POLITECNICO DI TORINO

Master's Degree in Computer Engineering
Cybersecurity Focus

Decentralized Identity Management: Building and Integrating a Self-Sovereign Identity Framework

Supervisor
Prof. Danilo Bazzanella
Company Supervisors
Dr. Alfredo Favenza
Dr. Silvio Meneguzzo

Candidate
Luca Rota

Accademic Year 2023/2024

Summary

In the era of technology transition, the traditional concept of identity has been redefined, giving rise to digital identity. This important shift in the identity management is setting the need for reconsidering how we treat the identities. The most feasible solution in this context is the Self-Sovereign Identity (SSI), which is a model designed to put identity under the control of individuals.

This master's thesis, conducted at the Links Foundation, delves into the growing world of SSI within decentralized systems. By using blockchain technology and a tool called MetaMask,a SSI standalone framework has been developed and later integrated into the Data Cellar project of Links Foundation, showcasing its real-world utility. Going through the aspects related to cryptography and cybersecurity, the research ends in a decentralized application (DApp), combining an easy-to-use interface (developed with React and JavaScript) with a robust foundation (built using NestJs), for a secure and user-centric authentication.

While the decentralized identities discussions keep on evolving, this thesis fits in the ongoing discourse, emphasizing the pivotal role of blockchain technology in creating self-sovereign identity solutions and at the same time supports the commitment of Links Foundation's to digital innovation.

Acknowledgements

Contents

List of Tables

List of Figures

Listings

Acronyms

SSI Self Sovereign Identity

DApp decentralized application

DLT Distributed Ledger Technology

MemPool Memory Pool

PoW Proof of Work

PoS Proof of Stake

SHA-2 Secure Hash Algorithm 2

UTXO Unspent Transaction Output

PKI Public Key Infrastructure

PIN Personal Identification Number

ARPANET Advanced Research Projects Agency Network

IP Internet Protocol

IDP Identity Provider

DID Decentralized Identifier

VC Verifiable Credential

IDM Identity Management

SSO Single Sign-On

PGP Pretty Good Privacy

W3C World Wide Web Consortium

DDO DID Document

URI Uniform Resource Identifier

VP Verifiable Presentation

ZKP Zero-Knowledge Proof

DHT Distributed Hash Table

IPFS Interplanetary File System

TEE Trusted Execution Environment

EdDSA Edwards-curve Digital Signature Algorithm

ECDSA Elliptic Curve Digital Signature Algorithm

DDoS Distributed Denial of Service

ABI Application Binary Interface

API Application Programming Interface

JWT JSON Web Token

SDK Software Development Kit

CLI Command Line Interface

IDMS Identity Management System

Nonce Number used once

GUI Graphical User Interface

EU European Union

HTTPS Hypertext Transfer Protocol Secure

SSL Secure Sockets Layer

Chapter 1

Introduction

In a world of constantly advancing technology, the concept of identity has experienced significant transformation. Managing identity becomes a critical issue in this time when our lives are becoming bound by digital platforms, services and, networks. Centralized identity systems, being popular, are however, encumbered by issues like the lack of user control and data breaches. As an answer to the above mentioned challenges, the idea of Self Sovereign Identity (SSI) has been gaining more attention. SSI adopts a decentralized and user-centric approach to identity management that enables individuals to assert and manage control over their identity data effectively. Through blockchain technology, SSI enables a secure and trust-based architecture of identity verification, where intermediaries are not required and identity is protected from theft. The present research centers on the terrain of digital identity, especially concentrating on the standards and implementations of SSI.

1.1 Objectives

This thesis delves into the ever-evolving ecosystem of Links Foundation to thoroughly explore the concept of SSI. Our ambitious objectives are to develop an autonomous SSI framework for managing decentralized identities by using MetaMask and the Ethereum blockchain. Furthermore, we strive to seamlessly incorporate this cutting-edge framework into the advanced Data Cellar project at Links Foundation.

Link Foundation is an innovative organization that facilitates digital transformation via applied research, innovation, and technology transfer projects [28].

Founded by the collaboration between the Compagnia di San Paolo and the Politecnico di Torino, the Links foundation is the central part of different technical and scientific disciplines, developing projects raging from Artificial Intelligence to Cybersecurity.

This exploration takes us into the intricacies of technology tracing the origins of identity from its non digital beginnings to its current form. By studying the foundations of SSI we gain insights, into the security and privacy measures in place as well as addressing the prevalent cybersecurity challenges within the SSI domain.

With a focus on practicality this research culminates in developing a SSI framework that incorporates groundbreaking elements like MetaMask and a customized Ethereum smart contract. This framework has the potential to revolutionize identity management, as evidenced by its integration into Data Cellar.

As we reach the end of this journey, we celebrate achieving an accomplishment, creating a decentralized application (DApp). This achievement not showcases an user friendly interface built with React and JS for frontend development and NestJs for backend development but also seamlessly integrates our SSI framework. This milestone not overcomes standing obstacles but also paves the way, for a future where individuals have greater control over their digital identities.

1.2 Outline

After a brief introduction and description of the goals of the thesis presented in Chapter [1], the remainder of the paper is structured as follows:

- **Chapter [2]**: This chapter, therefore, is all about the background of the SSI model used in this study, which is blockchain technology. It starts with the definition and roles of blockchain, passes to the analysis of different types of blockchains, and ends with a comparative review of two famous existing blockchains, namely Bitcoin and Ethereum.
- **Chapter [3]**: In this relatively concise chapter, an introduction to identity is provided, encompassing its evolutionary history from the pre-digital era to a comparison of the three primary paradigms utilized for digital identity management: centralized , federated and user-centric.
- **Chapter [4]**: This chapter focuses on the main theme of the thesis, namely the SSI model. It evaluates the state-of-the-art of this model of leading by starting from its historical context, then defining its main principles and presenting its advantages. Subsequently, the three main components of a SSI system, consist of DID, VC, and Verifiable Data Registry, will be discussed

followed by an architecture analysis with a main focus on the "Trust Triangle" relation between the three principal entity: holder, verifier, and issuer.

- **Chapter** [5]: According to the previous chapter's theme, this paragraph goes further to provide an in-depth study of the other strong cryptographic and cybersecurity aspects of the SSI model. Firstly, it highlights the working of VC proofs and then analyzes the attack vectors and their vulnerabilities to which this model is prone.
- **Chapter** [6]: This chapter ends with the theoretical part and starts the practical discussion applied during the thesis period. Overall, it explains the whole procedure of building an independent framework to handle the digital identities of individuals in a decentralized way under the SSI model. It demonstrates the instruments used for its implementation, the limitations encountered, and the final result.
- **Chapter** [7]: In this subsequent chapter, the integration of the previously mentioned framework into a practical project will be presented to show its practical significance. It provides a brief background of the project and then proceeds to a comprehensive analysis before and after integration.
- **Chapter** [8]: This final chapter summarize the conclusions drawn from the research.It begins with a succinct summary outlining the primary thesis topic and the executed practical endeavors, followed by an exposition of potential future work avenues necessitated by encountered limitations.

Chapter 2

Blockchain Technology

In the current landscape, blockchain technology has attracted increasing global interest due to its promising applications and potential transformative impacts on various sectors. Founded in 2008 by Satoshi Nakamoto as the mainstay of the Bitcoin system [19], blockchain has evolved from a simple ledger of financial transactions to a fundamental technology that revolutionizes the way information is recorded, shared and managed within decentralized networks.

2.1 What is the Blockchain

Blockchain is a core technology that drives many decentralized systems that offer transparency, trust and safety without having to have a central authority. It is basically a distributed ledger that operates through the peer-to-peer network [25], [21]. This section talks of the key components and architecture of blockchain, revealing its fundamental features and operating principles.

2.1.1 Core Elements of Blockchain

Blockchain comprises several key elements essential to its functionality:

- **Blocks**: A block is a basic unit comprising transactional data. Every block is securely connected to the previous one; this connection creates a chain of blocks. Transactions within a block are cryptographically secured; hence, immutability and integrity [25], [21].
- **Transactions**: Transactions are diverse interactions in the blockchain network. These interactions are not limited to financial transfers only; any

piece of valuable information can be considered as a transaction and diffused within the network [19], [35].

- **Decentralization**: Unlike centralized systems that depend on a single controlling authority, the blockchain is decentralized. It includes a web of interdependent nodes, each replicating the distributed registry. Resilience, transparency and no single points of failure are guaranteed by decentralization [25], [19].
- **Consensus Mechanism**: For verification and agreement of the state of a ledger, blockchain uses consensus mechanisms. The most common mechanism, PoW, requires miners to solve complicated cryptographic puzzles to add new blocks on the chain. Consensus mechanisms ensure consensus among network participants to prevent attacks by malicious actors [25], [19].

2.1.2 Blockchain Architecture

The structure of the system is carefully crafted to uphold its principles of decentralization, immutability and transparency;

- **DLT**: At the core of blockchain lies DLT, where all participants in the network have access to a ledger of transactions. This shared ledger eliminates redundancy, ensures that everyone has a trusted source of information across the network [21], [19].
- **Immutable Records**: One key feature of blockchain is its record immutability. Once a transaction is recorded on the ledger it cannot be tampered with. Any attempt to change a transaction requires adding one that preserves the integrity of data [21].
- **Smart Contracts**: Smart contracts are self-executing contracts with predefined rules embedded within the blockchain. These contracts enforce agreements between parties speeding up transaction processing and reducing reliance on intermediaries [21], [35].
- **Security Measures**: Cryptography plays a role in ensuring security by protecting transactions from tampering and fraud. Key pairs authentication enable secure digital identity management. In addition, cryptographic hashing guarantees data integrity and confidentiality [25], [35].
- **Peer-to-Peer Network**: The blockchain functions using a network architecture in which participants can directly communicate and interact. This decentralized structure promotes trust and resilience since transactions are verified and validated through consensus among distributed nodes [19], [35].

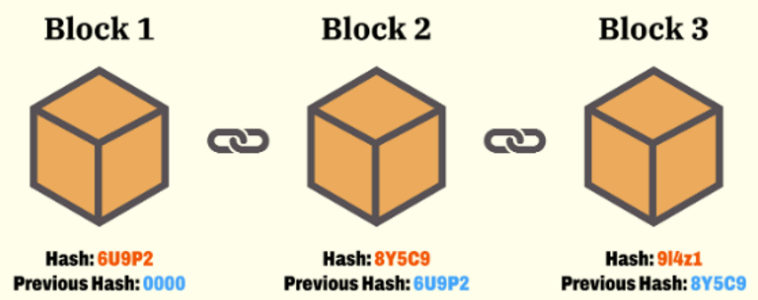

Figure 2.1. Blockchain structure

2.2 How the Blockchain works

The function of a blockchain system encompasses a complex chain of procedures that safeguard the integrity, transparency, and security of transactions. In this section, we delve into the fundamental mechanisms of blockchain.

2.2.1 Transaction process

Recording transactions

1. **Starting a Transaction:** In a network, users initiate transactions through their wallets. Each wallet has a pair of keys. A key, for starting transactions and a private key for authentication [25].
2. **Creating Blocks:** As transactions occur, they are grouped together into blocks of data. These blocks act as containers for recording details like asset movement participants involved and timestamps [21].
3. **Hashing:** Every block contains information and refers to the hash of the previous block. Secure hash functions like SHA 256 play a role in ensuring the integrity and immutability of transactions. Hashing allows each block to have a fingerprint, making identification and verification simple [21] ,[35].

Consensus Mechanism

4. **Verification Process:** When a transaction is initiated the information is sent to a network of distributed peer to peer nodes. These nodes work together to confirm the validity of transactions using consensus mechanisms [35], [18].

5. **Formation of New Blocks:** Valid transactions are added to a MemPool, where they wait to be included in a block. Miners, who are responsible for creating blocks solve cryptographic puzzles in order to mine blocks. This process, known as PoW requires resources and time [25], [18].

6. **Consensus Algorithm:** In order to add a block to the blockchain nodes must agree on its validity through consensus algorithms. The miner who successfully creates a block is rewarded. Consensus algorithms ensure that all nodes are synchronized and in agreement, about the state of the blockchain [18].

7. **Blockchain Integrity:** The interconnected nature of blocks ensures the immutability and integrity of the blockchain. Each block references the hash value of its predecessor, making it practically impossible to tamper with transactions without altering blocks [25] ,[35].

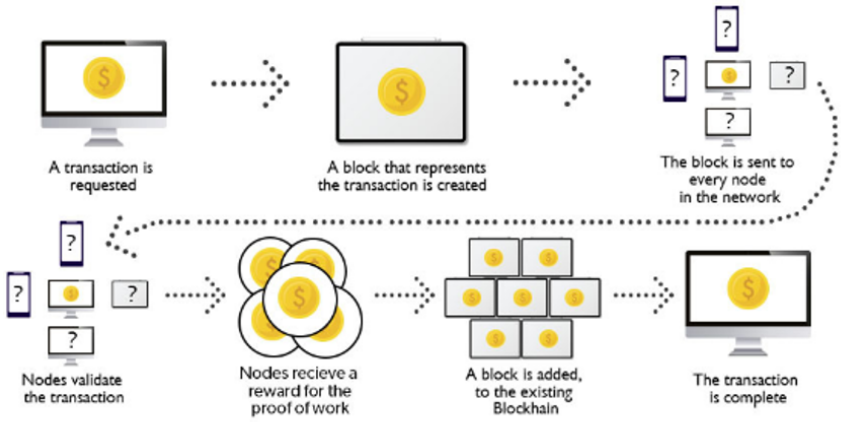

Figure 2.2. Blockchain transaction process

2.2.2 Blockchain Benefits

Due to the execution process described above and its architecture, blockchain offers many benefits in different areas:

- **Greater Trust:** Blockchain helps establish trust across the network by utilizing the decentralized ledger, ensuring that data is consistent and timely plus, it does not require intermediaries [25].

