number 1
- **Holder-B:** Device with mobile wallet number 2
- **Holders:** Both Holder-A and Holder-B
- **Issuer:** Issuer with a publicly available DID
- **Verifier:** Device with mobile wallet in verification mode

Table 6: Test Scenarios

Test descriptions	Steps
Create a connection between two wallets	1. **Holder-A:** shows the QR Invitation 2. **Holder-B:** scans the QR Invitation 3. **Holder-A:** shows the connection request • as pop up and accept • as pending action and accept 4. **Holders:** verify completed connection
Receive a verifiable credential	1. **Issuer** and **Holder-A:** create a connection 2. **Issuer:** sends credential offer 3. **Holder-A**: shows the credential offer • as pop up and request credential • as pending action and request credential 4. **Issuer:** accepts credential request and send credential 5. **Holder-A:** shows the credential • as pop up and accept • as pending action and accept 6. **Holder-A**: validates the received credential
Present a verifiable credential	1. **Holder-A:** receives a verifiable credential 2. **Holder-A:** shows QR invitation for proof presentation 3. **Verifier:** scans QR invitation 4. **Holder-A:** sends proof presentation offer 5. **Verifier:** sends proof presentation request 6. **Holder-A:** sends credential presentation 7. **Verifier:** verifies credential and presentation proof
Write a message to a peer over an existing connection	1. **Holders:** create a connection 2. **Holder-A:** sends a message over the connection 3. **Holder-B:** verifies that the message was received
Verify accuracy of the history feature	1. **Holders** and **Verifier:** do the tests above 2. **Holders** and **Verifier:** verify if events are accurate

10.2 State of the Mobile Wallet

To summarize the mobile wallet requirements from Chapter 6.3. The mobile wallet with SSI integration should be able to:

- ☒ Create a DID Comm connection between two peers (includes creating an invitation)
- ☒ Receive verifiable credentials
- ☐ Present verifiable credentials to other peers
    - ☒ Send complete verifiable presentations to other peers
    - ☐ Selective Disclosure and ZKP (see Chapter 10.3.4)
- ☒ Secure the storage of private keys and data
- ☒ Save and recover backuped mobile wallets

At the time of writing, the mobile wallet does implement every point mentioned above with exception of selective disclosure and ZKP which are currently based on an unfinished standard (see Chapter 10.3.4).

Mobile wallet owners can generate a connection invitation on their device and invite others to chat between them for example (see Figure 33).

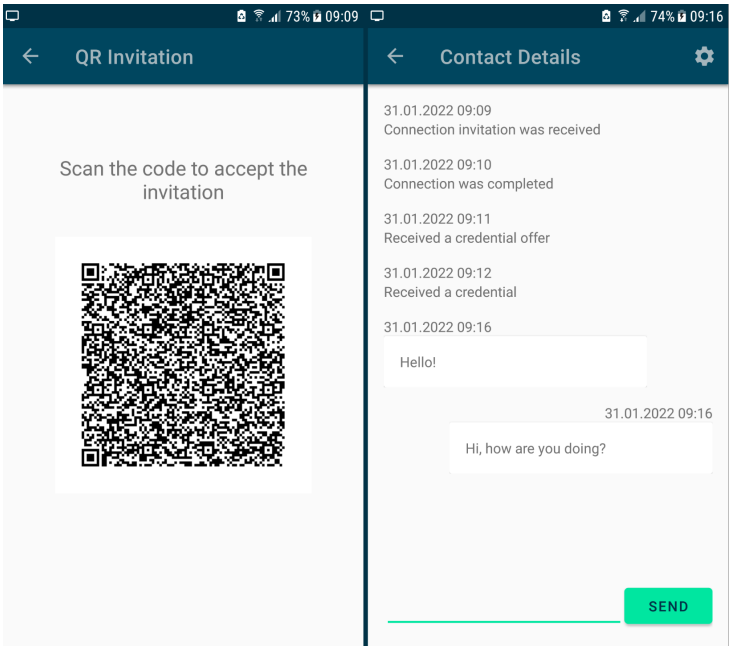

Figure 33: Mobile Wallet: Invite and Chat with Contacts

The wallet can receive VCs and send them to other peers to validate them(see Figure 34). But there are some limitation, due to the newer emergence of BBS+ signatures and their standard currently in draft.

It was not timely possible to implement privacy features such as selective disclosure and ZKP (more information in Chapter 10.3.4).

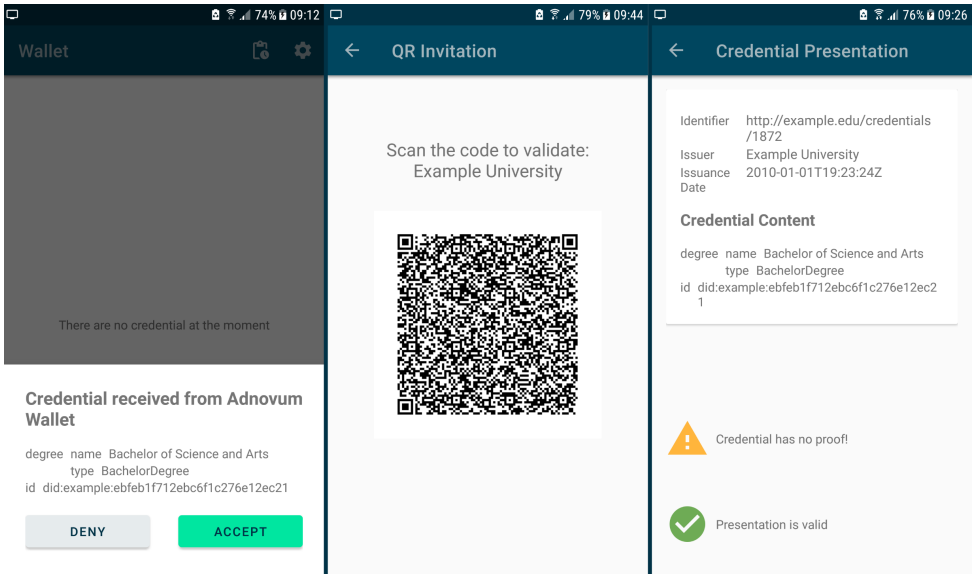

Figure 34: Mobile Wallet: Accept, Show and Validate a Credential

To secure the user's digital identity, the wallet encrypts the local storage and implement automatic backup process. The user can restore their identity by entering the recovery phrase on a new device or the same if it became insecure (see Figure 35).

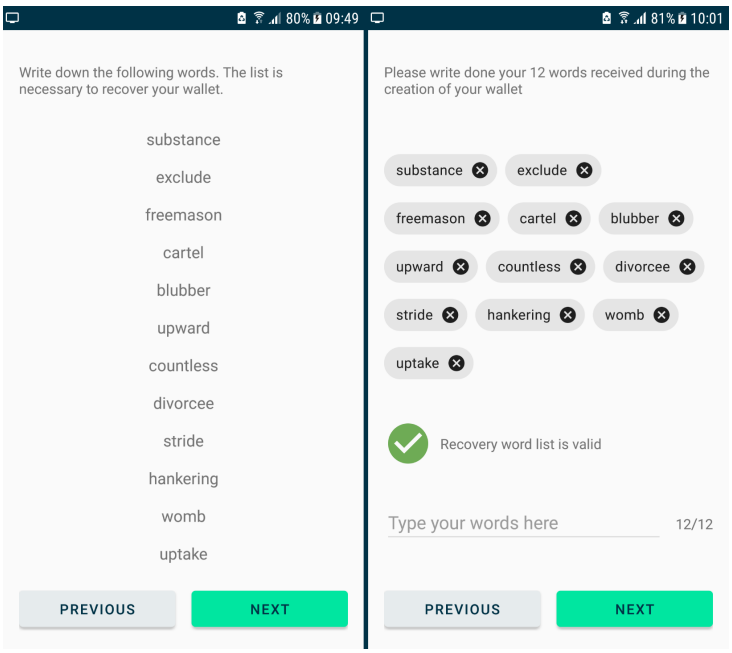

Figure 35: Mobile Wallet: Recovery Phrase and Recover Process

The complete collection mobile wallet screens can be found in Chapter 14.1.

10.3 Future Improvements and Work

The mobile wallet is not finished and there is enough room for improvements and additional tasks that would benefit users and / or developers. And depending on the wished quality level, it could also make it a publicly available SSI mobile wallet in the future.

10.3.1 Missing Message Handling

The mobile wallet does not include all problem handling and edge cases that are specified in the Aries RFC. Most implementations only handle the “Happy Path” for the use case in the physical world.

10.3.2 Insecure Connection to the Mediator

Currently, the mobile wallet connects with unsecured web sockets to the mediator. There should not be any issues with it, as the DID Comm is encrypted. Nevertheless, it would be advisable to use HTTPS or WSS to add an additional security layer that third party would have to break before accessing the actual DID Comm.

10.3.3 Transition to DIDComm V2

DIDComm V2 is the new standard defined by DIF and multiple frameworks are starting to prepare themselves to integrate it[91]. But currently only AF-Go has started and works actively on the seconde iteration. The development was not finished until the end of the thesis. Which means that a transition will be necessary at some point and will need adjustments in the mobile Aries package.

10.3.4 Selective Disclosure and ZKP with BBS+ Signatures

Selective disclosure and ZKP could not be implemented with BBS+ signatures during the master thesis. The BBS+ signature standard itself is currently in unofficial draft (version 13 June 2021) [24]. Understanding BBS+ signature and their usage could be a theme by itself and could be explored during a follow-up.

10.3.5 Mobile Application for iOS

Most applications come as a pair, one developed for Android and one for iOS to reach as many people as possible. In the case of a SSI mobile wallet, it would really be unfortunate to only distribute it only for one platform. For a realistic scenario of SSI mobile wallet distribution, an additional application for iOS should be developed.

10.3.6 Cross-Mobile Backup

Cross-mobile backups can be achieved by exporting manually the mobile wallet's content in a standardized and encrypted form. It would be managed by the user and could be imported easily into another device. But it lacks convenience as it is not automatic and rather time consuming to manage the backups.

It could be managed automatically by storing the backups on the mediator, as it is a critical infrastructure part for mobile wallets. The mediator could verify which backup belongs to whom by verifying signatures emitted with the private key coming from the recovery phrase. Inspiration comes from the Brave browser sync feature[105].

Cross-Mobile backups is a problem that is not solved currently, it needs a secure and hopefully convenient way to solve the issue. And the problem only becomes more complicated when in the vision of SSI it should also be exportable and interoperable with other mobile wallets.

10.3.7 Multi-Device Synchronization

Looking in the future, one of the missing blocks will also become synchronization between multiple devices. Use cases and possible devices that will be used with a SSI wallet can look up with the use of password managers. Both have the same goal, secure you digital identity and deliver proof when necessary. Most of the time, users have one password manager instance on their phone and a second one as browser plugin.

Synchronization becomes hard between them as in the best case, it should not be possible to correlate information and it should not be too easy to set up to avoid possible synchronization phishing attacks.