- **Enhanced Security:** Blockchain's security features are designed to thwart tampering as well as cybercrime. The blockchain structures transactions as read-only data which is difficult to change and ensures that the data is protected even when it is in transit. Moreover, cryptography is used as a verification mechanism and the cryptographic infrastructure is used to store and transmit secrets [21].
- **Time Savings:** The utilization of blockchain technology greatly decreases the processing time for transactions since they are completed within a quarter of an hour due to the removal of the centralized authority that confirms [21].
- **Cost Savings:** The blockchain adjusts the transaction procedure and lowers operational costs by removing intermediaries, as tasks are automated and there is lower duplication of activities through the usage of shared ledger [25], [35].
- **Efficiency Improvements:** The distributed architecture of blockchain increases system resilience, decreases processing costs, and removes the need for centralized network control by distributing network operations thinly, facilitating faster settlements, and solving identity management issues [35].
- **Transparency Enhancement:** A blockchain keeps a permanent record of transactions through which it delivers transparency, accountability and trust to network users via its chronological and transparent nature [35].

2.3 Blockchain Classification

2.3.1 Types of Blockchains

Blockchain technology comes in various forms, each with unique characteristics that adapt to specific needs and application contexts Beyond the well-known public and private blockchains, it is essential to recognize two other significant variants: hybrid chains and consortium chains.

- **Public Blockchain:** A public blockchain is open to everybody who wants to interact and contribute to the consensus protocol, e.g., Bitcoin. This model provides a high degree of transparency though, it is susceptible to 51% attacks. Its open nature fosters a distrustful environment, as no one individual or entity is solely relied on for transaction validation [25], [21].
- **Private Blockchain:** In contrast to public blockchains, private or permissioned (access is restricted to authorized entities) blockchains allow only a limited set of entities to access the network. This method is popular among situations that demand more features such as security as well as control where it is applied in applications like financial transactions that also handle sensitive data. Access to the network is controlled by a single entity or by particular criteria [25],[35].
- **Consortium Blockchain:** A consortium blockchain is governed by more than one organization or entity that works together to ensure the distributed ledger. On the other hand, blockchains that are public or private, consortium blockchains are the only ones that give power to a chosen group of nodes to validate and record transactions. This model is suitable where several subjects need to work with data and processes together but they do not want or cannot fully trust a unique central entity [21], [19].

Property	**Public Blockchain**	**Consortium Blockchain**	**Private Blockchain**
Consensus determination	All miners	Selected set of nodes	One organization
Read permission	Public	Could be public or restricted	Could be public or restricted
Efficiency	Low	High	High
Immutability	Nearly impossible to transfer	Could be tampered	Could be tampered
Centralized	No	Partial	Yes
Consensus process	Permissionless	Permissioned	Permissioned

Table 2.1. Types of Blockchain

- **Hybrid Blockchain:** In addition to the previous ones mentioned above, a new type of blockchain, hybrid blockchains, has also emerged. They are described by the property of switching between different modes having different types of operating systems, for example, public and private blockchains that suit specific needs. This approach provides a greater degree of flexibility and customization compared with conventional blockchain configurations; thereby, the actors have the option to decide the degree of decentralization and control that is suitable for them [25].

2.3.2 Types of Consensus Mechanisms

Blockchains also differ based on the type of consensus mechanism used. In the previous section we discussed an example of how blockchain uses a consensus mechanism called PoW [19], [42]. However there are several types of consensus mechanisms that can be used in a blockchain network.

In general, the two main types of consensus mechanisms are: PoW and PoS. With PoW miners solve puzzles to add new blocks, which requires significant computational resources and time [19], [42]. This is elaborated further in Section 2.4.1. On the hand PoS assigns the right to create blocks based on participants cryptocurrency holdings, providing benefits such, as energy efficiency and scalability advantages. Further information, about PoS can be found in Section 2.4.2. There are also other consensus mechanisms beyond PoW and PoS; refer to the table for further details.

Property	**PoW**	**PoS**	**PBFT**	**DPOS**	**Ripple**
Node identity management	Open	Open	Permissioned	Open	Open
Energy saving	No	Partial	Yes	Partial	Yes
Tolerated power of adversary	<25% computing power	<51% stake	<33.3% faulty replicas	<51% validators	<20% faulty nodes in UNL
Example	Bitcoin	Ethereum	Hyperledger Fabric	Bitshares	Ripple

Table 2.2. Types of Consensus Mechanism

2.4 Bitcoin versus Ethereum

Bitcoin and Ethereum stand as two big giants in the blockchain technology area, and both of them possess their own characteristics and they make their own contributions to the ever-changing world of decentralized systems. This section takes a comparative approach between Bitcoin and Ethereum; analyzing their basic architectures, transaction mechanisms, consensus methods, cryptographic techniques, and the advantages and lacks they can offer.

2.4.1 Bitcoin

Overview

Bitcoin, launched in 2008 by Satoshi Nakamoto, is a decentralized digital currency that is also accompanied by the great invention, the blockchain. It functions as a peer-to-peer decentralized electronic payment system; it introduces a new way of executing business transactions and ensuring data security [33]. Bitcoin is an open-source and permissionless blockchain, allowing for involvement of anyone with the required hardware.

Transaction mechanism

When a user sends bitcoins, sender hashed addresses, recipient hashed addresses, transaction amount and fee are recorded. Digital signatures prove property rights and the transaction is floated to the network, being submitted to the mempool for confirmation [3]. The mining being the process of validating transactions; therefore, miners pick transactions from the MemPool in order to create new blocks by solving complex mathematical puzzles called PoW [33]. The winner of the puzzle then broadcasts their newly found block to the network. After a verification done by other network nodes, the block is added to the blockchain, thus finalizing the transaction.

Proof of Work

Bitcoin uses the PoW consensus algorithm to maintain agreement among network participants about the validity of transactions. The PoW was first specified in the Hashcash paper [33]. It requires miners to search for solutions of cryptographic puzzles consuming computational power. Through the cost of computation, miners establish their commitment to the security of the network as changing the

blockchain will require enormous computational resources, hence, making fraudulent endeavors expensive.

Bitcoin Cryptography

Bitcoin uses the SHA-2 for cryptographic hashing, which ensures irreversible and collision resistive attributes. This guarantees the data integrity of the blockchain, which makes it impractical to tamper with transactions that are already in the database. Moreover, the Merkle Trees that Bitcoin natively uses for block data encoding take security to the next level [33].

Unspent Transaction Output

Bitcoin transactions are registered in the blockchain as UTXOs, which meaning constant fractions of bitcoins that are linked to a certain owner. This UTXO is pervasively distributed over the blockchain and represents ownership [3]. It is subsequently used in other transactions too. Bitcoin balances are not stored in user accounts but rather tracked through the UTXOs associated with their addresses.

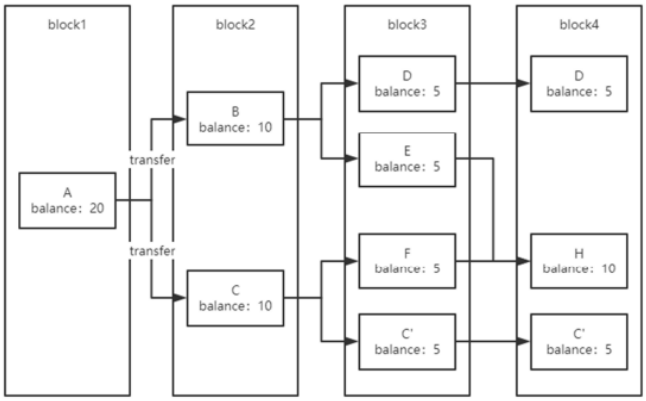

Figure 2.3. UTXO transaction process

Advantages and Limitations

Bitcoin is decentralized and provides encryption security that can be used to resist censorship, to ensure accessibility particularly on a global scale and to preserve integrity of the data. Nevertheless, Bitcoin has limitations such as the scalability problem, transaction throughput rate without the corresponding increase in the number of transactions and undulatory acceptance and valuation. Furthermore, there are additional security concerns, such as the "51% attack" and human error,

which continue to plague the Bitcoin ecosystem [33].

2.4.2 Ethereum

Overview

Ethereum, designed by Vitalik Buterin, is the introduction of Smart Contracts to the crypto-sphere, which is a novelty in the blockchain space, and, obviously, it has been a highly valuable addition. The September 2022 saw the newly born Ethereum 2.0 replace Ethereum 1.0 by incorporating the Beacon Chain and the shift from PoW to PoS consensus mechanism [16]. This change shall constitute one of the really important milestones reached by Ethereum, the blockchain providing scalability, sustainability, as well as security.

Beacon Chain Impact

The introduction of the Beacon Chain revolutionized Ethereum's operating model [16]. It replaced the original execution layer and introduced a new consensus layer, the PoS which was a major improvement in terms of power consumption and the amount of energy consumed was reduced from 99.95%. Validators, staking their ETH, assumed the responsibility of block production and transaction validation, moving away from energy-intensive mining. This transition helped create a sustainable network architecture as well as secure the information connections.

Transaction Mechanism

In Ethereum 2.0, validators play a pivotal role by depositing 32 ETH and operating three essential software components: a execution client, a client for consensus, and a client for validation as well. Here we have the validators appointed randomly for being picked on 12 second timeslots which then make up epochs composed of 32 slots. They formulate and verify blocks and this way they add to the Ethereum System. This process regulates all the transactions on the blockchain. Transactions include processing stages of creating, verifying, and block proposal irrespective of the fact that the transactions are created, verified, combined into execution payloads, and broadcasted across the network so as to make self-enforcement and fault-tolerance in proper working order. Finality being one of the main features of transaction irreversibility, is sealed by using checkpoints that initiate even the smallest epochs. As requisite effort to checkpoints pairs, validators vote in a supermajority to revert them, consequently, block reversal is discouraged and the entire Ethereum network is protected from brainwashing attacks [15].

Proof of Stake

The Ethereum 2.0 version puts the PoS consensus mechanism at the core of the protocol with validators given the chance to make the most of their investment through staking [15]. Validator's deposit their own ETH as bond to release validator software which helps in the job of validating the transactions and generating new blocks proposals. In contrast to proof-of-work, the PoS is doing well in maintaining both the security and the decentralization of validators that attempt to undermine the network due to its feature of penalization.

Figure 2.4. Ethereum Proof of Stake's features.

Ethereum Cryptography

Ethereum 2.0 still relies heavily on the standards of crypto- security for safety, keeping the Keccak-256 algorithm implemented by Ethereum 1.0. These algorithmic primitives give Ethereum strong security by making sure the data integrity, confidentiality, and authenticity [33].

Advantages and Limitations

The new Ethereum 2.0 version has several new advantages. The transition from PoW to PoS results in a substantial reduction in energy and makes Ethereum more sustainable. In addition, the PoS model makes the network more secure, decentralized and scalable, so that a thriving ecosystem of decentralized applications can grow within it [15]. However, even this version, still has some drawbacks. Processing time can still be a bottleneck, as Ethereum cannot process transactions at the same speed as traditional processes, such as Visa [33]. Besides, market fluctuations and rumors become the main problems for Ethereum's stability and reputation, which requires continuous technological development and management.

Chapter 3

Evolution of Identity

The notion of identity verification has changed drastically over the course of history, and with the advent of digitalization, a new era where different problems are met. The chapter aims at dissecting the phase of identity transition from non-digital to digital form, noting the historical background and major milestones that have shaped present-day identity verification. The development of digital identity models will be reviewed, including the centralized, federated and decentralized approaches that will be discussed, with a focus on their impacts on security and privacy in the digital age. Through this overview we attempt to create a full picture of how identity changes in the face of an increasingly interconnected world.

3.1 Pre-Digital Era

3.1.1 Origin of Identity

The identity concept has ancient origins and it has been expressed in a number of forms in ancient times, including jewelry, tattoos and cultural symbols. These material signs not only expressed own individuality but also signified social status, family ties, and community membership [4]. For example, Babylonians, Romans, as well as the Chinese ancestors used tattoos and fingerprints as primitive identification methods, and some cultures treated thumbprints as legal signatures.

3.1.2 Early Documentation

The requirement to identify people's origin developed with the growth of civilizations and the need for safe travel and commercial transactions. The earliest recorded types of documentation were in Persia in approximately 450 BCE and

these documents, which bordered on rudimentary passports, carried essential information about the traveler [4]. These documents proved to be the most important for securing safe travels and identifying an individual during the journey.

The idea of the official passport took form as societies reached a certain level of development, especially during Henry V's reign in England. In the beginning, passports started as "safe-conducts" signed personally by the king, while in the 20th century, passports turned into the standardized documents. Later, the use of photography in passports took identity verification a step further, providing a visual reference for identifying individuals accurately [4].