One solution proposed in Chapter 10.3.6 could help out. The same process could be used to synchronize devices. The only issue is that browser extensions do not have a secure storage. The browser plugin would need to store the private key on the machine itself via a local application with accessible APIs or in the cloud and access it with some form of authentication[106][107].

11 Conclusion

The goal of this thesis was to use SSI in the physical world. Manage and share your digital identity in your mobile wallet with people or machines near you.

Initially, existing mobile wallets working on online scenarios were hardly usable interactively between wallet owners. During the thesis, multiple SSI frameworks built on the in-work standards were compared to be integrated into a mobile application. Aries Framework Go was selected due to its support for the second iteration of the DID Comm protocol, as compared to the others. It was not the easiest framework to work with but it made the implementation of the mobile wallet possible. Several framework bottlenecks and bugs led to contributions to the framework by the author.

Even if the mobile wallet does not integrate all features that will be necessary to use the full potential of SSI (see Chapter 11.1), the result of the thesis meets the expectations at the outset. The mobile wallet can demonstrate SSI in the physical world, interact with other SSI internal infrastructures and secure user data with the highest security standards in the local HSM as well as an automated backup process.

SSI is still in the starting blocks and the involved standards need to mature to be able to implement scenarios like a national identifier (see Chapter 11.2).

11.1 SSI Today

SSI has had a great conceptual start with big ambitions and solid bases. However, the implementation is another story. There are multiple Aries frameworks that do work between themselves and to an extent with others.

The situation could be compared to the earlier years of web browsers. Not all implementations have the same roadmap to implement the available standards. Additionally, there are two versions of Aries RFC. AIP 1.0 has multiple mobile wallet implementations and also available infrastructure components. But currently, framework developers are working to implement AIP 2.0 and framework support is on track.

One of the current issues that need work before being usable are BBS+ signatures. With Hyperledger Indy, the VC needs a credential definition on the ledger and has a CL signature for every credential attribute[108]. It comes with the cost of being slow and signatures being large but it works and there are multiple Indy networks available for production. In counterparts, BBS+ needs more work on the standard, to be made viable for complete solutions.

Nevertheless, it is totally possible today to create a SSI solution with some caveats and limited features. The features and concepts of today's solutions are not always standardized and the community is

working on it[109]. One example would be a standardized way to enable push notifications on edge agents. Currently, each mobile wallet implementor has their own implementation but there is work done on that example[110].

11.2 SSI Tomorrow

SSI has a bright future and could solve headaches around internet security and data privacy. But to meet expectations, there is more work to be done on the implementation of the Aries Frameworks. There is more experimentation necessary to find out missing RFC features and to make it all work together. People start to implement more complete solutions based on SSI and will find missing concepts or rules to add in the future.

In the author's opinion, the most efficient route while being future-proof would be to drop AIP 1.0 support and concentrate on the second iteration that does not have the flaws and dependency of the first. But a more realistic short term solution would be to migrate to the stable AIP 2.0 RFCs (see Chapter 11.1). Standards for revocation and BBS+ signature are currently not usable, as opposed to AIP 1.0 technology, which is working.

An Aries Framework that proposes an agent implementation usable by servers, browsers and mobile devices would also significantly reduce development time and increase interoperability.

Finally, the author hopes that the contribution of this Master thesis result, which includes secure native storage on mobile devices and integration with the physical world, will help to advance the implementation of SSI and accelerate its wider adoption.

12 References

[1] K. Cameron, *The laws of identity*. 2005.

[2] C. K. Ameya Hanamsagar Simon S. Woo and J. Mirkovic, "How users choose and reuse passwords," 2016. https://www.isi.edu/publications/trpublic/pdfs/isi-tr-715.pdf (accessed Oct. 25, 2021).

[3] C. Allen, *The path to self-sovereign identity*. 2016.

[4] D. R. Alex Preukschat, *Self-sovereign identity*. Manning Publications Co., 2021.

[5] A. T. Tunggal, "The 59 biggest data breaches," 2021. https://www.upguard.com/blog/biggest-data-breaches (accessed Oct. 25, 2021).

[6] A. G. Johansen, "SIM swap fraud explained and how to help protect yourself," 2019. https://us.norton.com/internetsecurity-mobile-sim-swap-fraud.html (accessed Oct. 25, 2021).

[7] "Trust over ip foundation." https://trustoverip.org/ (accessed Oct. 25, 2021).

[8] "Decentralized identifiers (DIDs) v1.0." https://www.w3.org/TR/did-core/ (accessed Oct. 25, 2021).

[9] "DID methods." https://www.w3.org/TR/did-spec-registries/#did-methods (accessed Oct. 25, 2021).

[10] "Decentralized identity foundation." https://identity.foundation/ (accessed Oct. 25, 2021).

[11] "ION." https://github.com/decentralized-identity/ion-did-method (accessed Oct. 25, 2021).

[12] "ETHR DID method specification." https://github.com/decentralized-identity/ethr-did-resolver/blob/master/doc/did-method-spec.md (accessed Oct. 25, 2021).

[13] "VC data model by W3C." https://www.w3.org/TR/vc-data-model/ (accessed Nov. 03, 2021).

[14] "Verifiable presentation." https://www.w3.org/TR/vc-data-model/#presentations (accessed Nov. 03, 2021).

[15] “Drilling down: Co-development.” https://medium.com/decentralized-identity/drilling-down-co-development-in-the-open-765a86ab153f (accessed Nov. 02, 2021).

[16] “SSI architectural stack.” https://github.com/decentralized-identity/decentralized-identity.github.io/blob/master/assets/ssi-architectural-stack--and--community-efforts-overview.pdf (accessed Nov. 02, 2021).

[17] “Aries working group 17.11.2021.” https://wiki.hyperledger.org/display/ARIES/2021-11-17+Aries+Working+Group+Call (accessed Nov. 19, 2021).

[18] “DIDComm specification.” https://identity.foundation/didcomm-messaging/spec/ (accessed Nov. 03, 2021).

[19] “Brief overview and history.” https://medium.com/finema/anonymous-credential-part-1-brief-overview-and-history-c6679034c914 (accessed Nov. 03, 2021).

[20] “Zero-knowledge proofs explained.” https://www.expressvpn.com/blog/zero-knowledge-proofs-explained/ (accessed Nov. 03, 2021).

[21] “Internet identity workshop 31.” https://github.com/windley/IIW_homepage/raw/gh-pages/assets/proceedings/IIW_31_Book_of_Proceedings_1.pdf (accessed Nov. 03, 2021).

[22] “Why the verifiable credentials community should converge on BBS+.” https://www.evernym.com/blog/bbs-verifiable-credentials/ (accessed Nov. 03, 2021).

[23] “A solution for privacy-preserving verifiable credentials.” https://medium.com/mattr-global/a-solution-for-privacy-preserving-verifiable-credentials-f1650aa16093 (accessed Nov. 03, 2021).

[24] “BBS+ signature 2020.” https://w3c-ccg.github.io/ldp-bbs2020/ (accessed Nov. 03, 2021).

[25] “Understanding DIDComm.” https://medium.com/decentralized-identity/understanding-didcomm-14da547ca36b (accessed Nov. 03, 2021).

[26] “Protocols built on DIDComm.” https://didcomm.org/ (accessed Nov. 03, 2021).

[27] “Hyperledger aries.” https://github.com/hyperledger/aries (accessed Nov. 03, 2021).

[28] “Aries RFC.” https://github.com/hyperledger/aries-rfcs (accessed Nov. 03, 2021).

[29] “Aries interop profile.” https://github.com/hyperledger/aries-rfcs/tree/main/concepts/0302-aries-interop-profile (accessed Nov. 03, 2021).

[30] “Aries interoperability information.” https://aries-interop.info/ (accessed Nov. 03, 2021).

[31] “Zug ID: Exploring the first publicly verified blockchain identity.” https://medium.com/uport/zug-id-exploring-the-first-publicly-verified-blockchain-identity-38bd0ee3702 (accessed Oct. 25, 2021).

[32] “Veramo: uPort’s open source evolution.” https://medium.com/uport/veramo-uports-open-source-evolution-d85fa463db1f (accessed Oct. 25, 2021).

[33] “Evernym joins hyperledger.” https://medium.com/evernym/evernym-joins-hyperledger-94186fa22bef (accessed Nov. 03, 2021).

[34] “Hyperledger indy public blockchain.” https://wiki.hyperledger.org/download/attachments/20024919/Understanding%20Hyperledger%20Indy%20Ledger.pdf (accessed Nov. 04, 2021).

[35] https://vonx.io/ (accessed Nov. 03, 2021).

[36] “Sovrin.” https://sovrin.org/ (accessed Nov. 03, 2021).

[37] “Indicio.” https://indicio.tech/ (accessed Nov. 03, 2021).

[38] “IDunion.” https://idunion.org/ (accessed Nov. 03, 2021).

[39] “Sidetree specification.” https://identity.foundation/sidetree/spec/ (accessed Nov. 04, 2021).

[40] “KERI: For every DID, a microledger.” https://medium.com/decentralized-identity/keri-for-every-did-a-microledger-f9457fa80d2d (accessed Nov. 04, 2021).

[41] “KERI: A more performant ledger for trusted identities.” https://medium.com/spherity/introducing-keri-8f50ed1d8ed7 (accessed Nov. 04, 2021).

[42] “XMPP server.” https://xmpp.org/software/servers/ (accessed Nov. 04, 2021).

[43] “White label trinsic’s popular mobile wallet.” https://trinsic.id/white-label/ (accessed Dec. 16, 2021).

[44] “Trinsic wallet.” https://play.google.com/store/apps/details?id=id.streetcred.apps.mobile&hl=en_US&gl=US (accessed Nov. 04, 2021).

[45] “Esatus wallet.” https://play.google.com/store/apps/details?id=com.esatus.wallet&hl=en_US&gl=US (accessed Nov. 04, 2021).

[46] “Evernym wallet.” https://play.google.com/store/apps/details?id=me.connect&hl=en_US&gl=US (accessed Nov. 04, 2021).

[47] “Digitaler führerschein: ID wallet mit enormen startschwierigkeiten.” https://t3n.de/news/digitaler-fuehrerschein-id-schwierigkeiten-1410191/ (accessed Nov. 04, 2021).

[48] “Verified.me google play store.” https://play.google.com/store/apps/details?id=com.securekey.verifiedme (accessed Nov. 04, 2021).

[49] “Veramo documentation.” https://web.archive.org/web/*/https://veramo.io/docs/basics/introduction (accessed Nov. 18, 2021).

[50] “Aries interop profile 1.0.” https://github.com/hyperledger/aries-rfcs/tree/main/concepts/0302-aries-interop-profile#aries-interop-profile-version-10 (accessed Nov. 19, 2021).

[51] “Aries interop profile 2.0.” https://github.com/hyperledger/aries-rfcs/tree/main/concepts/0302-aries-interop-profile#aries-interop-profile-version-20 (accessed Nov. 19, 2021).

[52] “Aries mobile summit 16.11.2021.” https://wiki.hyperledger.org/display/ARIES/2021-11-16+Aries+Summit+Session (accessed Nov. 19, 2021).