3.2 Emergence of Digital Identity

3.2.1 PINs and Passwords

PINs and passwords became critical elements of digital identity validation. Starting in the 1960s, PINs were fundamental to secure data and transactions, growing from simple four-digit codes to more complex security measures such as "chip and pin" [20]. Simultaneously, passwords that employ characters and numbers have been used for decades to provide user authentication across multiple digital services. Together, the PINs and passwords have thus been playing a pivotal role in protecting the digital identities of individuals.

3.2.2 Introduction of ARPANET and IP

With ARPANET coming to be in 1969, a new digital era of connectivity began. Nevertheless, it was required to have parallel robust identity protection mechanisms, as a result of this global network. By adopting the username and password, ARPANET resorted this drawback, allowing secured access to the networked resources. On the other hand, IP has taken over as the standard for device addressing, which facilitates end-to-end data transferability across various networks. IP addresses, which are identifiers of networked devices, played a key role in packet routing and maintaining secure communication over the internet [20].

3.2.3 The Public Key Cryptography

Public Key Cryptography has revolutionized in digital identity protection, providing trusted communication over public networks. Diffie and Hellman introduced the idea of public-key cryptography in 1976 [4]. Using this system pairs of interconnected keys, one key for encryption and the other one for decryption, are

generated. PKI successfully added a layer to identity verification by matching public keys to specific entities, enabling secure transactions and data exchange.

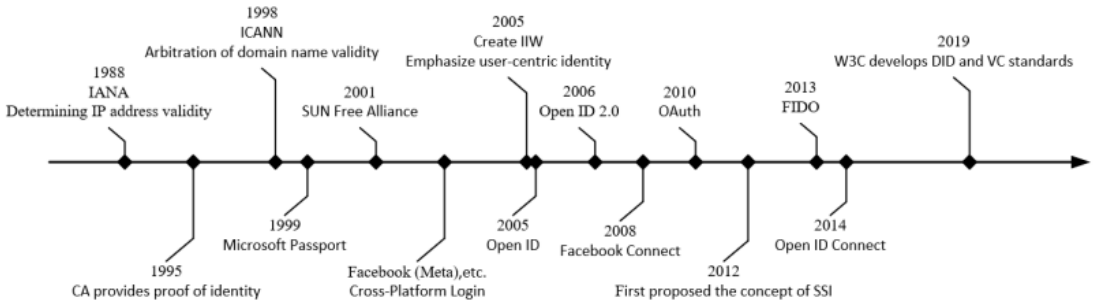

Figure 3.1. The timeline of digital identity development.

3.3 Shifting Identity Paradigms

This section focuses on the evolution of IDM models, which have been categorized into three main paradigms. These paradigms describe the evolution of digital identity management which show trends for more user-friendly and secure approach.

Model Name	Credentials Ownership of User	Optional Disclosure	Information Silo	Support Pseudonyms	Centralized Storage
Centralized Identity	X	X	X	X	V
Federated Identity	X	X	V	X	V
Self-Sovereign Identity	V	V	X	V	X

Table 3.1. Comparison of the characteristics of three models.

Centralized IDM model (IDM 1.0), includes organizations issuing credentials to users, using shared secrets, such as usernames and passwords, for authentication. Federated IDM model (IDM 2.0) brings in third-party IDP for SSO and so still

centralizes user's personal identifiable information. Finally, Self-Sovereign IDM model (IDM 3.0) allows users to have absolute mastery over their digital identities via Digital Wallets using standards such as VC and DID [32].

3.3.1 Pitfalls of Centralized Identity

Centralized IDM models were predominant in the initial stage of the digital identity development process, and these models were managed by organizations and institutions. Under this model, people literally gave their personal data to these centralized authoritative bodies, that in return gave them status or official recognition [23]. Having all personal data in one place was a big plus; on the other hand, it also had some disadvantages.

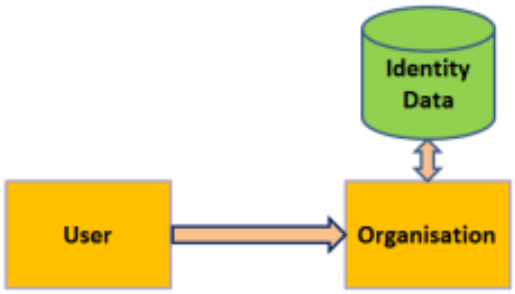

Figure 3.2. Centralized Identity Management Model (IDM 1.0)

One of the primary issues with Centralized IDM models is that they are naturally prone to security breaches. The fact that all user data were stored in a single location meant that in case of a breach in the system, victims of this breach would be subject to widespread data leaks and identity theft [4]. Furthermore, users had to deal with the bother of having too many account names and passwords for the different platforms which increased the risks of password-related security issues. Even though it was convenient at first, centralized identity model has proved to be inadequate in helping to solve emerging security threats and protecting user privacy.

3.3.2 Emergence of Federated Identity

To deal with the issues caused by the Centralized IDM models, the idea of the Federated IDM models appeared, which is much more flexible and user-centered [23]. These identity management solutions paved the way for IDP, a system which enables the creation, handling, and implementation of online identities. of federated identities, the users could use a unique identity registered with an IDP to

access different network applications within their domain, and the authentication process would be simplified to just clicking once.

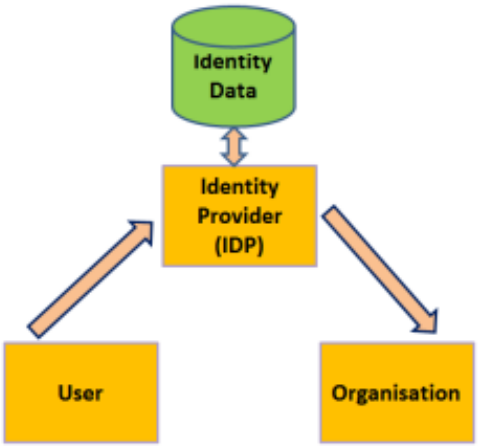

Figure 3.3. Federated Identity Management Model (IDM 2.0)

With the adoption of federated identity solution, user convenience and inter-platform operability have been enhanced, but new security challenges also arise. An expansion of the IDP gave rise to a more vulnerable attack surface for Federated IDM models that made them more prone to data breaches and cyber-attacks [23]. Furthermore, the usage of several IDP suggested the possibility of privacy and user consent problems, as the user's identity information was spread across multiple service providers. Federated IDM models persisted in gaining popularity even though they encountered many challenges owing to their flexibility and ease of use.

3.3.3 Evolution towards User-Centric Identity

In direct response to the rapidly developing issues surrounding traditional Centralized and Federated IDMs, user-centric identity appeared as a concept, which aims to give users back control over their digital identities. Specifically, the so-called Self-Sovereign IDM models have emphasized the importance of user control and consent in identity management, allowing them to store authenticators and certificates issued by a range of service providers in their personal devices [2].

The introduction of blockchain technology was a pivotal transformations in user-centric identity and it provides a decentralized and immutable ledger for identity verification. The SSI model was based on blockchain that enabled users to retain authority over their data, discarding intermediaries and centralized authorities

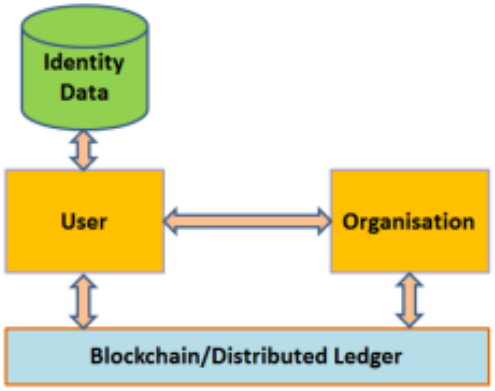

Figure 3.4. Self-Sovereign Identity Management Model (IDM 3.0)

[2]. In the SSI model, users can securely and privately own and control data and identity information while sharing them in a way, that protects their data sovereignty. Eventually, the deployment of SSI will need to be fully developed and adopted, however, it constitutes a major milestone on the way to a more secure and user-centric model for identity management.

Chapter 4

State of the Art of SSI

As mentioned in earlier chapters, the principle of SSI has become popular in contemporary times, particularly in the context of cybersecurity and identity management. SSI follows the decentralized method of managing digital identities, resulting in increased personal control and independence for individuals.

In this chapter, we discuss current status of the SSI model in the state-of-the-art. First, we give a historical background and then follow with an examination of evolution of SSI. Next we focus on its architecture with an in-depth look at its core components and operational steps. Through this sector, we look to furnish a full picture of the identity management framework and the SSI model in particular.

4.1 History of Self-Sovereign Identity

4.1.1 Origins of SSI

The roots of SSI can be traced back to the early 90s with the introduction of encryption methods like PGP by Phil Zimmerman [4]. PGP introduced the public key type of encryption, which formed the basis of later models of SSI. PGP allowed the users to directly exchange cryptographic keys, and, thus, enabled the creation of a network of trust without any need for centralized intermediaries.

4.1.2 The Seven Laws of Identity

The "Seven Laws of Identity" of Kim Cameron was pivotal in the development of SSI approach in 2005 [17]. These laws described the main building blocks of the user-oriented and conducive identity framework. The Seven Laws of Identity

articulate key principles guiding SSI, including:

1. **User control and consent**: Technical identity systems must only reveal information identifying a user with the user's consent.
2. **Minimum disclosure for a constrained use**: The solution which discloses the least amount of identifying information and best limits its use is the most stable long-term solution.
3. **Justifiable Parties**: Digital identity systems must be designed so the disclosure of identifying information is limited to parties having a necessary and justifiable place in a given identity relationship.
4. **Directed Identity**: A universal identity system must support both "omnidirectional" identifiers for use by public entities and "unidirectional" identifiers for use by private entities, thus facilitating discovery while preventing unnecessary release of correlation handles.
5. **Pluralism of Operators and Technologies**: A universal identity system must channel and enable the inter-working of multiple identity technologies run by multiple identity providers.
6. **Human Integration**: 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. **Consistent Experience Across Contexts**: The unifying identity metasystem must guarantee its users a simple, consistent experience while enabling separation of contexts through multiple operators and technologies.

4.1.3 Modern Development

Christopher Allen had also made the term SSI more popular than ever with his 2016 article "The Road to Self-Sovereign Identity" [4]. Allen's work was built on Cameron's principles that highlighted the crucial aspects of user control, longevity, portability and limited disclosure in SSI frameworks. These principles constitute the foundation of a system that is about individuals having the ultimate power over their personal information.

For the last few years, an increase in the use of blockchain technology has led to a rapid development of SSI. With blockchain being a decentralized and immutable system, it forms the backbone of SSI systems that secures digital identities due to their immutable and transparent nature. DID and VC are among others some of

the components of modern SSI architectures, which enables peer-to-peer transactions and having no need of central authorities.

4.2 Advantages and Principles

4.2.1 Advantages of Decentralized Identity

Advantages for Organizations

- **Efficient Verification**: Organizations thus can verify information faster without involving the manual verification procedures, improving operational efficiency [12].
- **Prevention of Certificate Fraud**: Decentralized identity systems prevent abuse of fake certificates minimizing the chances of credentials being forged .
- **Enhanced Data Security**: Public-key cryptography is utilized by organizations to encrypt data safely, which therefore helps reduce the risk of data breaches.
- **Reduced Cybersecurity Risks**: Minimizing user data information makes organizations less sensitive to cybersecurity attacks. The overall cybersecurity posture improves.

Advantages for Individuals

- **Data Ownership and Control**: People retain both ownership and control of their data, and they can independently manage their digital identities [12].
- **Self-Verification**: Individuals can establish their statements without the help of third parties, thus confidence and autonomy are promoted.
- **Privacy Protection**: Decentralized identity management provides better privacy because it can shield from indiscriminate tracking and allows for selective disclosing of data.
- **Immutable Identity**: Identities will be stored and kept in the decentralized digital wallets in which they cannot be arbitrarily deleted and provide every person with a reliable digital persona.

Advantages for Developers

- **Enhanced User Experience**: The developers can build applications for users with short and easy user identification processes, resulting in increased ease of use.
- **Privacy-Preserving Data Requests**: Developers can request the data directly from users while respecting their privacy and increasing trust-users and transparency.
- **Streamlined Transactions**: Developers can simplify transactions by reaching the relevant information securely via decentralized identity wallets without any time-consuming data collection activities [12].

4.2.2 Principles of SSI

Foundational Properties

- **Existence**: People can digital assets for their characteristics which exist in the digital domain. This is what the SSI achieves [1].
- **Autonomy**: SSI provides full independence for self-governance in identity management to issue, edit, and revoke digital identities independently.
- **Ownership**: Users as the final decision-makers own their identities, including self-asserted and third-party-attributed claims.
- **Access**: Users don't have to worry about their identity; they can freely control when it is needed.
- **Single Source**: Since individuals are the single point of reference for their identities, they also prevent unauthorized information exchange without their consent.

Security Properties

- **Protection**: SSI provides strong protection via cryptographic means, thus ensuring trustworthiness, privacy and integrity of stored information.
- **Availability**: Readily accessible identities must be cross-platform compatible, while being resistant and recoverable [1].
- **Persistence**: Identities should be preserved as long as needed, protected through secure identity storage and transmission.

Controllability Properties

- **Choosability**: Potential users could decide what data relating to identity they ought to disclose, granting access to information only in line with their preferences.
- **Disclosure**: Users have the luxury of sharing identifiable data in a selective manner to third party as long as it is done in a structured way that allows for fine-grained control [1].
- **Consent**: Identity information is delegated only with user consent, ensuring privacy protection and personal autonomy.

Flexibility Properties

- **Portability**: The identity projects need the portability across platforms in order to ensure its longevity and inter-operability.
- **Interoperability**: Maximum interoperability should be achieved by SSI systems, which in that case would allow for effortless communication with existing identity systems [1].
- **Minimization**: User objectives are meant to be covered, which minimizes data disclosure, thus ensuring data protection and efficiency of the data processing.

Sustainability Properties

- **Transparency**: SSI should be transparent, open-source, and accessible, which will produce trust, and participation of the community
- **Standardization**: The Identity should be compliant with open standards that enables it to have better portability and interoperability.
- **Cost**: Identity solutions should be cost-effective or free to make them affordable and that can lead to their wide adoption and inclusivity [1].

4.3 Key Components of SSI

This section elaborates on the three main pillars of SSI model, each of which is of critical relevance to the revolution of digital identity management. We shall properly discuss DIDs, VCs, and Verifiable Data Registries, highlighting their importance and functionality in the identity ecosystem which is decentralized.

4.3.1 Decentralized Identifiers

DIDs represents a cutting edge feature of SSI. Unlike the conventional identifiers such as emails and usernames that are centralized, DIDs provides a secure and decentralized way of verifying digital identities. DIDsis an individual identifier, which can be used separately for people, organizations, and data models, without the need of any centralized authority. DIDs are owned by users and are held in identity wallets, enabling people to retain control of their original data verified by certified issuers. [5] They are as well a solution to some of the problems linked to centralized identifiers such as identity theft and data leakage. Significantly, DIDs do not contain personally identifiable information, that is why they are secure [26].

Syntactic Components of DIDs

DIDs are composed of distinct syntactic components, delineating their structure and facilitating interoperability across decentralized systems [8] :

- **Schema**: The schema serves as a guideline to provide a uniform structure and format of the DIDs thereby ensuring compatibility to standardized conventions and guidelines. These frameworks are mostly created by organizations like the W3C, and they provide a bedrock for the deployment of DIDs across various ecosystems.
- **Method**: Method part defines a protocol or mechanism used for creating and maintaining DIDs within a certain ecosystem. Varying methods may involve different cryptographic algorithms or diverse distributed ledger technologies to develop and complete DIDs. Examples of methods are "ethr" for EthDIDs-based DIDs and "key" for standard key-based DIDs.
- **DID Method-Specific Identifier**: This identifier being unique for this method, differentiates the corresponding DIDs within the indicated ecosystem. It works as the reference point on DID documents finding and network interactions enabling within the decentralized systems.

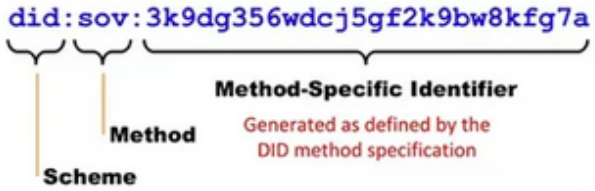

Figure 4.1. Decentralized Identifiers (DID) URI syntax.

DID Document

A DDO outlines the basic details connected to a certain DID, being the main part of the identity system. It typically includes [1]:

- **Verification Methods**: Public keys or cryptographic objects serving as authentication and verification purposes.
- **Authentication Methods**: Holder of the DID is verified through specific validation mechanisms applied.
- **Service Endpoints**: Identifiers referring to services or endpoints associated with DID subject, allowing interactions and data exchange to be completed.
- **Timestamps**: Proof data records, which host the verification history data or temporal metadata, enhancing transparency and auditability.
- **Signature**: Cryptographic signatures used for the purpose of security and trust of the DDO.

DID Resolution

DID resolution is the procedure for transforming a DID into the corresponding DDO that allows for retrieval of information related to a particular DID. This procedure of querying distributed ledgers or repositories seeks the relevant DDO [8]. Upon the resolution, verifiers obtain crucial attributes and secure materials, thus being able to have secure interactions and identity verification within the delegated systems. In addition to that, DID resolution enables interoperability by standardizing mechanisms for use of DID associated data as well as for its access and interpretation.

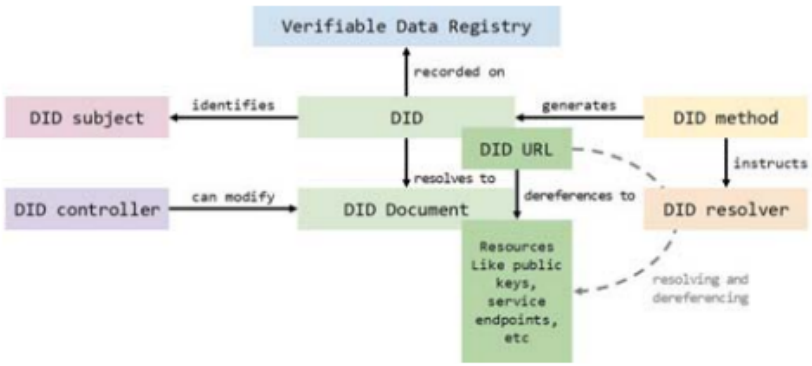

Figure 4.2. The basic components of DIDs architecture.

4.3.2 Verifiable Credentials

In a SSI framework, VCs represent the most crucial element as they permit identification and authentication in a decentralized manner. Briefly, a VC represents a claimed certified identity, stored by an authorized user, residing in their digital wallet, and comprising integrity features that can be verified by issuing entities [27]. They consist of attributes denoting assertions as well as ensuring the truthfulness of data provided to create a user profile. VCs have an important role in enabling individuals to direct their identity thus able to conduct secure and privacy-preserving communication in the digital ecosystem.

Syntactic Components of VCs

The syntactic structure of a VC encompasses several key components [40]:

- **Context**: This element defines a shared set of terms for interoperability among different systems giving option of using short aliases mapped to complex URIs that define the attributes and values for specific credentials.
- **Id**: An additional attribute which allows the unique identification of entities within a credential, usually by using a URI or a DID.
- **Type**: Mandatory specification which identifies the kind of credential, allowing software systems in processing and verification.
- **CredentialSubject**: Functional for stating the claims focused on one or several subjects of the credential, as well as for presenting the needed details.
- **Issuer**: Represents the entity issuing the credential and including information like issuer identifier and further metadata.
- **IssuanceDate**: Shows the date and time when the credential is firstly valid, particularly important to determine whether it is still valid.
- **Proof**: Comprises cryptographic proofs needed to verify the authenticity and integral of the credential, including mechanisms like digital signatures
- **ExpirationDate**: May be included optionally to mark the validity duration of the credential which must be relevant over the time.
- **CredentialStatus**: Gives the current valid status of the credential, whether active, suspended, or expired.

Figure 4.3. A simple example of verifiable credential.

Verifiable Presentations

VPs are verifiable proofs of a person's claims that only provide selective disclosure of identity data to verifiers [40]. They are usually derived or generated from one or more VCs that have metadata and crypto signatures that can be verified by the recipient. Credentials from different sources are combined through VP which allow fast and privacy-preserving interactions within the SSI framework [27].

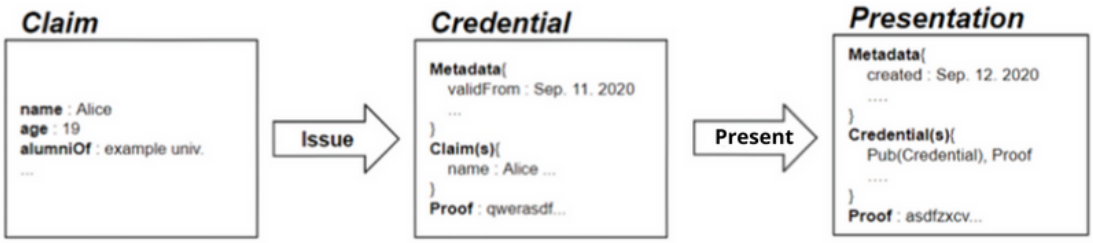

Figure 4.4. Core data model in W3C's VC.

Zero-Knowledge Proof

ZKPs are a type of proof that allows one to demonstrate a value without actually revealing the value itself [40]. Essentially, ZKPs provide privacy and support

selective discloser of credential attributes in the VCs ecosystem. The ZKPs enable hidden proving with the provision holding claims of participants without revealing the sensitive information, they provide privacy-protecting interactions in the SSI ecosystems. This provides the opportunity to issue zero-knowledge verifiable proofs of possession, to allow holders to disclose important information without revealing their secret. This conceals the information disclosed to verifiers by individuals utilizing ZKPs while remaining in charge of their personal data.

4.3.3 Verifiable Data Registries

Verifiable data registries form a key pillar in SSI systems which provide a decentralized mechanism for the secure storage of identity-related data such as DIDs and VCs. These registries guarantee the validity and reliability of the data without any need to the central authority [40].

Blockchain as a Distributed Verifiable Data Registry

Blockchains provide distributed verifiable data registries as a critical feature for SSI infrastructures. Being a distributed ledger technology, blockchains ensures immutability and decentralization, which makes them a perfect choice, for recording and validating transactions in a trustless environment [1].

In the case of blockchain-based SSI frameworks, the transactions containing the assertions of the identity and the verifications of these assertions are recorded in blocks linked chronologically. Each block is given a cryptographic hash of the previous block, thus the blockchain is made secure and the ledger is done without interruption. Through cryptographic signatures of stored data and events, Blockchains provide a tamper-proof encryption repository for identity data.

Distributed Hash Table in SSI

As in blockchain, DHTs ensures a decentralized way of storage and retrieval of data within the scope of SSI frameworks. DHTs are the decentralized data store components and are known for their rapid lookup capabilities based on the key-value pair [1]. Technology platforms such as the IPFS utilize DHTs to make the storage and retrieval of archival data more efficient, increasing the overall performance and improving the decentralization of content delivery.

IPFS utilizes the DHT technology, distributing data across multiple nodes of a decentralized peer-to-peer network. Such an approach not only increases data resilience and availability, but also minimizes the dependence on a centralized server for data retrieval. In addition, the DHT structure used in systems like

IPFS facilitates content addressing, which makes it possible for users to retrieve the data using the unique hash addresses while not having to depend on any centralized storage infrastructures.

4.4 Architecture of Decentralized Identity

The Decentralized Identity architectural framework consists of four layers that correspond to different building blocks, each of them being a critical part in implementation and operation of the system. Such layers are built to offer trust and smoothen interactions among parties existing in the decentralized identity ecosystem. The implementation of this infrastructure is based on the works of the pioneers in the field, including papers from the Trust over IP Foundation [9].

4.4.1 The Four Layers

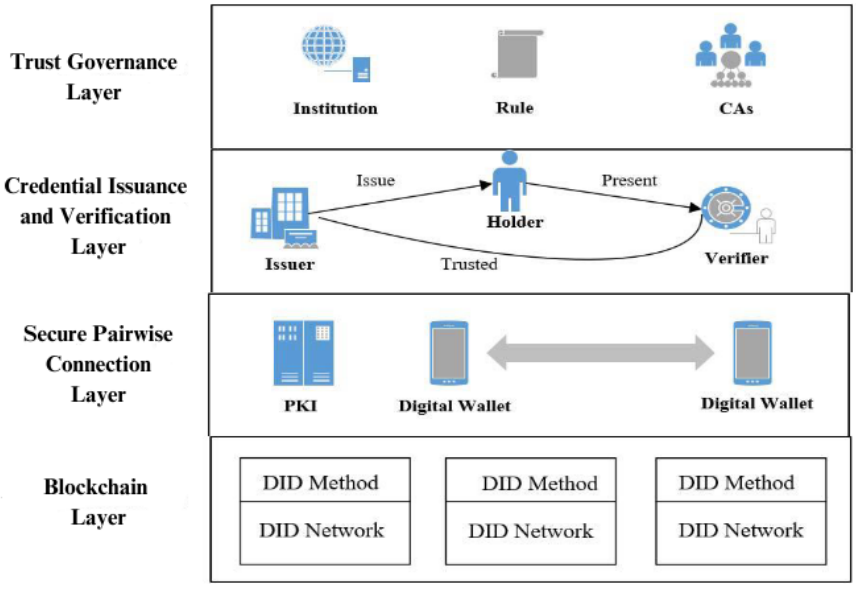

Figure 4.5. SSI architecture.

1. **Blockchain Layer (Layer 1)**:
   - It works as a base layer, making a retrievable data registry and storage for DIDs and the associated DDOs [9].
   - Deals with the definition, storage and management of VCs and their definitions, schemas and descriptions respectively.

- It facilitates credential revocation by maintaining the Revocation Registry record when a credential issuer revokes a credential.
- Incorporates proof of consent mechanisms for the exchange of data among entities securely.

2. **Secure Pairwise Connections Layer (Layer 2)**:
   - Enables secure communication between agents and digital wallets through share pairwise links.
   - Is in charge of the creation and maintenance of secure links between two parties, which are the agents.
   - It makes communication of personal messages safely through encrypting and decrypting.
   - Its job is to deal with the digital wallet information, guaranteeing the secure storage and management of credentials.
3. **Credential Issuance and Verification Layer (Layer 3)**:
   - Enables the issuer to issue VCs to holders in order to form the trust triangle.
   - Enables the holders to have their digital wallets and accommodate the credentials from multiple issuers.
   - Provides ability to combine different claims from several VCs into one complex compound proof for attestation.
   - Enables verifiers to verify the holder's proof using the VCs and disregarding the issuer.
4. **Trust Governance Layer (Layer 4)**:
   - Establishes governance structures, policies, and contracts to build up trust among parties participating in the ecosystem.
   - Specify a set of rules in which entities can stay within the guidelines of the decentralized identity system.
   - Uses Trust Anchors, Auditors, or known governance organizations which are responsible for issuer and verifier's integrity.