[53] “Acapy interoperability efforts.” https://github.com/hyperledger/aries-framework-go/search?q=ACAPyInterop (accessed Nov. 19, 2021).

[54] “What is a chain of trust.” https://www.ssl.com/faqs/what-is-a-chain-of-trust/ (accessed Nov. 19, 2021).

[55] “Global legal entity identifier foundation.” https://www.gleif.org/en/ (accessed Nov. 19, 2021).

[56] “Veramo community.” https://github.com/uport-project/veramo/discussions/733 (accessed Nov. 18, 2021).

[57] “Aries framework go commits.” https://github.com/uport-project/veramo/discussions/733 (accessed Nov. 18, 2021).

[58] “GoMobile command.” https://pkg.go.dev/golang.org/x/mobile/cmd/gomobile (accessed Nov. 22, 2021).

[59] “Android version distribution statistics will now only be available in android studio.” https://www.xda-developers.com/android-version-distribution-statistics-android-studio/ (accessed Dec. 08, 2021).

[60] “Aries RFC 0434: Out-of-band protocol 1.1.” https://github.com/hyperledger/aries-rfcs/tree/main/features/0434-outofband (accessed Nov. 24, 2021).

[61] “Aries RFC 0211: Mediator coordination protocol.” https://github.com/hyperledger/aries-rfcs/tree/main/features/0211-route-coordination (accessed Nov. 24, 2021).

[62] “Aries RFC 0092: Transports return route.” https://github.com/hyperledger/aries-rfcs/tree/main/features/0092-transport-return-route (accessed Nov. 24, 2021).

[63] “Aries RFC 0453: Issue credential protocol 2.0.” https://github.com/hyperledger/aries-rfcs/tree/main/features/0453-issue-credential-v2 (accessed Nov. 24, 2021).

[64] “Aries RFC 0454: Present proof protocol 2.0.” https://github.com/hyperledger/aries-rfcs/tree/main/features/0454-present-proof-v2 (accessed Nov. 24, 2021).

[65] “DIDComm bluetooth standard.” https://github.com/decentralized-identity/didcomm-bluetooth/blob/main/spec.md (accessed Nov. 25, 2021).

[66] “Transfer data between iOS and android via bluetooth?” https://stackoverflow.com/questions/18884705/transfer-data-between-ios-and-android-via-bluetooth (accessed Nov. 25, 2021).

[67] “Do new iPhones (iOS 13) support NFC in peer-to-peer mode?” https://apple.stackexchange.com/questions/379527/do-new-iphones-ios-13-support-nfc-in-peer-to-peer-mode (accessed Nov. 25, 2021).

[68] “iOS and wi-fi direct.” https://developer.apple.com/forums/thread/12885 (accessed Nov. 25, 2021).

[69] C. E. Shannon, “A mathematical theory of communication,” *Bell System Technical Journal*, 1948.

[70] “Deep dive: EFF’s new wordlists for random passphrases.” https://www.eff.org/deeplinks/2016/07/new-wordlists-random-passphrases (accessed Dec. 08, 2021).

[71] “How big is 2**128?” https://bugcharmer.blogspot.com/2012/06/how-big-is-2128.html (accessed Dec. 15, 2021).

[72] “Amount of simple operations that is safely out of reach for all humanity?” https://security.stackexchange.com/questions/6141/amount-of-simple-operations-that-is-safely-out-of-reach-for-all-humanity/6149#6149 (accessed Dec. 15, 2021).

[73] “Navigation.” https://developer.android.com/guide/navigation (accessed Dec. 16, 2021).

[74] “Kotlin coroutines on android.” https://developer.android.com/kotlin/coroutines (accessed Dec. 16, 2021).

[75] “LiveData.” https://developer.android.com/reference/kotlin/androidx/lifecycle/LiveData (accessed Dec. 16, 2021).

[76] “Kotlin flows on android.” https://developer.android.com/kotlin/flow (accessed Dec. 16, 2021).

[77] “StateFlow and SharedFlow.” https://developer.android.com/kotlin/flow/stateflow-and-sharedflow (accessed Dec. 16, 2021).

[78] “Event bus pattern with kotlin SharedFlow.” https://medium.com/tech-takeaways/how-to-implement-the-event-bus-pattern-with-kotlin-sharedflow-in-your-android-app-768529828607 (accessed Dec. 16, 2021).

[79] “Dependency injection with hilt.” https://developer.android.com/training/dependency-injection/hilt-android#hilt-and-dagger (accessed Dec. 16, 2021).

[80] “SecureRandom documentation.” https://developer.android.com/reference/java/security/SecureRandom (accessed Dec. 15, 2021).

[81] “SecretKeyFactory documentation.” https://developer.android.com/reference/javax/crypto/SecretKeyFactory (accessed Jan. 19, 2022).

[82] “NIST SP800-57.” https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf (accessed Jan. 19, 2022).

[83] “Password hashing competition.” https://www.password-hashing.net/ (accessed Jan. 19, 2022).

[84] “Argon2 wrapper developed by signal.” https://github.com/signalapp/Argon2 (accessed Jan. 19, 2022).

[85] “Android keystore documentation.” https://developer.android.com/training/articles/keystore (accessed Dec. 15, 2021).

[86] “Android keystore exctraction prevention.” https://developer.android.com/training/articles/keystore#ExtractionPrevention (accessed Dec. 15, 2021).

[87] “Hardware-backed keystore.” https://source.android.com/security/keystore (accessed Dec. 15, 2021).

[88] “How you should secure your android’s app biometric authentication.” https://medium.com/csg-govtech/how-you-should-secure-your-androids-app-biometric-authentication-10d9231215e4 (accessed Dec. 15, 2021).

[89] “Back up user data with auto backup.” https://developer.android.com/guide/topics/data/autobackup (accessed Dec. 15, 2021).

[90] “Aries framework go fork.” https://github.com/adn-misa/aries-framework-go (accessed Jan. 17, 2022).

[91] “Aries summit session 2021-11-16.” https://wiki.hyperledger.org/display/ARIES/2021-11-16+Aries+Summit+Session (accessed Dec. 13, 2021).

[92] “Cannot build android bindings.” https://github.com/hyperledger/aries-framework-go/issues/2890 (accessed Dec. 13, 2021).

[93] “Samsung galaxy S5 specification.” https://www.gsmarena.com/samsung_galaxy_s5-6033.php (accessed Dec. 13, 2021).

[94] “Snapdragon 801 processor.” https://www.qualcomm.com/products/snapdragon-processors-801 (accessed Dec. 13, 2021).

[95] “OpenAPI generator.” https://openapi-generator.tech/ (accessed Jan. 17, 2022).

[96] “Cannot query connections and queries with parameters return nothing on mobile agent.” https://github.com/hyperledger/aries-framework-go/issues/3009 (accessed Dec. 13, 2021).

[97] “Can not generate a presentation by credential id.” https://github.com/hyperledger/aries-framework-go/issues/3103 (accessed Dec. 13, 2021).

[98] https://github.com/hyperledger/aries-framework-go/pull/3049 (accessed Jan. 20, 2022).

[99] “Replace hardcoded KMS key identifier with kid (key id) from the KMS.” https://github.com/adn-misa/aries-framework-go/commit/5531c69a60a0020b4a15b15283be9effae108e79 (accessed Jan. 20, 2022).

[100] “Cannot resolve DID with universal resolver due to missing context.” https://github.com/hyperledger/aries-framework-go/issues/3062 (accessed Dec. 13, 2021).

[101] “Temporary fix issues with universal resolver.” https://github.com/adn-misa/aries-framework-go/commit/f62e9e048eb2dc17afc1609f87a9d6475140b6bd (accessed Jan. 20, 2022).

[102] “Issue with the message service registration on mobile.” https://github.com/hyperledger/aries-framework-go/issues/3083 (accessed Dec. 13, 2021).

[103] “Fix the message service issue on mobile.” https://github.com/hyperledger/aries-framework-go/pull/3086 (accessed Dec. 13, 2021).

[104] “Add option to enable remote document loading for mobile bindings.” https://github.com/hyperledger/aries-framework-go/pull/3134 (accessed Jan. 20, 2022).

[105] “How does brave sync work?” https://web.archive.org/web/20211217080552/https://www.reddit.com/r/brave_browser/comments/bty9hb/how_does_brave_sync_work/ (accessed Dec. 17, 2021).

[106] “Chrome.platformKeys.” https://developer.chrome.com/docs/extensions/reference/platformKeys/ (accessed Dec. 17, 2021).

[107] “Chrome.identity.” https://developer.chrome.com/docs/extensions/reference/identity/ (accessed Dec. 17, 2021).

[108] “Selective disclosure and CL signature.” https://medium.com/finema/anonymous-credential-part-2-selective-disclosure-and-cl-signature-b904a93a1565 (accessed Jan. 24, 2022).

[109] “Aries summit 23.11.2021.” https://wiki.hyperledger.org/display/ARIES/2021-11-23+Aries+Summit+Session (accessed Jan. 20, 2022).

[110] “Native push notifications.” https://github.com/hyperledger/aries-rfcs/pull/699 (accessed Jan. 20, 2022).

13 Glossary

ACA-Py Aries Cloud Agent Python. 43, 44

AF-.NET Aries Framework for .NET. 43, 44

AF-Go Aries Framework Go. 43, 44, 47, 48, 49, 56, 60, 62, 63, 66, 67, 68, 75, 114, 116

AFJ Aries Framework JavaScript. 43

AIP Aries Interop Profile. 43, 44, 47, 50, 66, 77, 78

BLE Bluetooth Low Energy. 54, 55

CAS Content-Addressable Storage. 29, 30

DEK Data Encryption Key. 63, 64, 65

DID Decentralized Identifier. 18, 19, 22, 23, 25, 28, 29, 30, 31, 42, 44, 45, 46, 48, 68

DID Comm DID Communication. 19, 22, 23, 25, 26, 27, 28, 32, 42, 47, 49, 54, 55, 60, 66, 68, 72, 74, 77

DID Doc DID Document. 18, 19, 20, 22, 23, 28, 29, 31, 68

DIF Decentralized Identity Foundation. 18, 19, 25, 26, 27, 28, 29, 31, 48, 68, 75

DPKI Decentralized Public Key Infrastructure. 43

EFF Electronic Frontier Foundation. 58, 63

FADP Federal Act on Data Protection. 9

GDPR General Data Protection Regulation. 9

HSM Hardware Security Module. 48, 58, 59, 64, 65, 77

IdP Identity Provider. 9, 17

IPFS InterPlanetary File System. 30

KDF Key Derivation Function. 58, 64

KERI Key Event Receipt Infrastructure. 30

KMS Key Management System. 68

KYC Know Your Customer. 37

NFC Near Field Communication. 36, 55

OIDC OpenID Connect. 14

P2P Peer to Peer. 28, 40

PoC Proof of Concept. 27, 28

SE Secure Element. 64

SSI Self Sovereign Identity. 8, 9, 15, 18, 21, 23, 24, 25, 26, 27, 28, 31, 32, 34, 35, 36, 40, 41, 42, 44, 46, 47, 48, 57, 60, 70, 72, 74, 75, 76, 77, 78, 116

TEE Trusted Execution Environment. 64

VC Verifiable Credential. 15, 16, 17, 18, 20, 21, 22, 23, 25, 26, 27, 28, 29, 31, 32, 33, 36, 37, 38, 42, 44, 45, 46, 49, 52, 53, 72, 77

VCX AriesVCX. 43

VP Verifiable Presentation. 52

ZKP Zero-Knowledge Proof. 26, 27, 72, 73, 75

14 Appendices

This chapter contains supplementary informations to accompany this thesis.