The security measures are implemented across the layers 1 and 2, using cryptography primarily [9]. This helps improve security to a great extent. These tasks

range from the generation of public and private keys through the associated cryptographic operations during the interval of credential lifecycle (issuing, generating, verifying, and revocation), and to the encryption and decryption of messages that are exchanged between wallets and agents.

4.5 The SSI Trust Triangle

In the realm of decentralized identity management, the SSI architecture operates on the basis of a trust triangle involving three primary actors: Holder, Issuer and Verifier. Clearly, the role and interactions of these components are the determining factor in understanding the dynamics of decentralized identity systems.

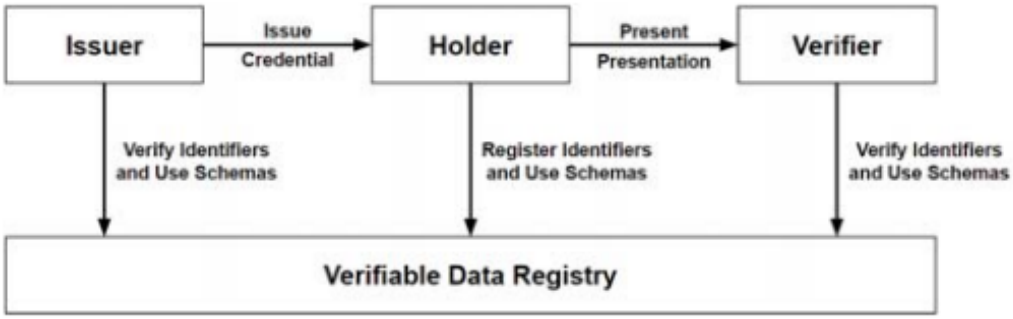

Figure 4.6. Basic concept of W3C's VC.

4.5.1 Key Actors of SSI

- **Holder**: The Holder is the individual user within the decentralized identity ecosystem. Equipped with a digital wallet application, the Holder generates their DIDs. The DID acts as a unique identifier for the user and belongs to a digital wallet in which VCs are stored. VCs are cryptographic attestations, which are issued by trusted parties, the Issuers [12].
- **Issuer**: Issuers are the organizations or entities that are responsible for verifying the identities of users and issuing VCs. These credentials are signed with the Issuer's private key and therefore establish the authenticity and integrity of the information contained within. After the verification, the VC is linked with the Holder's DID and it is stored in their digital wallet.
- **Verifier**: Verifiers are entities that use the provided VCs to verify the users' identities. Upon the presentation of the Holder's credentials by the Holder,

the Verifier runs the public DID on the blockchain to confirm the credibility of the issuer. The verification of the cryptographic signatures embedded within the credentials by the Verifier ensures that the presented information was not tampered with and originates from a trusted source.

4.5.2 Workflow of Verifiable Exchange

1. **Creation of Decentralized Identity**: Users start the whole procedure by generating a DID for each piece of data that they want to share. This DID functions as the user's identification post within the decentralized network in a cryptographic manner [5].
2. **Issuance and Validation of Verifiable Credentials**: On request, an Issuer validates the Holder's DID usually by referring to an immutable public ledger. After validation, the issuer creates and signs the VC containing the relevant information. This credential is paired with Holder's DID and stored in their digital wallet.
3. **Presentation of Verifiable Credentials**: The Holder uses the stored VCs in their digital wallet to establish a specific claim. Through signing the provided credential, the Holder builds a VP of their claimed identity.
4. **Verification by the Verifier**: After receiving the presentation, the Verifier performs a set of verifications to ensure its authenticity is intact. This process entails checking the cryptographic signatures appearing in the credentials and authenticating the nearest public DIDs on the blockchain. The Verifier confirms the authenticity of the delivered information and thus guarantees the credibility of the reported identity [1].

Chapter 5

Cryptography and Cybersecurity Aspects

Cryptography and cybersecurity serve as vital elements in a digital identity management system, in particular, in the decentralized systems of SSI, which are based on blockchain. Cryptography enables the data to be tamper-proof, secure and privately shared through mechanisms such as hashing and digital signatures. This chapter explains the role of cryptography in SSI while providing steps on how to create and verify integrity proofs, also, we demonstrate a digital signature algorithm implementation of EdDSA.

As mentioned earlier, cybersecurity also plays a key role in protecting SSI systems from various threats and vulnerabilities. In this chapter we will highlight how, despite the inherent security features of blockchain technology, SSI models still face several security challenges. Once this is done, potential attacks on SSI systems, which arise precisely from these vulnerabilities, will be evaluated.

5.1 Cryptography behind SSI

5.1.1 Data Integrity in SSI

Among numerous cryptographic aspects in the SSI approach, one of the most significant is perhaps the concept of proof used to verify the integrity and authenticity of VCs within the trust triangle, which forms the core of the SSI model. In the rest of this subsection, we delve into how a proof is created, verified, and what are the parts that make it up.

Proof Creation and Verification

The essential steps for creating a data integrity proof in an SSI system entail [41]:

1. **Transformation**: This initial step entails the preparation of data inputs through converting them using canonicalization algorithms and binary-to-text encoding.
2. **Hashing**: Cryptographic hash factories, for instance SHA-3, BLAKE-3 etc., provides hash identifiers that are resistant against hash collision and hence ensure integrity and safety of data.
3. **Proof Generation**: Implementing the proof serialization algorithms the values are calculated to make the input data secure from any inadmissible change. Well-known examples are, digital signature and proofs of participation.

Figure 5.1. Process to create a cryptographic proof.

Next, verifying the proof requires three steps to be completed [41]. The initial steps comprise the Transformation and Hashing processes which were previously presented, followed by the Cryptographic Proof Verification which involves some specific algorithms that confirm the reliability of the data being put in. It could mean checking up on the validity of digital signatures or verifying participation proofs.

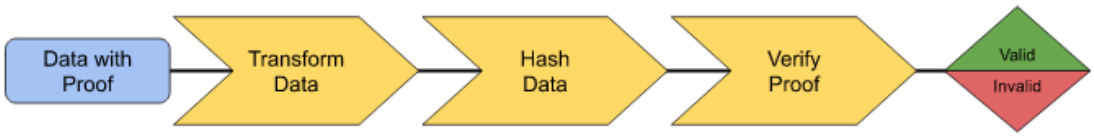

Figure 5.2. Process to verify a cryptographic proof.

Proof Data Model

A data integrity proof in SSI systems consists of a number of parameters, some mandatory and some optional; among the first ones there are [41]:

- **Type**: Identifies the exact proof type for cryptographic proof, thus, allows the relevant fields that are used to validate and verify the provided proof.
- **Proof Purpose**: Explicates the purpose of the proof, protects against the misuse and guarantees its proper implementation.
- **Verification Method**: Provides a description of the procedure and the data that may be required to verify the proof, which may involve cryptographic proofs or other verification tools.
- **Proof Value**: It contains the encoded binary data that is required for proof verification, which is the process that uses the specific verification method.


```
{
  "title": "Hello world!",
  "proof": {
    "type": "DataIntegrityProof",
    "cryptosuite": "jcs-eddsa-2022",
    "created": "2023-03-05T19:23:24Z",
    "verificationMethod": "https://di.example/issuer#z6MkjLrk3gKS2nnkeWcm
cxiZPGskmesDpuwRBorgHxUXfxnG",
    "proofPurpose": "assertionMethod",
    "proofValue": "zQeVbY4oey5q2M3XKaxup3tmzN4DRFTLVqpLMweBrSxMY2xHX5XTYV
8nQApmEcqaqA3Q1gVHMrXFkXJeV6doDwLWx"
  }
}
```


Figure 5.3. A simple example of cryptographic proof.

In addition, other optional parameters can also be included within a proof, such as [41]: id, created, expires, domain (useful for representing the security domains in which the proof is meant to be used, ensuring its proper application within specified contexts) ,challenge (only if the domain is specific, to mitigate replay attacks), previousProof and nonce (useful to increase privacy by decreasing linkability, that is the result of deterministically generated signatures).

5.1.2 Digital Signature using EdDSA

In the manner before, the usage of the approach of digital signature is very common on the way of creating a proof. As a part of different algorithms, one that has

the most outstanding impact is EdDSA, which its implementation we will cover in this subsection. Using the elliptic curve cryptography, the EdDSA is one of the most known for its speed and security.

EdDSA Key Generation

Key generation in EdDSA consists in the creation of a private-public key pair. The private key ($priKey$) consists of 32 octets of cryptographically secure random data (256 bit). For the public key ($pubKey$) the hashing algorithm (SHA-512) is applied to the private key as the first step in the process. And finally, the hashing product is converted using specific bit operations, as well as scalar multiplication, in order to get the public key [34]. Specifically, this process involves the following steps:

1. **Hashing**: The first step is hashing the private key (priKey) of 32-byte, using SHA-512, and store the result in a 64-octet buffer denoted as h'; only the first 32-byte will be considered for the $pubKey$.

2. **Buffer Pruning**: Then, some bit manipulations are performed on the buffer to ensure compliance with specific encoding requirements, for example some specific bits are clear in the first and last octets.

3. **Scalar Multiplication**: The pruned buffer is then considered as a secret scalar represented by a little-endian integer and a fixed-base scalar multiplication $[s']B$ is performed, where B represents a base point on the elliptic curve.

4. **Encoding**: Finally, the resulting point $[s']$ is encoded, through manipulation of the y-coordinate values of the curve. In this way the $pubKey$ has also been generated.

EdDSA Signing

Signatures are formed by using the private key to sign the message. This process is based on combining the message and the private key by using different cryptographic operations [34]. In our case, the message corresponds to the value of the proof to be signed. Specifically, this process involves the following steps:

1. **Hashing and Scalar Derivation**: The private key ($priKey$) is hashed using SHA-512 to obtain a digest (h'), of which the first half is used to create the secret scalar s', while the second half is denoted as prefix.

2. **Message Hashing**: The message (M') is hashed using SHA-512, along

with prefix, context information (CTX) and a flag (FLG), so as to obtain a 64-octet digest (i).

3. **Point Calculation**: The point $[i]B$ is calculated, where B is the base point on the elliptic curve. This calculation includes the reduction of i modulo L, that is, the group order of B, and finally the result is encoded, obtaining R.

4. **Digest Calculation**: Then, another digest is computed by hashing the concatenation of CTX, FLG, R, the public key (A), and a modified hash of the message ($PH(M')$), and from this is obtained the little-endian integer k.

5. **Scalar Calculation**: $S' = (i + k \cdot s') \mod L$ is calculated.

6. **Signature Construction**: Finally, the signature is created by concatenating R (32 octets) and S' (32 octets, with the three most significant bits at 0) in little-endian encoding.

EdDSA Verify Signature

Signature verification attests to the authenticity of a certain signature by utilizing the public key, message (in our case the proof), and signature data. This method is based on decoding the signature, forming a digest out of the message and public key, and finally, verifying the group equation to ensure the integrity of the signature [34]. Specifically, this process involves the following steps:

1. **Signature Decoding**: Using the public key A as reference point, the signature is decoded into two 32-octet parts, which represent the point R and integer S', respectively.

2. **Digest Generation**: Then, a 64-octet digest is generated by hashing the concatenation of CTX, FLG, R, public key (A), and a modified hash of the message ($PH(M')$).

3. **Group Equation Verification**: Finally, the validity of the signature is verified by checking the group equation $[8][S']B = [8]R + [8][k]A'$, or alternatively, $[S']B = R + [k]A'$; where A' is the public key encoded.

5.2 Cybersecurity within SSI

5.2.1 Security in Blockchain and SSI

As previously outlined, in the context of SSI, blockchain may be used as a distributed ledger to implement certain security features for the system. Nevertheless,

this might not be sufficient enough because the SSI model still contains some vulnerabilities. Next, we present the main features of blockchain technology, useful for the SSI system, and after that we examine the security challenges according to the security assessment of the SSI system.

Security Features in Blockchain

Among the key security features of blockchain, those most crucial for decentralized identity management include [36]:

- **Tamper Resistance**: Data immutability is ensured via blockchain thus cryptographic hashing methods, linking each block cryptographically. Consensus Protocols like Nakamoto consensus and digital signature algorithms such as ECDSA mitigate the tampering of data.
- **DDoS Resistance**: The decentralized architecture of blockchains consensus protocols allows the reduction of DDoS attacks' impact, since they permit transaction processing even with offline network nodes. In this scenario, attackers must compromise a significant portion of the network to make it inoperable.
- **Double Spending Resistance**: Consensus protocols and transparent transactions permit the mitigation of double-spending attacks, while verification mechanisms guarantee transaction's validity, maintaining network integrity.
- **%51 Resistance**: Attacks made against consensus protocols require the attacker to gain majority control, threatening the integrity of the transaction history. Various consensus protocols have different susceptibility thresholds, so several robust security measures must be developed.

Security Assessment of SSI

The main challenges that SSI models must approach are [36]:

- **Dependency on Manufacturer Reliability**: SSI systems depend on TEEs supplied by the manufacturers, which is crucial for the security of the systems. The reliance put on the security model manufacturers come up with is also dubious and might introduce further vulnerabilities associated with single points of failure.
- **Data Availability and Memory Risks**: Storing verifiable credentials in a local storage in the SSI systems can decrease the accessibility and can expose issues related to memory corruption or misuse. The availability of the data

spot on without ruining the integrity is a big question.

- **Confidentiality Protection and Information Disclosure Risks**: Although SSI systems are using different methods such as anonymous authentication and ZKPs to protect privacy, risks for disclosing sensitive identity information still remain. The reasonable selective disclosure mechanisms are still under development. The implementation of these mechanisms should be improved to mitigate the risks.

5.2.2 Potential Attacks on the SSI System

The vulnerabilities identified in the previous segment could give malicious actors the push to conduct different types of attacks and lead to damage or steal identities of other people to the SSI system. These attacks can be grouped into three categories, and an attack tree can be defined for each of them in order to better analyze them.

Fake Identity Attacks

Fake identity attacks pose a major threat to SSI systems, using fake identities to gain access to services they are not authorized for. Attackers exploit vulnerabilities in the SSI architecture to create fake credentials. These attacks can damage the trust and reliability of the identity management system as a whole [36].

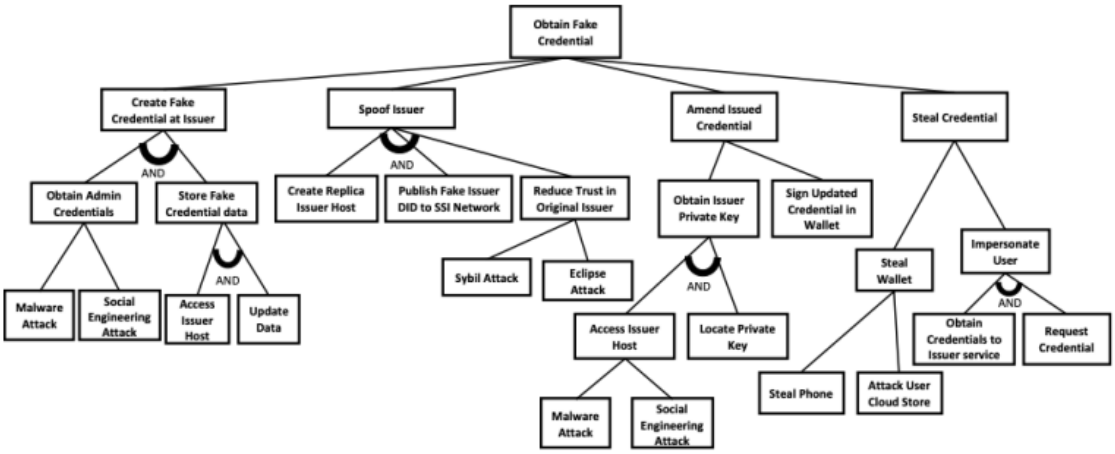

Figure 5.4. An attack tree of Faking Identity Attacks in the SSI system.

Attack Tree Analysis: In this attack scenario, attackers pretend to be trusted issuers by creating fake credentials, like DIDs and public keys. Once inside the network, they trick it into believing these fake credentials are real. In addiction, vulnerabilities in the architecture, like hacked network machines, can give attackers access to secret administrative credentials and keys, allowing them to change them. For instance, in the Eclipse Attack, attackers take control of the peer-to-peer network by redirecting connections to fake nodes, making the network accept the false credentials [31].

Identity Theft Attacks

Identity theft attacks use vulnerabilities, in the SSI system to steal sensitive and personal information, from user wallets without permission. These actions put individuals' privacy at risk and could result in types of fraudulent activities and improper use of personal data [36].

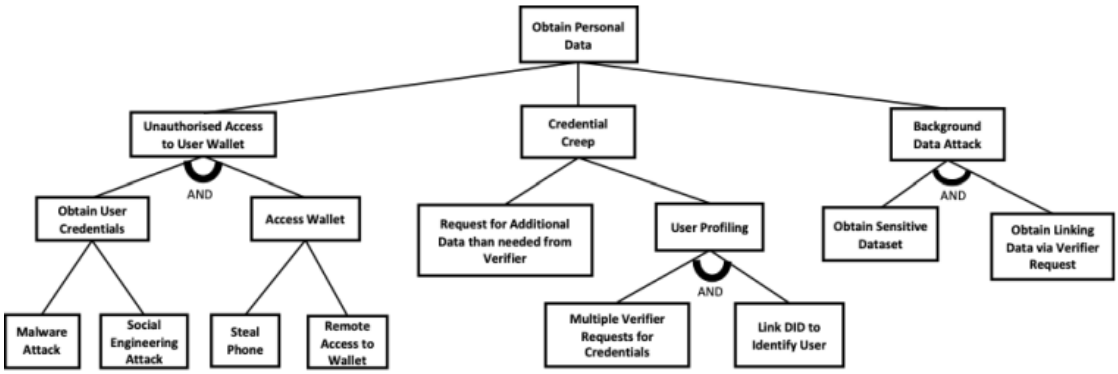

Figure 5.5. An attack tree of Theft Identity Attacks in the SSI system.

Attack Tree Analysis: In this type of attack, malicious actors might access data in wallets without permission by exploiting vulnerabilities in the SSI infrastructure. To do that, They could exploit authentication weaknesses or misuse credential verification methods to obtain additional personal information, a practice known as Credential Creep. The impact of identity theft attacks goes beyond users impacting the credibility and reliability of the SSI environment as a whole [31].

DDoS Attacks

DDoS attacks pose a serious risk to the availability and reliability of SSI system services. Through attacking the system with a large volume of traffic, the attackers intend to cripple the users' access to the system and breach the system's function [36].

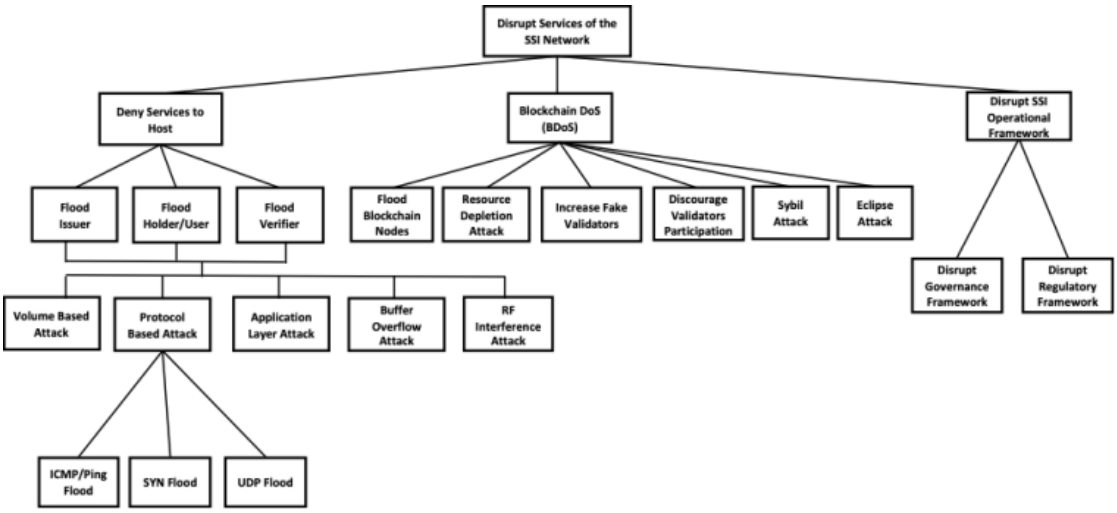

Figure 5.6. An attack tree of Distributed DDoS Attacks in the SSI system.

Attack Tree Analysis: This type of attacks aim at several parts of the SSI system, such as issuer, holder, and verifier hosts, along with the distributed ledger nodes. Malicious actors can exploit vulnerabilities to attacks using vulnerabilities in the blockchain infrastructure, for example flooding the nodes or disrupting the consensus process. Furthermore, operational frameworks can be targeted, leading to difficulties in governance and regulatory processes that are imperative for the functioning of the SSI ecosystem [31].

Chapter 6

Design and Implementation

Once the initial research was completed on the basics of blockchain and the current stage of the development of SSI models, the objective of this thesis was developed. The first goal is to make the standalone framework for the decentralized user identity management, applying the SSI models with the blockchain technology as the foundation. In this model, the identity management process consists of authentication and authorization procedures that do not rely on centralized systems or databases. The blockchain technology, smart contracts, and VCs are used for this purpose. Therefore, it guarantees a high level of seclusion and security for the personal and sensitive information of the users.

Subsequently, the aim was to integrate this system into an ongoing project at Links Foundation named Data Cellar. Chapter 7 will delve into what Data Cellar is and discuss the integration of the framework into that project.

In this chapter instead, the work will be analyzed in the design and implementation of the standalone framework for decentralized identity management. The work comprised several phases: analysis of existing projects in this domain, selection of the environment and libraries for developing the project, actual implementation of the project, and analysis of its limitations arising from the lack of robust support structures not yet commercially available, given the relatively short time since the development of blockchain-based SSI systems.

6.1 Framework Development Journey

In this section, the design phase of the standalone framework is explored, tracing the journey from initial research to the final selection concerning the development environment, namely the blockchain and libraries for managing DIDs. The choices

made regarding the management of another fundamental element of the SSI model, namely VCs, will be discussed in section 6.3.

6.1.1 Initial Research

Firstly, we conducted an examination of the primary IDMS blockchain products on the market. Several scientific papers and articles were examined to identify their technology choices, means of operation, and functionalities they offer [24], [33], [17]. The three primary solutions encountered were:

- **ShoCard**: This SSI technology underlines the decentralization providing every user with full control over their identities and data. *ShoCard* uses the public Bitcoin blockchain for digital identity and authentication, when users hold their identities on their devices using private keys.
- **The Sovrin Network**: Governed by *Sovrin Foundation*, it is an open-source solutions framework that provides users with sovereign and decentralized digital identities. It uses Sovrin ledger as a blockchain to support *Hyperledger Indy* framework and other *Hyperledger* libraries like *Aries* and *Ursa*.
- **The uPort System**: An open-source identity system management with SSI. It uses Ethereum blockchain for identity representation and IPFS for data storing.

Project	**ShoCard**	**Sovrin**	**uPort**
Blockchain Implementation	Bitcoin	Hyperledger Indy	Ethereum
Blockchain Type	Permissioned	Public Permissioned	Permissionless
Storage	Off-Chain	On/Off-Chain	Off-Chain
Interoperability	Yes	Yes	Yes
Selective Disclosure	No	Yes	Yes
Auth	Yes	Yes	No

Table 6.1. A comparative analysis of blockchain based SSI system.

In order to avoid the usage of the existing systems and to bring up a decentralized identity management system from scratch, the main work was done on the

fundamentals. The first step was the creation of a unique user representation that in the case of a decentralized system is ensured by the DIDs. Several libraries facilitating DID creation and management were explored [7], including:

- **Ethr-did**: Utilizes Ethereum's accounts and smart contracts to implement the registry for DIDs resolution. It stores DDO on the Ethereum blockchain.
- **Did-algo**: A framework for the *Algorand* blockchain, storing DDOs on IPFS. It offers a CLI software for managing operations and supports various cryptographic keys.
- **Did-indy**: Part of the *Hyperledger Indy* ecosystem, based on a permissioned blockchain with multiple logical ledgers. It enables role assignment to DIDs and provides mechanisms for VC issuance and verification.

6.1.2 Challenges and Failed Attempts

Initially, there were works focusing on the analysis of the *Did-indy*, which is also built on the *Hyperledger Indy* blockchain. Nevertheless, the development was challenged by the outdated *Indy* SDK and the absence of the comprehensive guideline.

These constraints thus provoked an exploration of other solutions, which resulted in a focus on *algo-did*, which works on the *Algorand* blockchain. It turns out that there were certain challenges that we encountered with *algo-did*, such as documentation issues, technical problems that remain unresolved, and the lack of community support that we have noticed. These problems led to a re-evaluation of the relevance of this system for the framework.

6.1.3 Final Choice

After a thorough screening and exclusion process, the spotlight focused on *ethr-did*, which is based on the Ethereum blockchain, one of the most widely used blockchains at present. The *ethr-did* system was not only robust but also well-documented, and *uPort* also made use of it, as discussed before, which made it the ideal option.

In the beginning, the *ethr-did* library's functionality was tested by running locally executed software, using *Infura* to communicate with the Ethereum blockchain. Developers can easily get access to the blockchain networks through *Infura* by creating an API key, and this is done without requiring them to directly manage full nodes on their devices [22].

However, anticipating the integration of the framework with the Data Cellar project, envisioned as a browser-accessible application for users, the framework

was promptly developed in the form of a browser DApp supported by the use of MetaMask. MetaMask serves not only as a digital wallet for users but also facilitates direct communication with the blockchain through the browser, eliminating the need for an API key and thus *Infura*.

6.2 Framework Main Components

In this section, the libraries, blockchain, and support tools utilized in the development of the standalone framework designed as a browser DApp for identity management will be explored. This framework operates under the decentralized identity management model based on blockchain technology, focusing on the creation and management of DIDs. The section excludes the management of VCs, which will be addressed in the subsequent section.