14.1 Appendix A: Application Screens

This appendix contains the collection of available screens in the mobile wallet produced in this thesis.

The whole application is available in two themes, dark and light theme (see Figure 36) which are shown depending on the system preference of the user. For simplicity, only the light theme will be presented in the following chapters.

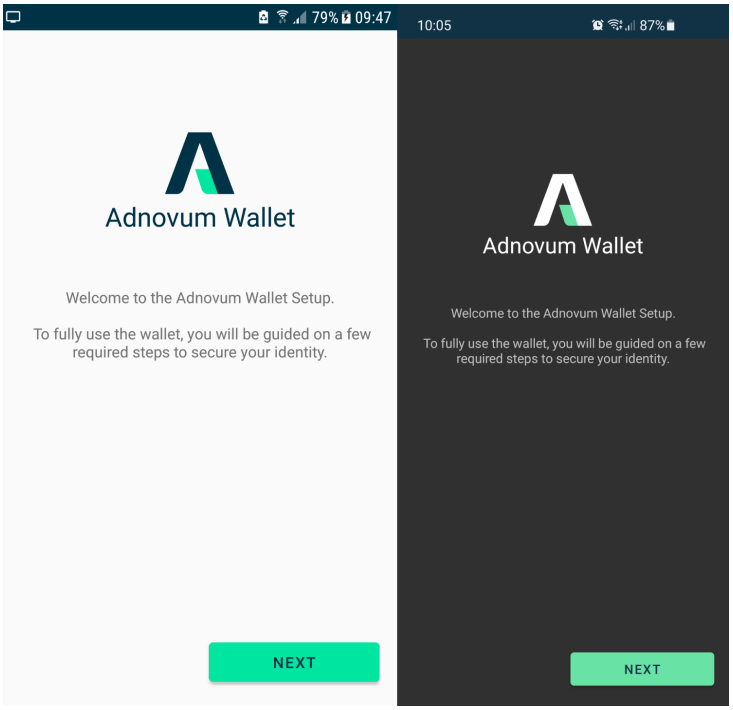

Figure 36: Mobile Wallet: Light and Dark Theme

14.1.1 Mobile Wallet Navigation

In Figure 37, each block represents a sub-chapter and the link between them represent possible navigation routes that a user can take depending on their actions and context (e.g., at the first start, the user will be redirected to the setup process).

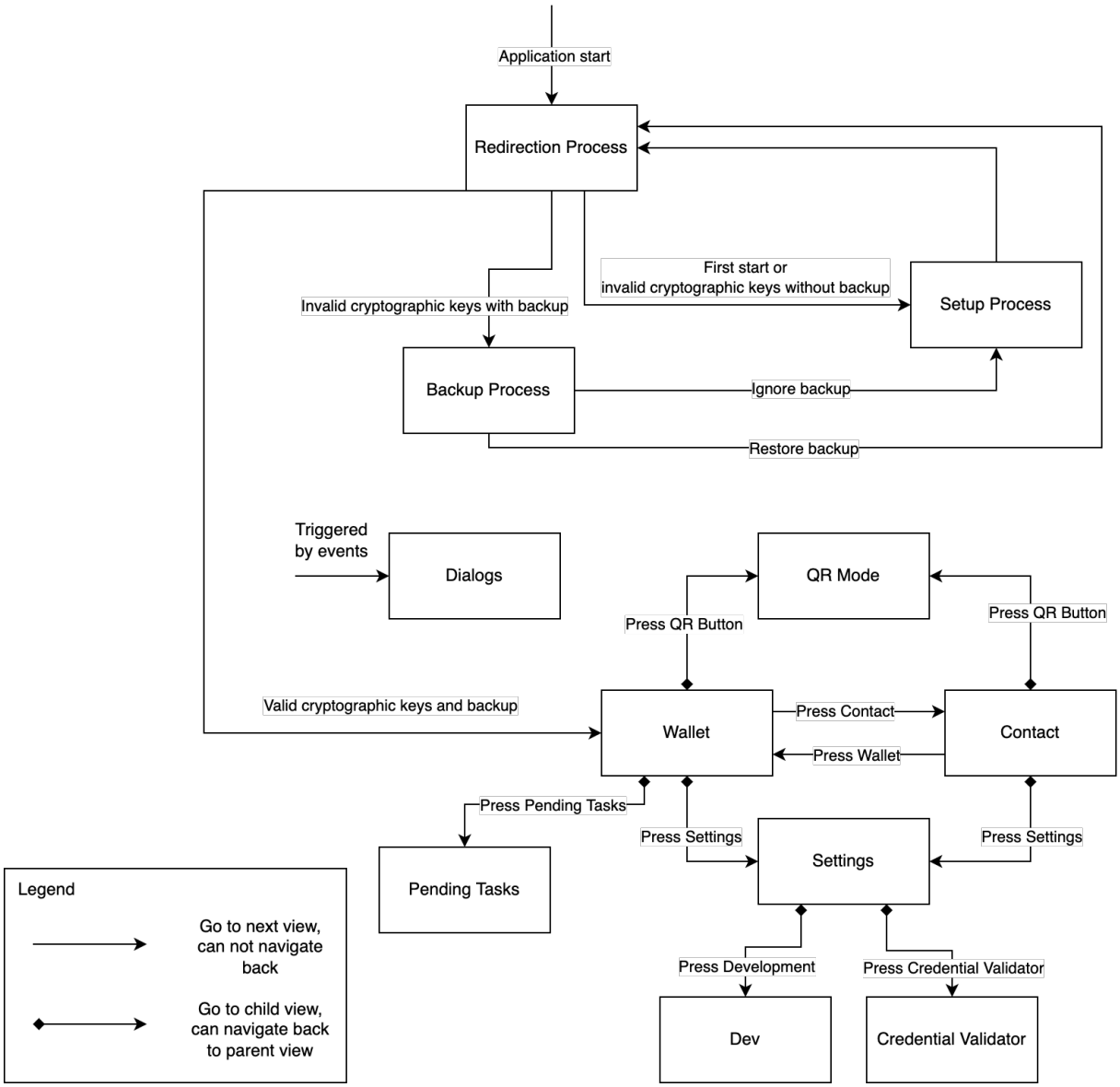

Figure 37: Mobile Wallet: Navigation Diagram

14.1.2 Setup Process

The first time the mobile wallet is started (see Figure 38), the user is guided on multiple steps to create their mobile wallet. They first have to write down their recovery passphrase (see Figure 39) and validate it by reordering the last six words of their passphrase (see Figure 40).

After validation, the user is prompted to enter their biometrics used later to open their mobile wallet(see Figure 41). Once completed, the user is redirected to the application redirection process (see

Chapter 14.1.4).