6.2.1 Ethr-did and Other Libraries

The foundation for standalone framework development is *ethr-did* library. Through *ethr-did*, we try to use Ethereum addresses as a fully self-directed DID so as to help easy key development and management for these identifiers. *ethr-did* standardizes the identity methods and provides a scalable mechanism for public keys and Ethereum addresses [38]. Thus, the collection of on-chain and off-chain data becomes possible.

This library relies on two additional libraries:

- **ethr-did-registry**: This library contains the Ethereum contract code allowing owners of *ethr-did* identities to update attributes in their DDOs. It provides an API enabling developers to interact with the contract functions using JavaScript, designed for resolving public keys for off-chain authentication [39].
- **Ethr-did-resolver**: Intended for use with Ethereum addresses or secp256k1 public keys as fully self-managed DIDs, wrapping them in a DDO. It supports the DIDs spec from the W3C Credentials Community Group and relies on the *did-resolver* library [11].

ethr-did together with *ethr-did-registry* and *ethr-did-resolver* are all based on *ethers*, which is a thin and well-known library for smart contract interactions and DApps development, most commonly used for digital wallets, like MetaMask.

The other hand, *ethr-did* makes use of the *did-jwt* library thereby introducing the signing and verification of JWTs using various algorithms. However, specific

functions from *did-jwt* were not utilized in the framework created.

In this case, only the *ethr-did* constructor was used to generate DIDs associated with users' Ethereum accounts and resolved them later using the *ethr-did-resolver* library. Supplementary functions within *ethr-did* were unnecessary for the purposes and thus remained unused.

Ethr-did Constructor

The majority of functions within the *ethr-did* library can be executed either locally or interact directly with the Ethereum blockchain, depending on the chosen approach for constructing the *ethr-did* object, whether or not the etherDidController class from the *ethers* library.

Now, let's analyze in detail the parameters required by the constructor [38]:

- **identifier**: Represents the identifier used for DID creation, which can be either an Ethereum address, a public key, or an Ethereum address as a string. (Required)
- **chainNameOrId**: Represents the name or ID of the blockchain network. (Optional)
- **registry**: Represents the address of the DID registry contract. (Optional)
- **signer**: Represents the JWT signer. (Optional)
- **alg**: Represents the signature algorithm, which can be either 'ES256K' or 'ES256K-R'. (Optional)
- **txSigner**: Represents the transaction signer. (Optional)
- **privateKey**: Represents the private key associated with the Ethereum address. (Optional)
- **rpcUrl**: Represents the RPC URL used to connect to the Ethereum network. (Optional)
- **provider**: Represents the Ethereum provider. (Optional)
- **web3**: Represents the web3 provider. (Optional)

The constructors logic checks, for the existence of an Ethereum provider or network details. If present, sets up the *txSigner* using an object that can sign transactions or generates one, from a *Wallet*. If a private key is defined and theres no existing *txSigner* object a fresh *Wallet* object is generated with the given key, which is then utilized as the *txSigner* for signing transactions.

The *EthrDidController* class is created to work with the ERC1056 contract, which's a standard used to represent identities, on the Ethereum platform. When the *EthrDid* object constructor is called it triggers the constructor of *EthrDidController* generating a DID with *did:ethr* on the designated network, in a process. This means that only DIDs generated through *EthrDidController* and actions executed using its functions are able to communicate with the Ethereum network.

6.2.2 Ethereum and Smart Contracts

As previously mentioned, for the purpose of building the following framework of decentralized identity management, the Ethereum blockchain has been chosen. Ethereum is possibly one of the most widely used blockchains in the world today due to the fact that it has completely redefined the concept of blockchain itself through the functions of smart contracts and through the support for the development of DApp [14].

In particular, Ethereum's main test networks, namely *Goerli* and *Sepolia*, were used, given the possibility of obtaining the relevant tokens via faucets. A smart contract was set up on these blockchains to serve as a registry, for keeping track of which DIDs linked to users Ethereum accounts had registered on the browser DApp. In addition, the smart contract used by *ethr-did* for managing DIDs, which is useful during their resolution, already present on *Goerli*, has also been deployed on *Sepolia*.

Deployment of these contracts on *Goerli* and *Sepolia* will be facilitated by using *Truffle*, a development framework for Ethereum that eases and hastens the creation of smart contracts and DApps. After correctly configuring *Truffle* [37], a brief script was created to deploy the contract, using the following commands:

- **"truffle compile"**: to compile the contract and obtain the ABI, which describes the methods it accepts, the parameters they accept, and the values returned.
- **"truffle migrate –network"**: to deploy the contract correctly on the given blockchain network, provided that the account used has enough tokens to cover the fees. During deployment, the address of the block where the contract had been stored on the blockchain was retrieved.

```
const registry = artifacts.require("DataCellarRegistry");
module.exports = async function (deployer, accounts) {
  try {
    await deployer.deploy(registry, accounts[0]);
    const registryInstance = await registry.deployed();
    console.log("DataCellarRegistry contract deployed at:",
   registryInstance.address);
  } catch (error) {
    console.error("Error deploying the contract:", error);
  }
};
```

Listing 6.1. Deploying DataCellarRegistry contract.

The Smart Contract

A smart contract is a self-executing contract wherein the conditions of the agreement are automatically captured in code. It executes the agreement conditions automatically and intermediaries are no longer a necessity.

In the scope of the project, Solidity language is used to write a smart contract that will perform functions of user registration management in a decentralized way. It is realized by updating a mapping in which a boolean value is attached to each Ethereum account to show the user status whether he is registered or not. In using the Ethereum blockchain which is inherently decentralized and immutable, the Smart contract ensures the integrity and transparency of the system, while eliminating the need for a central authority. Participants can register or unregister themselves safely and autonomously, with the contract recording these actions on the blockchain.

Functions of the smart contract:

1. **registerUser(address _userAddress)**: It registers a user in the registry. This function can only be called by the user, and the user must not be already registered.
2. **unregisterUser(address _userAddress)**: Unregisters the user. This function can be called only by the user itself, and the user should have registered before that.
3. **isUserRegistered(address _userAddress)**: Checks if a user is registered in the registry. This function can be used by anyone, and it returns a boolean value indicating whether the user is registered or not.

The contracts three functions all require an Ethereum account as input. The final function is read only displaying whether a user is registered or not while the other two functions alter the contracts state. These state altering functions need the contract to update its status on the blockchain leading to transaction fees being paid to miners for overseeing and validating these modifications.

```
pragma solidity ^0.8.2;
contract DataCellarRegistry {
    mapping(address => bool) private registeredUsers;
    event UserRegistered(address indexed user);
    event UserUnregistered(address indexed user);
    modifier onlySelf(address _userAddress) {
        require(msg.sender == _userAddress, "Can only
   perform action on yourself");
        _;
    }
    modifier onlyUnregistered(address _userAddress) {
        require(!registeredUsers[_userAddress], "User
   already registered");
        _;
    }
    modifier onlyRegistered(address _userAddress) {
        require(registeredUsers[_userAddress], "User not
   registered");
        _;
    }
    function registerUser(address _userAddress)
        external onlySelf(_userAddress) onlyUnregistered(
   _userAddress) {
        registeredUsers[_userAddress] = true;
        emit UserRegistered(_userAddress);
    }
    function unregisterUser(address _userAddress)
        external onlySelf(_userAddress) onlyRegistered(
   _userAddress) {
        registeredUsers[_userAddress] = false;
        emit UserUnregistered(_userAddress);
    }
    function isUserRegistered(address _userAddress)
        external view returns (bool) {
        return registeredUsers[_userAddress];
    }}
```

Listing 6.2. Smart contract for DataCellarRegistry.

6.2.3 MetaMask

One of the crucial factors that led to the construction of the framework as a browser DApp is MetaMask. MetaMask is a user-friendly Ethereum blockchain software wallet designed for convenient transaction operations [30]. In other words, MetaMask serves as an interface between the users and Ethereum DApps, providing access to Ethereum wallets via browser extensions or mobile applications. The advent of MetaMask has provided users with a convenient way to manage their cryptocurrencies, transact online, and engage DApp across different browsers and mobile platforms.

Specifically, the browser DApp is able to monitor the presence of the MetaMask extension in the browser being used and the existence of an Ethereum account in the wallet, which most of the time means that the extension is connected to the application. Additionally, the DApp can identify the primary account and the desired reference networks on MetaMask in real-time. As a matter of fact, MetaMask not only help in handling multiple accounts but also multiple network like *Goerli* and *Sepolia* which has been deployed the smart contract.

Finally, MetaMask provides the ability to securely store the private keys of DApp accounts in a secure wallet, allowing users to sign and execute transactions that will enable them to register or deregister without the private keys ever leaving the extension in clear or encrypted form.

```
const _updateWallet = useCallback(async (providedAccounts)
   => {
  const accounts = providedAccounts || await window.ethereum
   .request({method: 'eth_accounts' })
  if (accounts.length === 0) {
    setWallet(disconnectedState)
    return
  }
  const chainId = await window.ethereum.request({method: '
   eth_chainId'})
  setWallet({ accounts, chainId })
}, [])

useEffect(() => {
  const getProvider = async () => {
    const provider = await detectEthereumProvider({
   mustBeMetaMask: true, silent: true })
    setHasProvider(provider)
    if (provider) {
      updateWalletAndAccounts()
```

```
17        window.ethereum.on('accountsChanged', updateWallet)
18        window.ethereum.on('chainChanged',
    updateWalletAndAccounts)
19      }
20    }
21    getProvider()
22    return () => {
23      window.ethereum?.removeListener('accountsChanged',
    updateWallet)
24      window.ethereum?.removeListener('chainChanged',
    updateWalletAndAccounts)
25    }
26  }, [updateWallet, updateWalletAndAccounts])
```

Listing 6.3. Function for updating wallet and accounts.

6.3 Verifiable Credentials and Limitations

As outlined in section 4.3, VCs are crucial, in the SSI model. So, once the blockchain and libraries for DID support were determined, the focus of the research turned to finding for managing, issuing, and verifying VCs.

6.3.1 Utilization and Challenges

The idea was to make dual use of VCs within the SSI framework: first, to obtain user information certified by an entity, and then, having obtained this information, to create a VC issued and signed by the browser DApp, with a triple purpose:

- Ensuring that the user has registered in the registry on the blockchain, i.e., the smart contract uploaded to *Goerli* and *Sepolia*, through the browser DApp.
- Maintaining the information provided by the user within an immutable credential, being signed by the browser DApp.
- Allowing the user to present the VC issued by the DApp to other entities to demonstrate registration with the service.

In this way, the user could authenticate the browser DApp just if they offer the VP related to the VC issued by the DApp itself, from which the data will be read and the user would be allowed to perform certain actions, such as based on age, country of origin, or profession. Furthermore, all of this is carried out in a fully decentralized way, which means that the browser DApp contains no

centralized database which stores the user's sensitive information. The VC is the one maintained by the user, and then it is temporarily stored in a session cookie to allow them to perform certain functions once authenticated.

However, even though the concept is well founded and aligns completely with the standards outlined by the state of the art of the SSI model, two significant challenges emerged. These limitations arise from the absence of frameworks and resources, for overseeing and upholding these VCs:

- The absence, for now, of VCs issued by genuine certified entities, easily findable and provided by the user during the registration phase to allow the creation of the VC issued by the browser DApp. For example, there is currently no VC corresponding to an identity card issued by the Italian government or similar entities. This limitation forced the direct request to the user to manually enter their data via a form, which is automatically taken as "trusted" and inserted into the VC that will be issued.
- The absence of non-proprietary wallets capable of storing and managing VCs for different purposes and containing DIDs with any type of DID method. Such wallets are necessary to allow the user to securely store the VC provided and to present a VP, signed by them and containing only the necessary information requested. This limitation forced the direct request to the user for the VC provided, without being able to apply the concept of selective disclosure.

6.3.2 Solution Exploration

Another idea that emerged due to a possession of the MetaMask as a wallet for the Ethereum accounts of our users was *Masca*. *Masca*, being a MetaMask Snap (extension), adds support for decentralized identity management of DIDs, storage of VCs, and creation of VPs [29].

This framework appeared to be suitable for the requirements; however for VC creation and signing, *Masca* must use the MetaMask account of the user connected to the DApp browser at that time. By the default mechanism, it was not possible to designate one particular account as an issuer that could have represented an application. The only possible way to deal with the problem after contacting *Masca* Support on dedicated forums was to pass the VC payload from the user to the server, at which point it would be temporarily stored. Later, the application administrator would use another DApp to access the requests from the server and sign them, which in turn create the VC to be finally transmitted to the user via other means.

However, this solution was disregarded because it not only prohibited immediate user registration and authentication within the application but also necessitated the implementation of database within the service, thus eliminating the complete decentralization of the system.

6.3.3 Final Decision

The ultimate choice in this case relied on sending the VC from the browser DApp to the user, where it is saved on the users device along with a suggestion to keep it safe in a place, like a wallet if possible.

In terms of selecting a library to use for VC creation and verification, the library *did-jwt-vc* was selected. This library, also used by *Veramo*, the current implementation of *uPort*, has APIs such that they allow for the creation of a VC by sending a payload that contains the user's account-associated DID, along with that of the issuer, that is EtherDid object which represents the browser DApp [10]. The process of signing and verifying JWTs using the ES256K and EdDSA mechanisms, respectively, is accomplished via the invocation of the functions present in the *did-jwt* library.