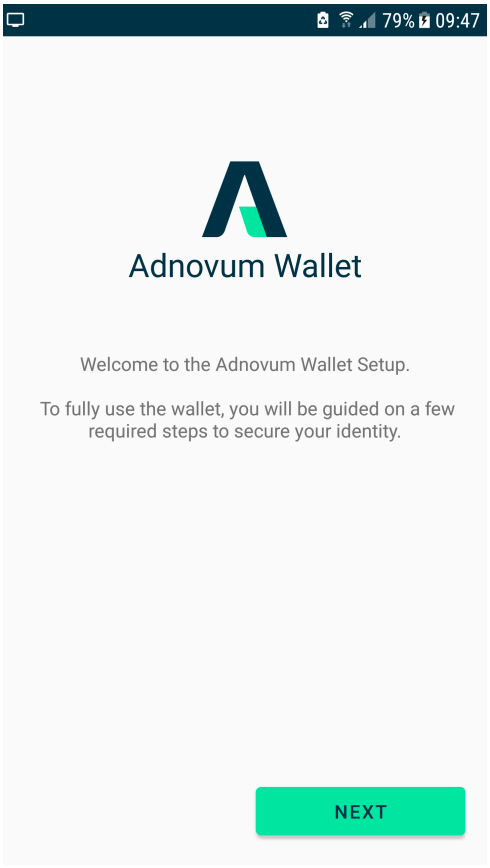

Figure 38: Mobile Wallet: Setup Start Page

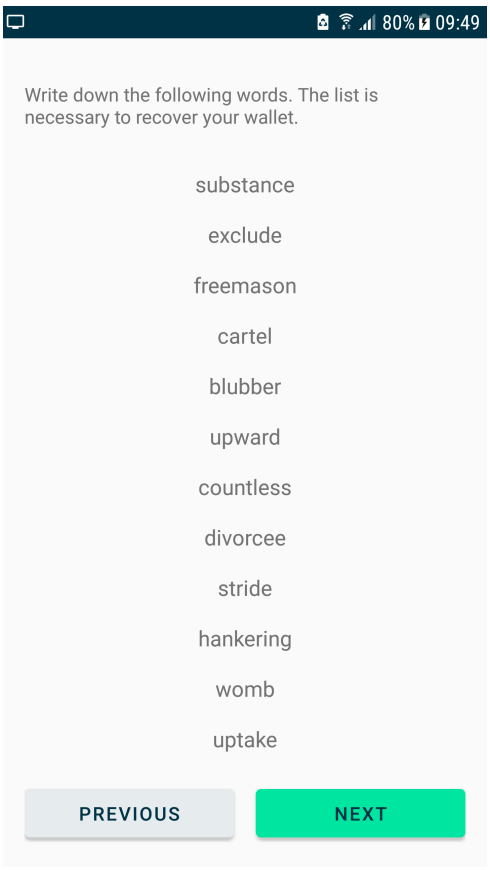

Figure 39: Mobile Wallet: Recovery Passphrase

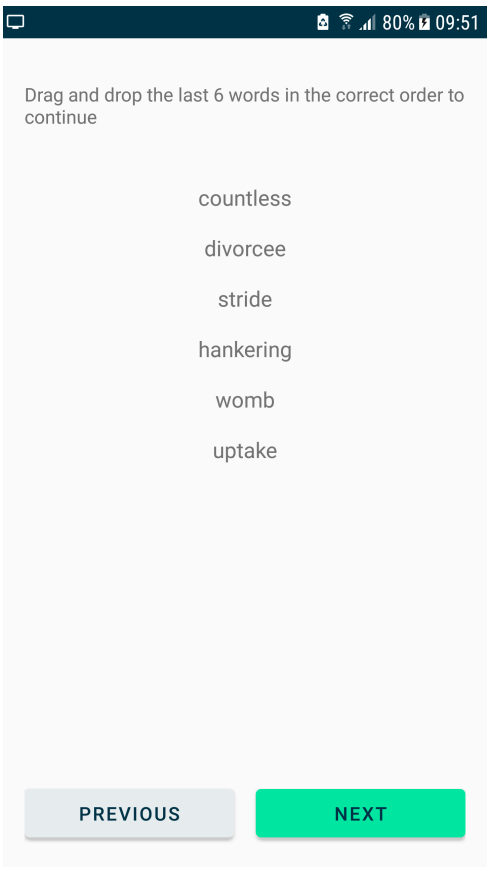

Figure 40: Mobile Wallet: Passphrase Validation

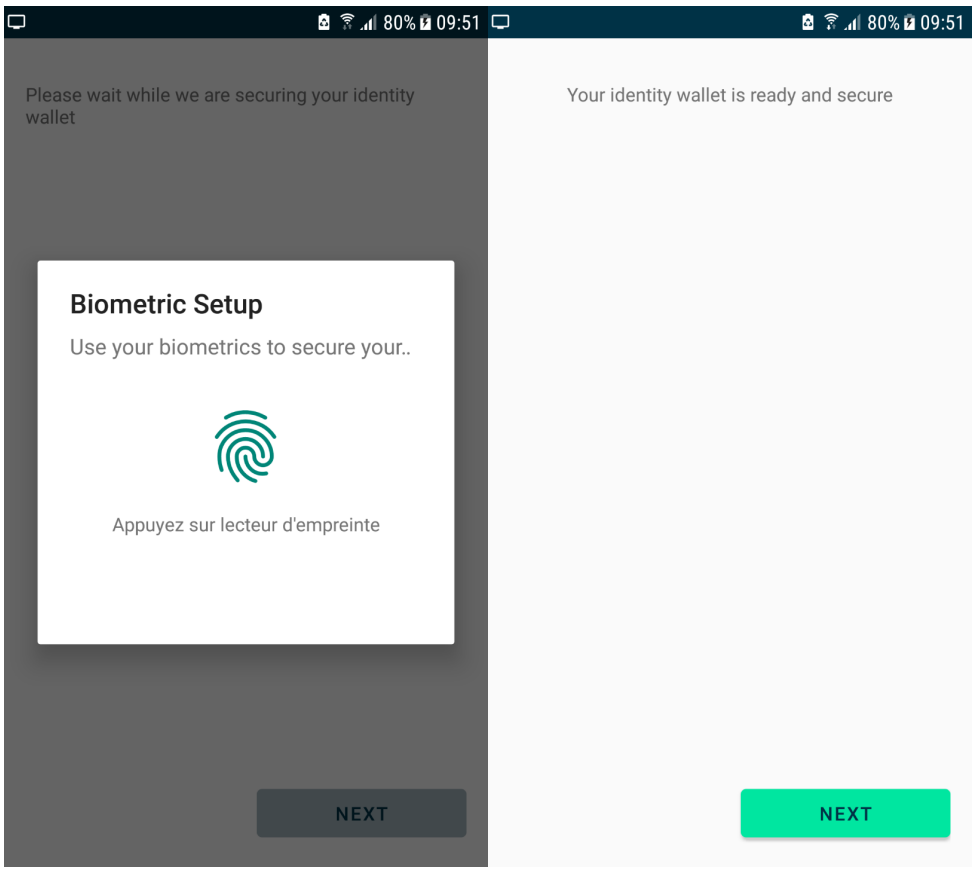

Figure 41: Mobile Wallet: Setup Biometric Input

14.1.3 Recovery Process

During the start-up, when the decryption key is not found or invalid, the user receives a similar screen to the setup start (see Chapter 14.1.2) but gets the option to recover the data from their mobile wallet (see Figure 42).

The user is asked to input their recovery passphrase in the right order by clicking on word suggestion (see Figure 43). When 12 words are entered, a message comes up to signal to the user if the passphrase is valid or not (Figure 44).

On a valid passphrase, the “Next” button is enabled and the restoration begins with the user’s biometric input (see Figure 45).

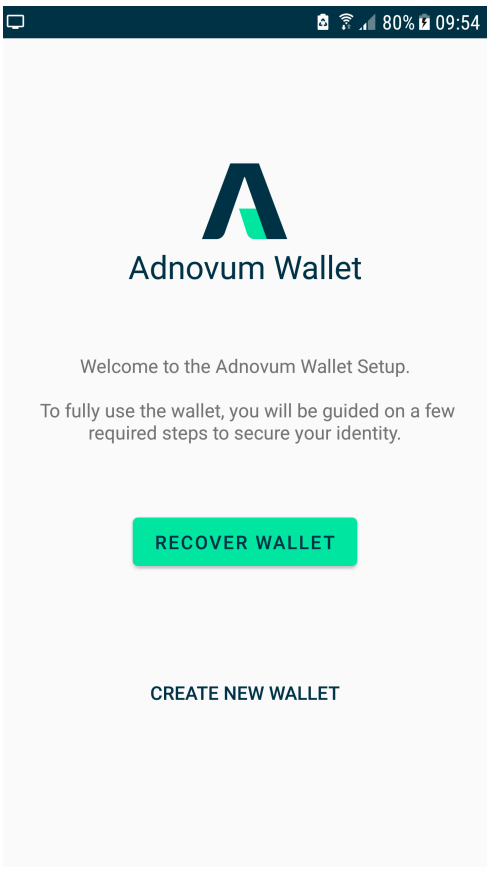

Figure 42: Mobile Wallet: Recovery Start Page

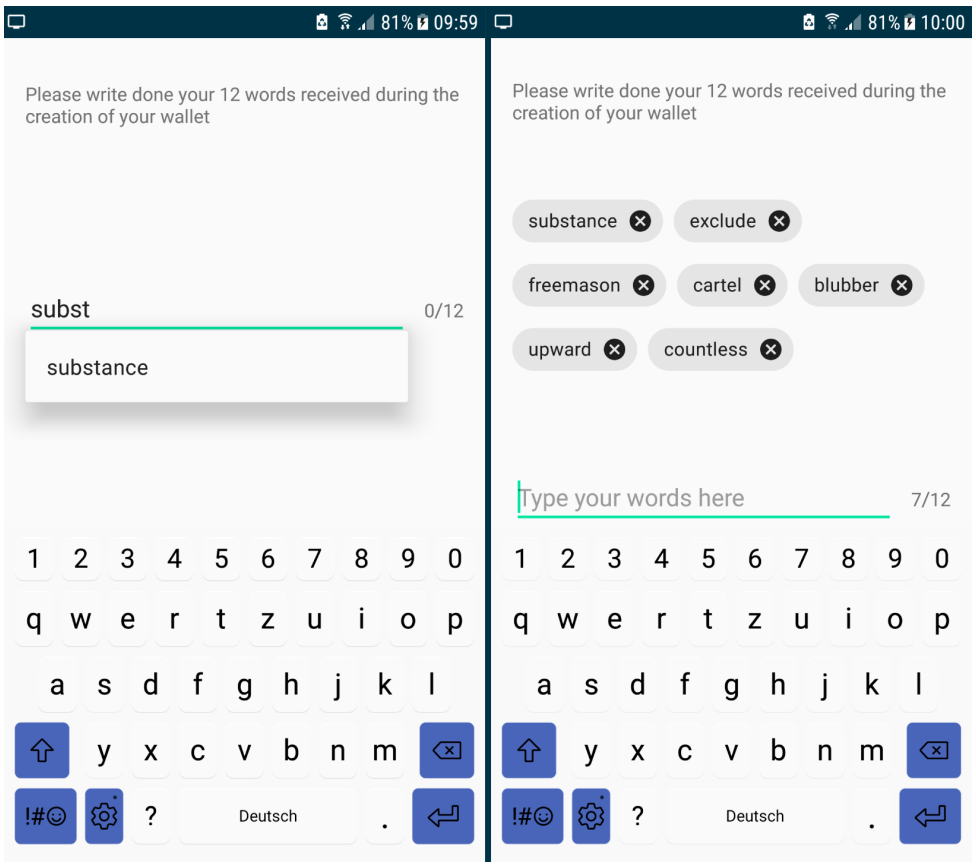

Figure 43: Mobile Wallet: Recovery Passphrase Input

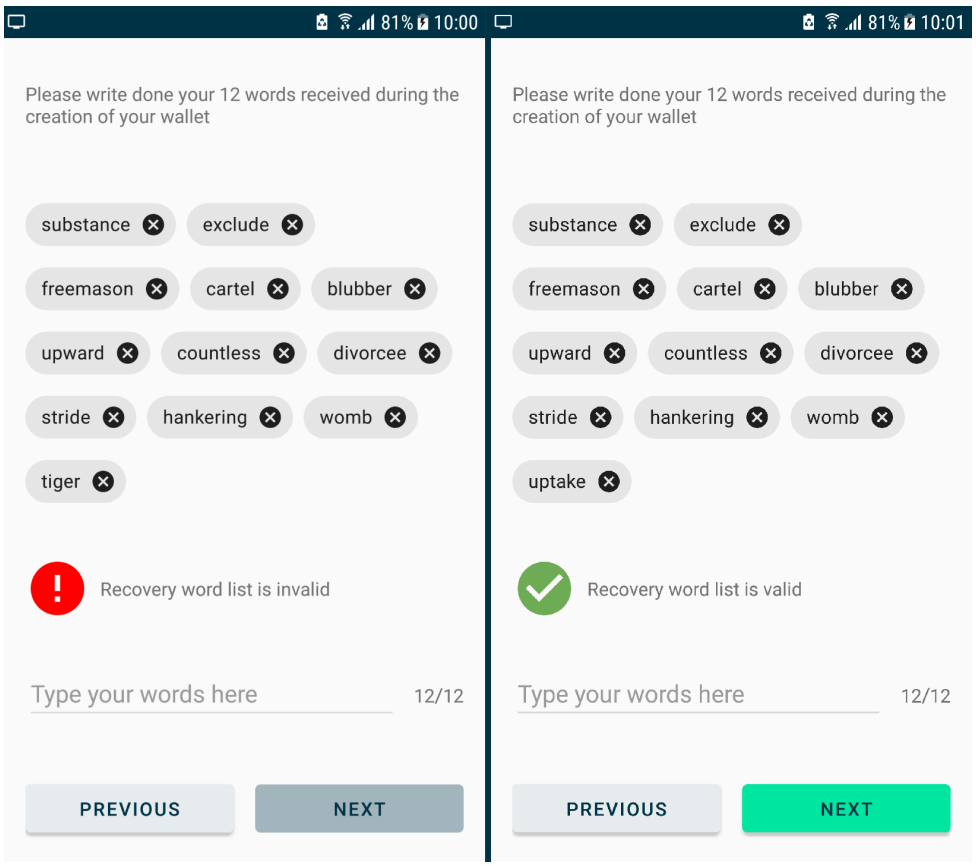

Figure 44: Mobile Wallet: Recovery Passphrase Validation

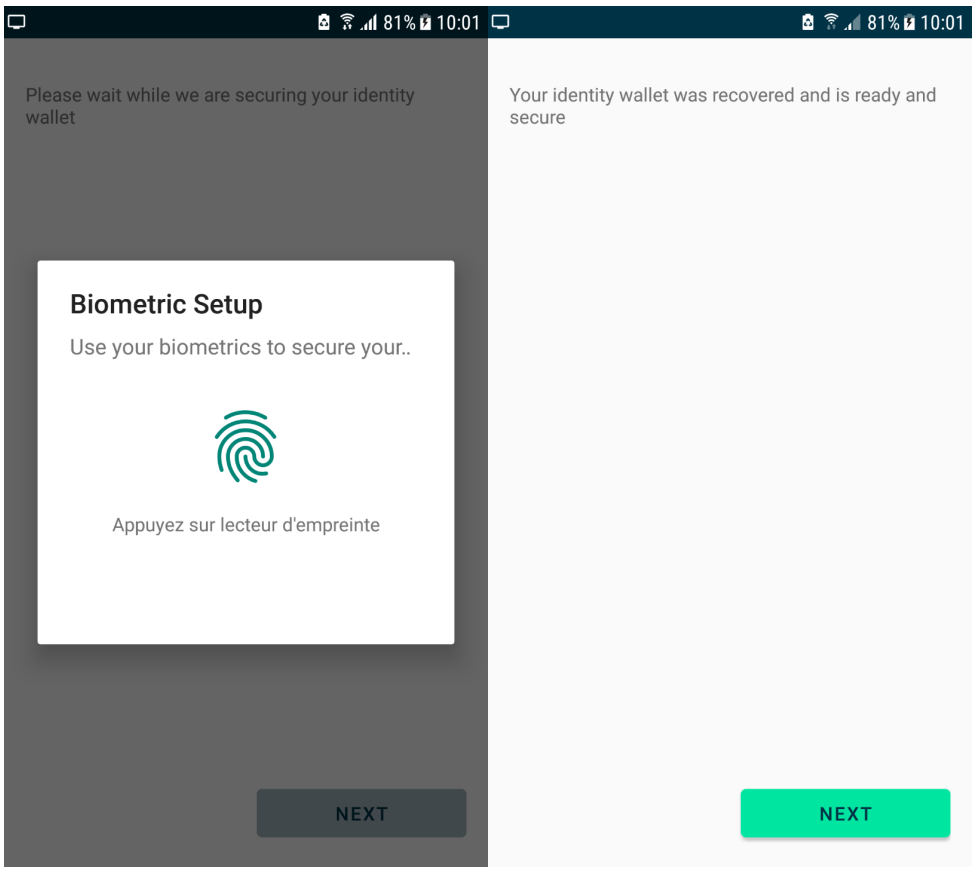

Figure 45: Mobile Wallet: Recovery Biometric Input

14.1.4 Redirection Process

At each application startup, the mobile wallet verifies in which mode the mobile wallet should start (see Figure 46). There are three modes available:

- Setup starts if the mobile wallet is opened the first time (see Chapter 14.1.2)
- Recovery starts if there are backup files but no valid cryptographic key material for the mobile wallet (see Chapter 14.1.3)
- Normal mode starts if everything is available and shows the user their credentials (Chapter 14.1.5).

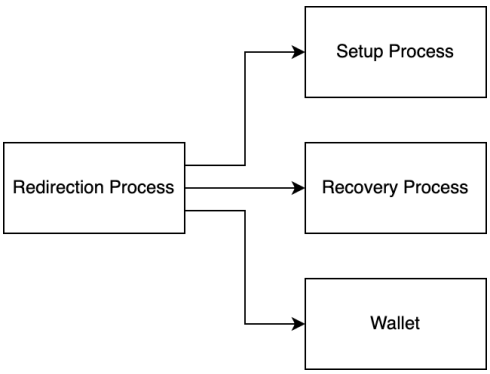

Figure 46: Mobile Wallet: Redirection Process

14.1.5 Wallet

The wallet screen shows a list of the users credentials (see Figure 47). Each credential has a shortcut button to display their validation QR code (see Figure 48) or to go into the credential's details by clicking else where on the credential (see Figure 49).

From the wallet, it is possible to go to the pending tasks(see Chapter 14.1.9) and settings(see Chapter 14.1.10) on the upper action bar. A user can also go to the contact screen(see Chapter 14.1.6) or the QR scan mode(see Chapter 14.1.7) from the bottom navigation bar.

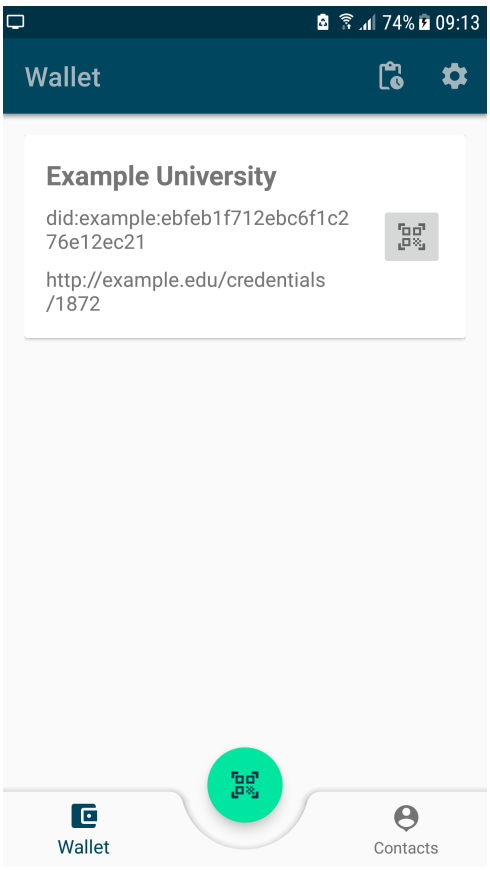

Figure 47: Mobile Wallet: Wallet Screen

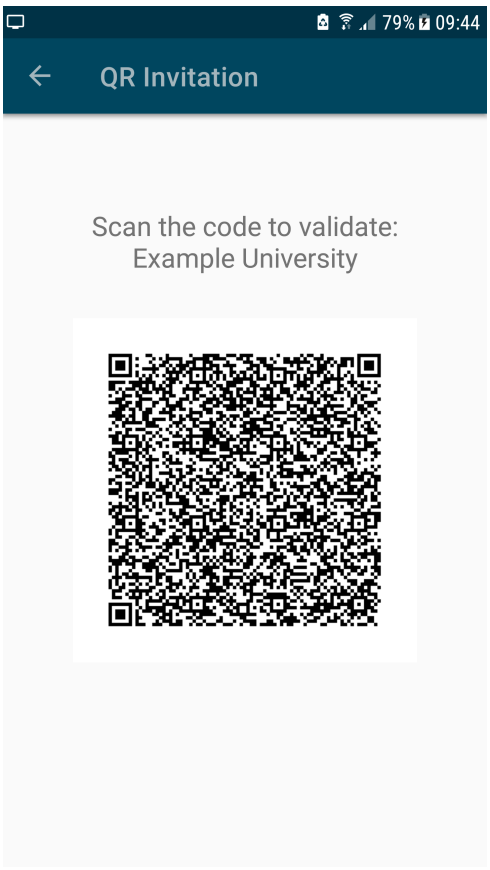

Figure 48: Mobile Wallet: Credential Validation Screen

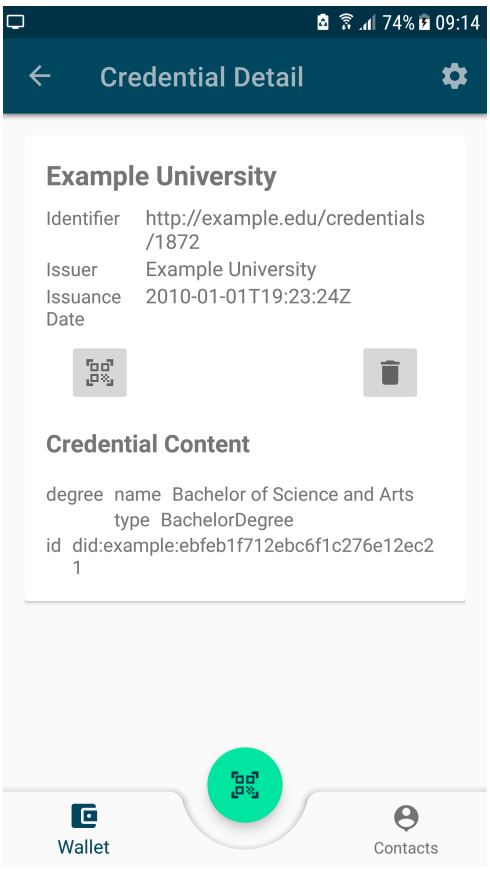

Figure 49: Mobile Wallet: Credential Detail Screen

14.1.6 Contact

On the contact screen, is a list of all connections made with the mobile wallet(see Figure 50). Contacts can be deleted by sliding it to the right (see Figure 51). By clicking on a contact, the detail view shows up and contains passed events and messages with the other contact(see Figure 52). Messages can be received and sent from the contact detail screen.

To invite a new contact, there is the outlined button “INVITE” on the top right. It shows an invitation QR code(see Figure 53) that other mobile wallet holders can scan to connect to the mobile wallet owner.

From the contact screen, it is possible to go to the settings(see Chapter 14.1.10) on the upper action bar. A user can also go to the wallet screen(see Chapter 14.1.5) or the QR scan mode(see Chapter 14.1.7) from the bottom navigation bar.

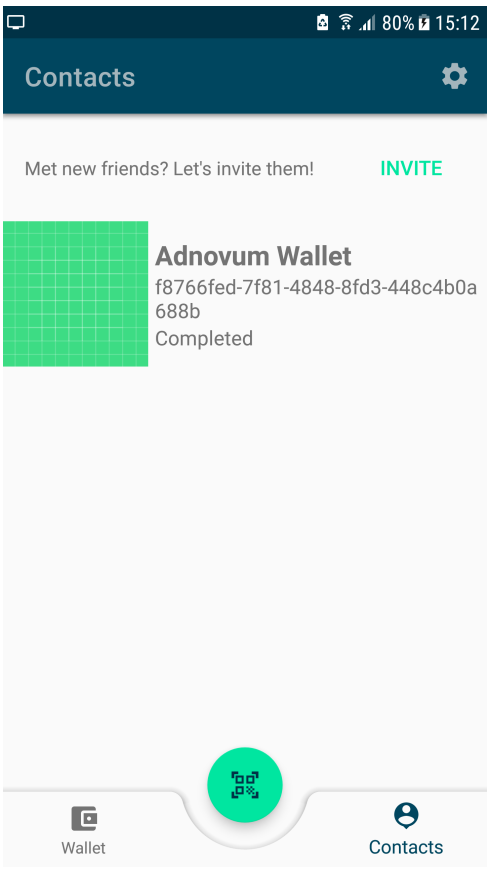

Figure 50: Mobile Wallet: Contact Screen

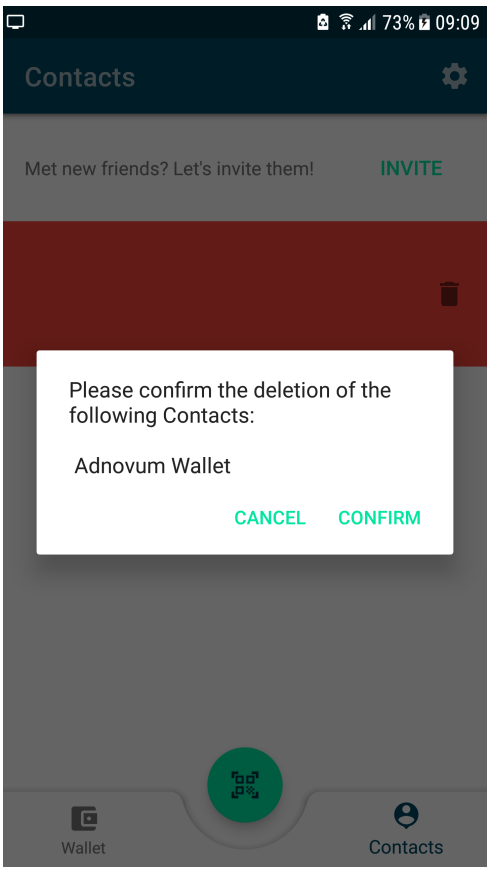

Figure 51: Mobile Wallet: Contact Deletion

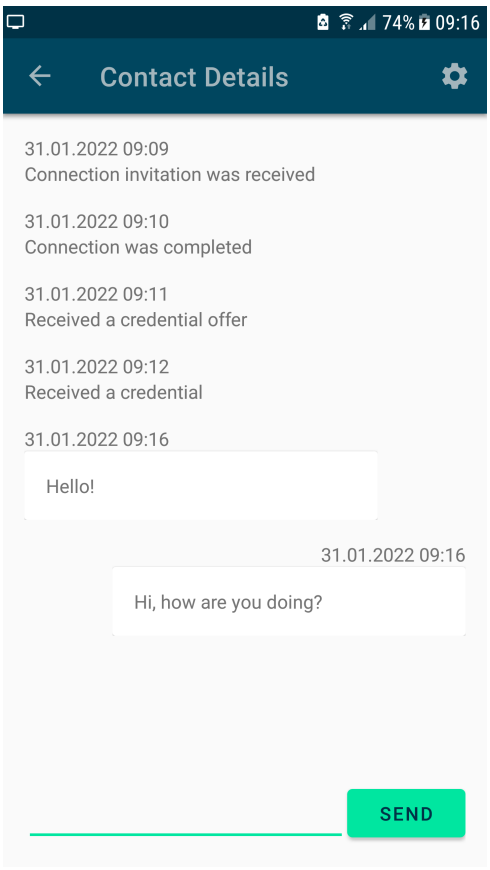

Figure 52: Mobile Wallet: Contact Detail Screen

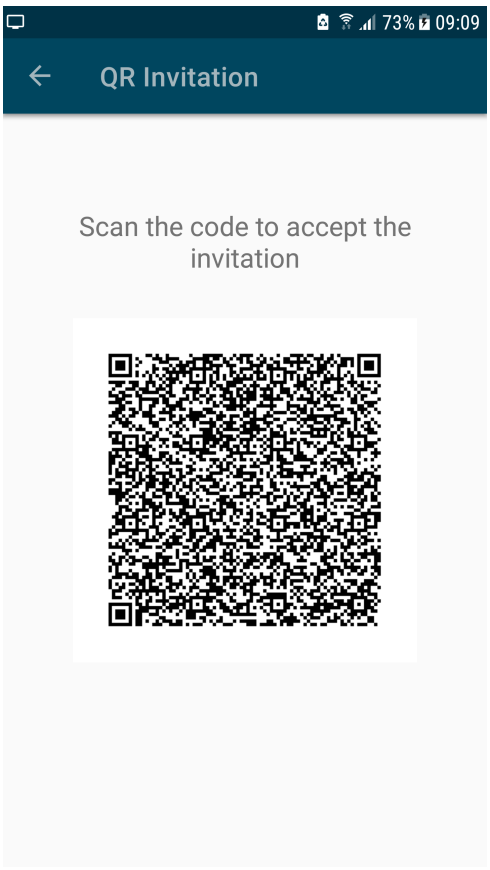

Figure 53: Mobile Wallet: Contact QR Invitation Screen

14.1.7 QR Scan

The QR scan mode enables the mobile wallet to scan other's QR code and if necessary react to them with the mobile wallet dialogs(see Chapter 14.1.8). The qr scan screen shows a preview from the camera feed and can be canceled by clicking the left arrow icon on the top left(see Figure 54).

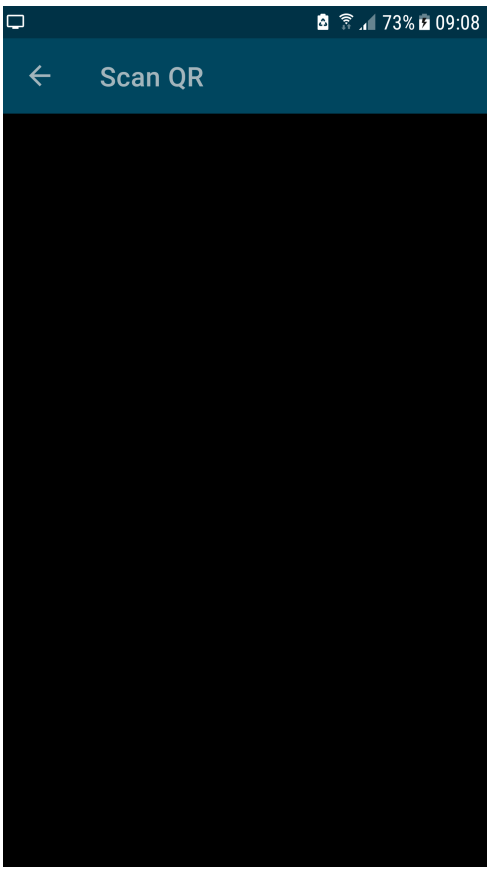

Figure 54: Mobile Wallet: QR Scan Screen

14.1.8 Dialogs

Dialogs are pop-up screens that are mainly used to asynchronously ask for decisions from the mobile wallet user. They are triggered by application events and overlap the current mobile wallet screen. The main ones are:

- Accept a connection request(see Figure 55)
- Accept an issued credential(see Figure 56)
- Choose which credential to present as proof(see Figure 57)
- Confirm deletion of entries(see Figure 58)

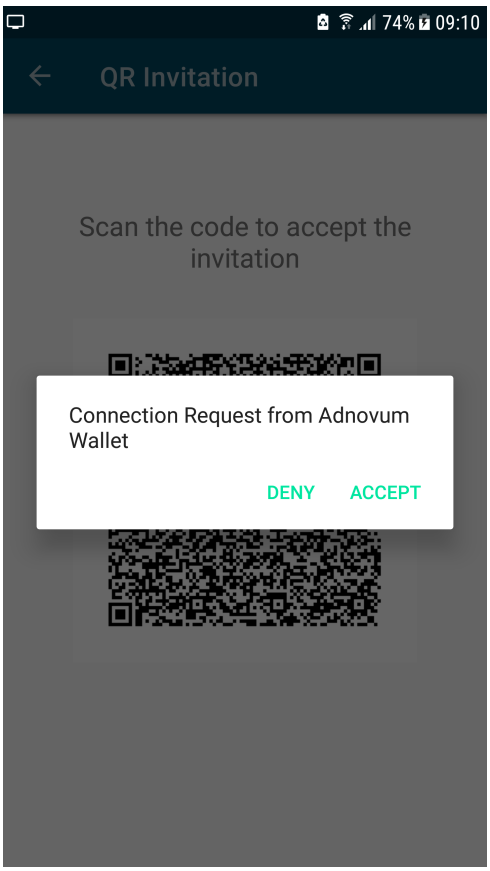

Figure 55: Mobile Wallet: Connection Request Dialog

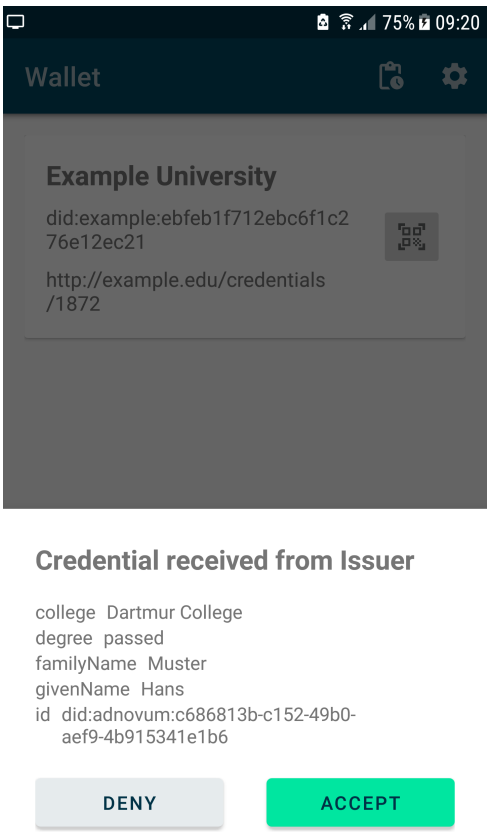

Figure 56: Mobile Wallet: Issue Credential Dialog

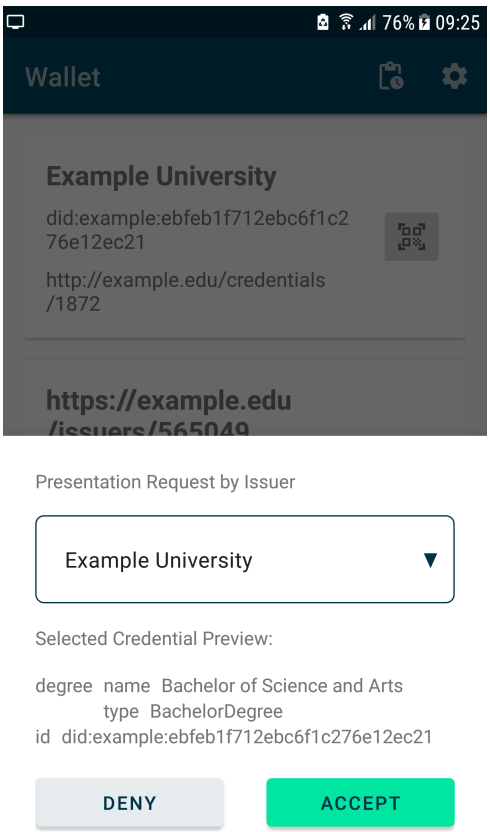

Figure 57: Mobile Wallet: Credential Presentation Dialog

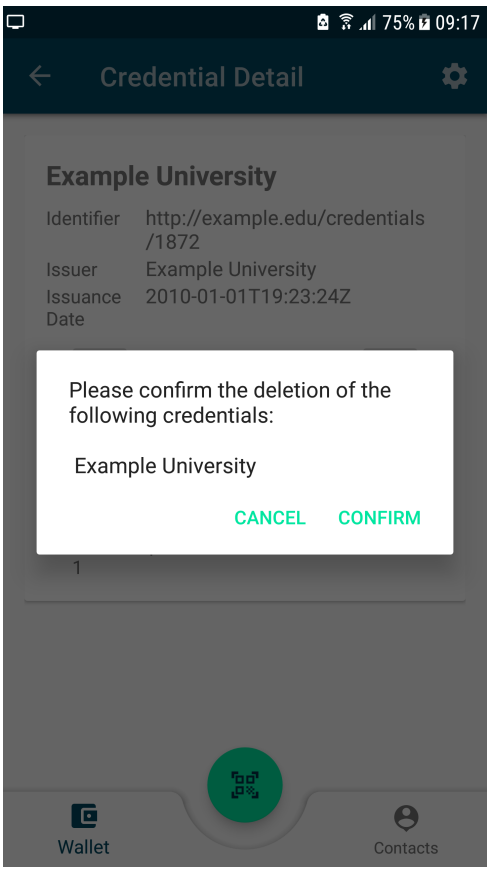

Figure 58: Mobile Wallet: Credential Deletion Dialog

14.1.9 Pending Tasks

The pending task screen has a list of pending choices for the user (see Figure 59). Most entries come when users are dismissing dialogs for events such as credential issuing. Each entry has the ability to reopen the dialogs and make a choice at a later point in time(see Figure 60).

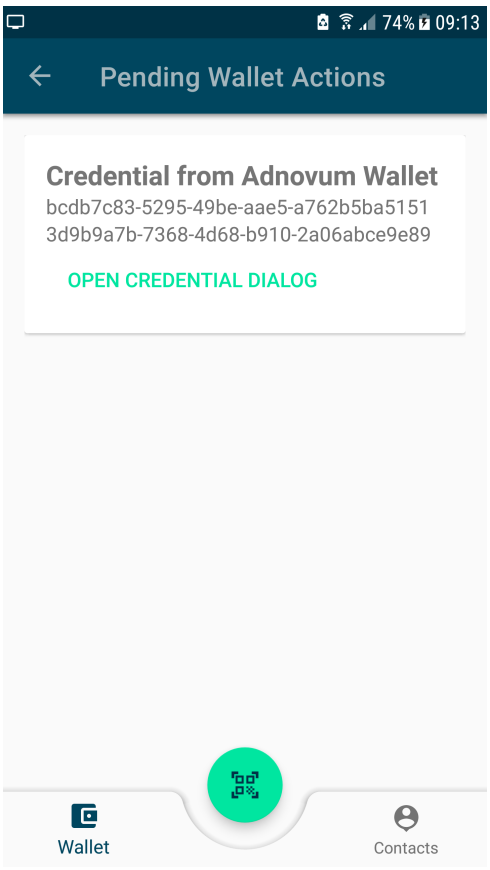

Figure 59: Mobile Wallet: Pending Tasks Screen

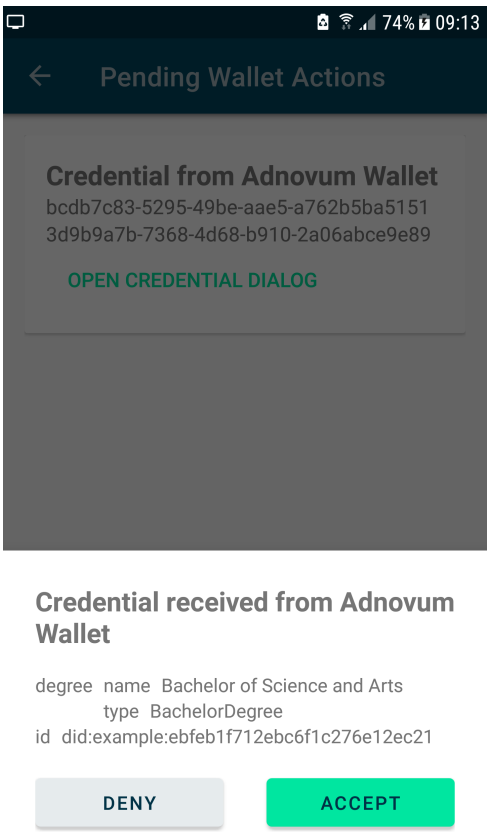

Figure 60: Mobile Wallet: Reopen Dialog from Pending Tasks

14.1.10 Settings

The setting screen contains the name of the mobile wallet, the commit version of AF-Go (see Figure 61). It also stores two menu entries that would not be available on production. The first opens the development panel that was used to interact with AF-Go when visual controls were not implemented yet(see Chapter 14.1.12). The second opens the credential validator which scans for credential proof QR code and shows validity and content to the user(see Chapter 14.1.11).

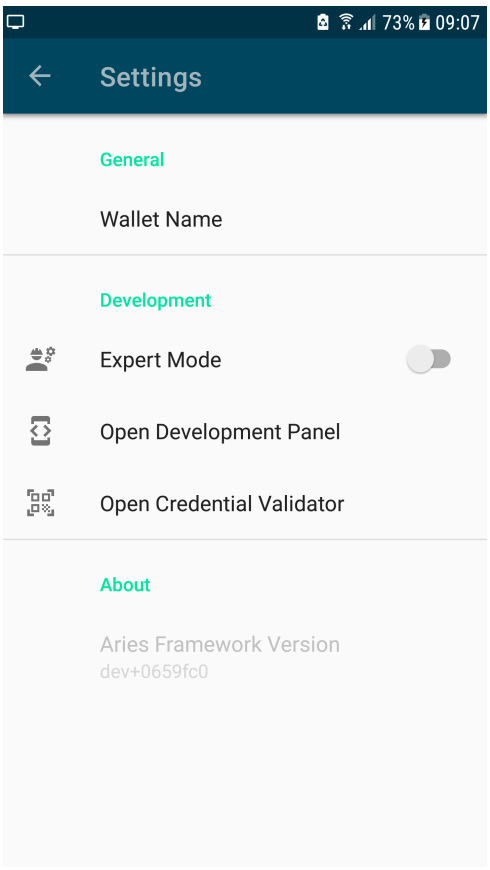

Figure 61: Mobile Wallet: Settings Screen

14.1.11 Credential Validation

The credential validation could also be a separated application. In this thesis it was combined into the mobile wallet for simplicity and time savings.

The validation starts in the QR scan mode and shows the current camera feed(similar as in Chapter 14.1.7). Once successfully scanned, the user is redirected to the credential validation screen. The credential's content is displayed and the validity of the credential and the presentation is presented below(see Figure 62). There are three validity states for the proofs. Either they could be invalid, inexistent (left on Figure 62) or valid (right on Figure 62).

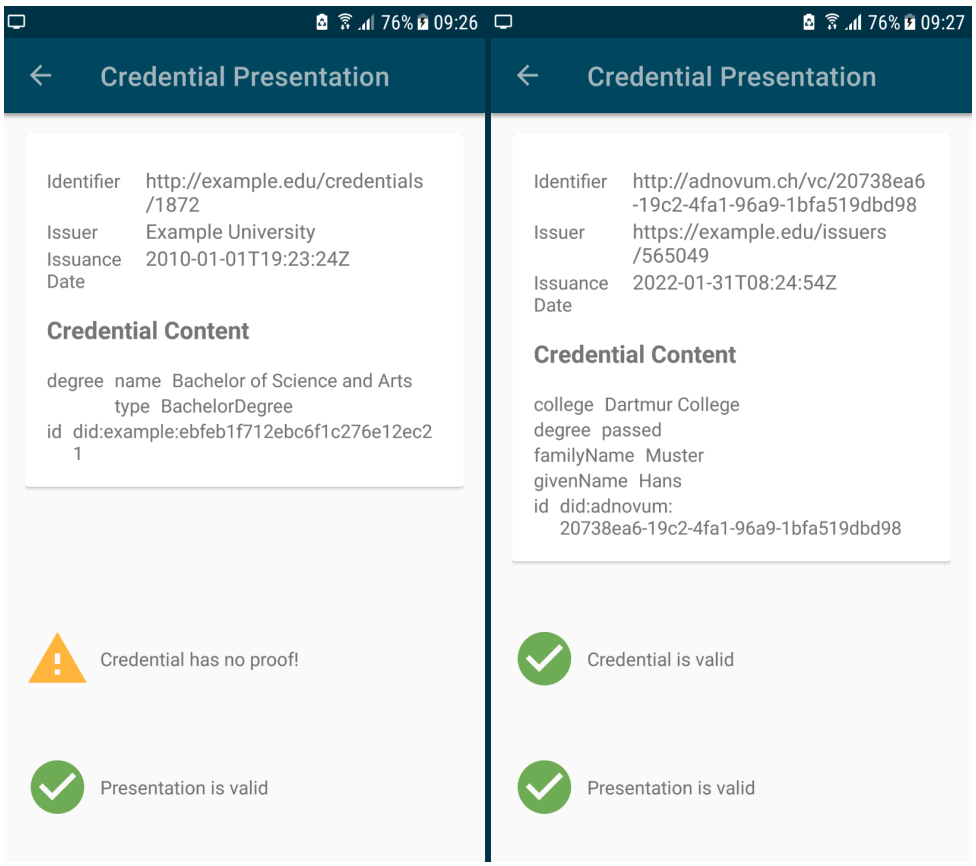

Figure 62: Mobile Wallet: Credential Validation Screens

14.1.12 Development

The development screen contains a set of visual input and buttons to control the SSI framework AF-Go underneath (see Figure 63). It does only contain the minimum as it was mainly used to control the framework before the real screens were implemented. As of now, it remains as a debugging tool.

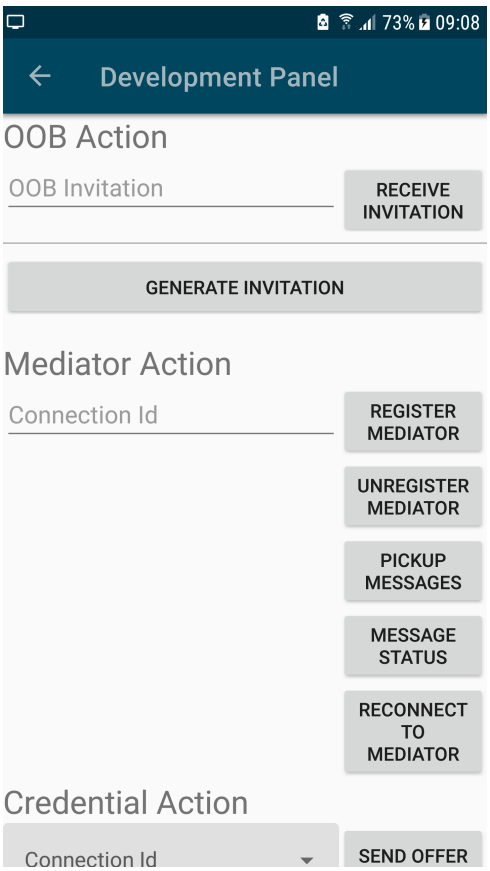

Figure 63: Mobile Wallet: Development Screen