Subsequently , the VC can be validated by a *Resolver* object that allows the resolution of the DDO. Inside DDO there are verification methods which contain the issuer of VC. These methods are extracted and used for the verification of the signature. Signature verification provides the assurance that the VC's signature complies with the public key that is related to the verification method. All these operations are invoked through calling for the functions within *did-jwt* library.

```
export const verifyVc = async (vc) => {
  const providerMetamask = new BrowserProvider(window.
   ethereum, parseInt('0x539', 16));
  const providerConfig = {
    provider: providerMetamask,
    chainId: '0x539',
    registry: getContract("0x539")
  }
  const resolver = new Resolver(getResolver(providerConfig))
  try {
    const verifiedVC = await verifyCredential(vc, resolver);
    return verifiedVC;
  } catch (err) {
    return false;
  }}
```

Listing 6.4. Function for verifying verifiable credentials.

6.4 Application Functionality Analysis

Let us now look in detail at how the standalone SSI framework, which was created for decentralized identity management, was built, following the blockchain-based SSI model, its functionality and the mechanisms it uses. As mentioned earlier, the framework was developed as a browser DApp, consisting of a frontend, developed with *React*, and a backend, built using *JavaScript*, which serves as the DApp server.

In addition, to solve problems with the secure storage of authentication tokens, within session cookies, the application is supported by HTTPS. To do this, *OpenSSL* and *mkcert* were used to generate SSL certificates, which are required to enable HTTPS functionality. Thus, this enhancement also allowed the overall security of the framework to be strengthened, aligning with established best practices for web application development.

```
useEffect(() => {
  const cookies = document.cookie.split(';');
  const authTokenCookie = cookies.find(cookie => cookie.trim
   ().startsWith('authToken='));
  if (authTokenCookie) {
    const authToken = authTokenCookie.split('=')[1];
    const { payload: { publicAddress, name, surname, email,
   profession, country, region } } = jwtDecode(authToken);
    setAuthState({ publicAddress, name, surname, email,
   profession, country, region });
  }
}, []);

const handleLoggedIn = (auth, publicAddress, credentials) =>
    {
  document.cookie = 'authToken=${JSON.stringify(auth)}; path
   =/; samesite=None; secure';
  setAuthState({ publicAddress, name: credentials.name,
   surname: credentials.surname, email: credentials.email,
   profession: credentials.profession, country: credentials.
   country, region: credentials.region});
  navigate('/');
};
```

Listing 6.5. Handling cookie and setting authentication state.

In the frontend, which corresponds to the most substantial part of the browser DApp, in addition to the complete manage the graphical user interface, the user

authentication process is also managed, also taking advantage of MetaMask support. This process consists of 3 steps: *Access* (or connection), *Sign up* and *Sign in* (or authentication). For each of them a different web page has been defined and within each there is always an "information" button that explains to the user what exactly he has to do at that precise moment.

Below, in the remaining subsections, the server's functions first and then each step of the authentication process within the frontend, will be examined in detail.

6.4.1 Server Implementation

Being decentralized, the DApp server does not contain any database. However, it performs two important functions:

- Generating the VC with the payload provided by the user and signed with the private key of the application administration, stored as an environment variable within a .env file, during the registration phase of a new user.

```
async generateJwtToken(signature, pubAddress, nonce,
    credentials) {
  if (!signature || !pubAddress || !nonce || !
    credentials) {
    throw new UnauthorizedException('Request should have
    signature, pubAddress, nonce and credentials.');
  }
  const msg = `I am signing a one-time nonce: ${nonce}`;
  const msgBufHex = eth_utils.bufferToHex(Buffer.from(
    msg, 'utf8'))
  const address = eth_sign_utils.
    recoverPersonalSignature({
    data: msgBufferHex,
    sig: signature,
  });
  if (address.toLowerCase() === pubAddress.toLowerCase()
    ) {
    try {
      const payload = await this.generatePayload(
    credentials, pubAddress);
      const accessToken = await this.jwtService.
    signAsync(payload);
      return accessToken;
    } catch (error) {
      throw new UnauthorizedException('Error generating
    token.');
```

```
18      }
19    } else {
20      throw new UnauthorizedException('Signature failed.')
      ;
21    }
22  }
```

Listing 6.6. Async function for generating JWT token.

- Creating and signing, using a secret also stored as an environment variable within the .env file, a JWT to be inserted into the session cookie. This JWT contains the user's data, which was contained in the VC issued to him, which he has to replenish during the authentication phase. The purpose of the JWT is to grant specific permissions to the user, based on the data in it, during that session. In addition, before doing so, a check is made on the correctness of a Nonce, which is signed by the user, within a message, during the authentication phase. This allows to increase the security level of the process itself, preventing replay attacks.

```
1  async generateVerifiableCredential(publicAddress,
     credentials) {
2    if (!credentials || !publicAddress) {
3      throw new UnauthorizedException('Request should have
       publicAddress and credentials.');
4    }
5    try {
6      const issuer = new ethr_did.EthrDID({
7        identifier: process.env.OWN_ADDRESS,
8        privateKey: process.env.PRIV_KEY,
9        alg: 'ES256K-R',
10       chainNameOrId: 1337,
11     });
12     const holder = new ethr_did.EthrDID({
13       identifier: publicAddress,
14       alg: 'ES256K-R',
15       chainNameOrId: 1337,
16     });
17     const payload = await this.generateVcPayload(
     credentials, holder.did);
18     const vc = await did_jwt_vc.
     createVerifiableCredentialJwt(payload, issuer);
19     return vc;
20   } catch (error) {
```

```
21      throw new Error('Verifiable Credential creation
        failed.');
22    }
23  }
```

Listing 6.7. Async function for generating verifiable credentials.

6.4.2 Access Process

Upon launching the framework, the user will be presented with a single button that will redirect them to the page where they can perform the installation of the MetaMask browser extension, which is essential for the proper functioning of this SSI framework, as it allows users to interact with the blockchain by signing transactions directly through their wallet.

Once MetaMask is installed, this one button will be replaced by another that will allow the user to connect the MetaMask extension, to the DApp and unlock the wallet, in case it is still locked, thus allowing the DApp to access the Ethereum accounts within it, representing the user.

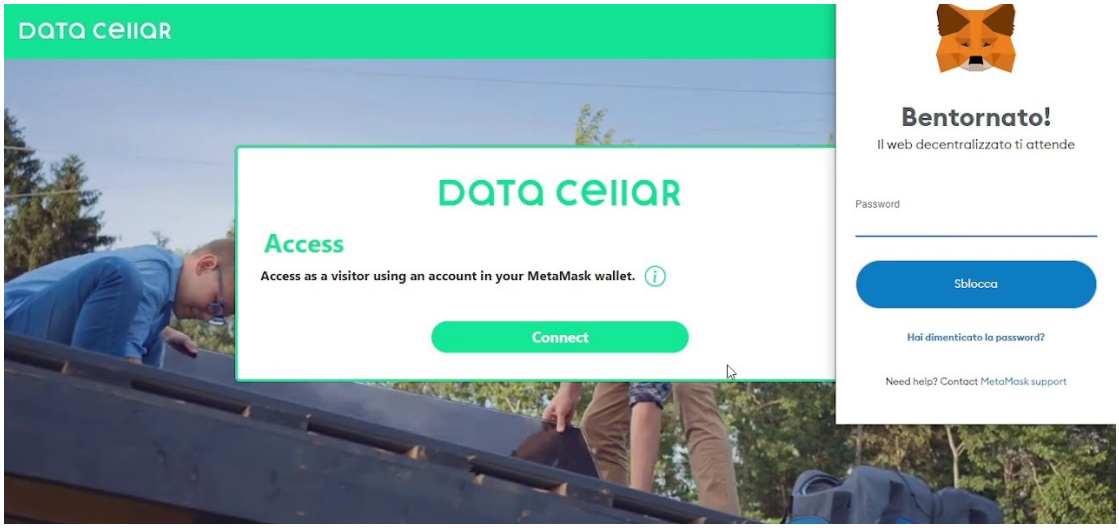

Figure 6.1. Access page with MetaMask installed.

After doing so, the user will be in the Home of the DApp, but will have logged in as a visitor, so they will only be able to view certain parts of the DApp and perform only limited functions.

However, two buttons will appear in the navigation bar: one for accessing the *Sign in* page and one for the *Sign up* page. In addition, two text boxes will appear showing the network and account selected at that particular time, within the MetaMask extension, providing the user with real-time information about the blockchain and the address they are working with.

6.4.3 Sign-Up Procedure

This step in the process allows the user to register with the selected account within the MetaMask wallet and using one of the blockchains, in which the smart contract that serves as the registry for the DApp has been previously deployed.

To begin, the user must fill in all the fields on the form presented to him; every value entered by the user is properly validated, so as to prevent possible security problems, but this still does not fully follow the SSI paradigm. This is because there are currently no authentic VCs from which to take this data (as explained in detail in the previous section). This data will constitute the VC payload, which the DApp browser, in the manner described above, will send through the server to the user.

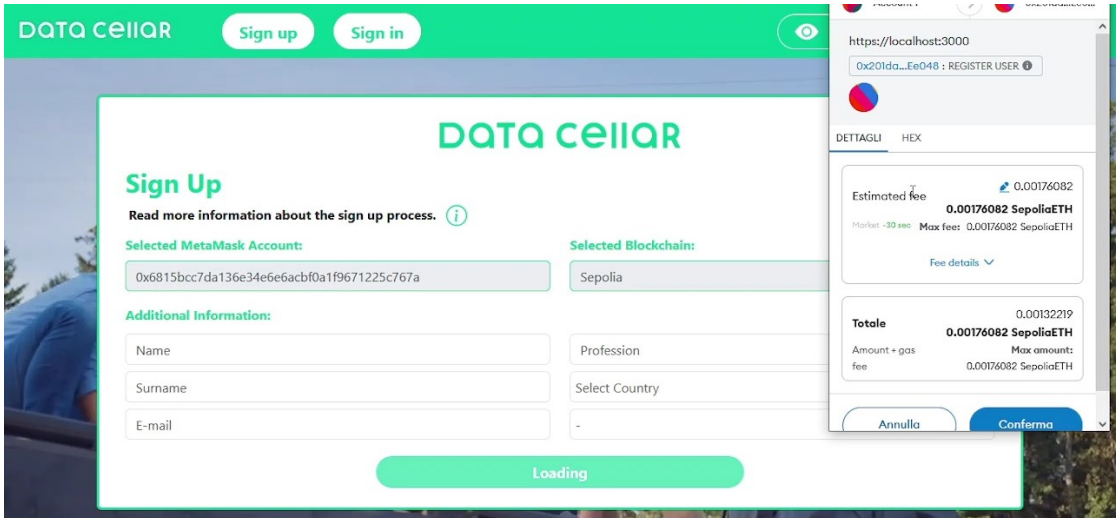

Figure 6.2. Sign Up page with confirm transaction pop-up.

Before receiving this VC, however, the user must register by adding his account in the DApp's smart contract, which serves as a registry, present on the referenced blockchain. To do so, he must make a transaction, via the MetaMask wallet, which will involve payment of fees, both from MetaMask and from the network used. This

registration is necessary not only to allow the application administrator to know how many users are registered and with which accounts, but also to control and issue only one VC per account.

```
const signupDataCellar = async () => {
  try {
    const provider = new ethers.BrowserProvider(window.ethereum, parseInt(wallet.chainId, 16));
    const signerPromise = provider.getSigner(wallet.accounts[0]);
    const signer = await Promise.race([signerPromise, timeoutPromise(10000)]);
    if (!signer) {
      throw new Error('Timeout');
    }
    const dataCellarRegistry = new ethers.Contract(contractAddress, contractAbi, signer);
    const registeredCheck = await isRegistered();
    if (!registeredCheck) {
      const tx = await dataCellarRegistry.registerUser(wallet.accounts[0]);
      await tx.wait(1);
      clearError();
      return true;
    } else {
      setErrorMessage('The selected account is already registered.');
      return false;
    }
  } catch (err) {
    if (err.message === 'Timeout') {
      window.location.reload();
    } else {
      setErrorMessage('The sign up operation on DataCellar smart contract failed.');
      return false;
    }
  }
}
```

Listing 6.8. Async function for signing up in DataCellar.

Once the registration is completed, the VC will be provided to the user who, pushing a button, can download it to his device and store it securely, as indicated.

6.4.4 Sign-In Mechanism

In this final and perhaps most important step, user authentication occurs, using the account selected within the MetaMask wallet and the blockchain with which the registration was made. A note reminds the user to change the network selected in the MetaMask extension in case it is incorrect.

To do this, the user must provide the VC issued during the sign up phase, from the browser DApp, for the selected Ethereum account; this is because there is currently no way to request the corresponding VP (as explained in detail in the previous section). The DApp will then check the validity of this VC, following the methods described above, to ensure both that it has been issued by the DApp and that it contains the DID, associated with the account selected by the user, in his MetaMask wallet, at that time.

```
{
  "credentialSubject": {
    "name": "Luca ",
    "surname": "Rota ",
    "email": "lucarota@gmail.com",
    "profession": "studente",
    "country": "Italy",
    "region": "Liguria",
    "id": "did:ethr:0xaa36a7:0
   x6815Bcc7DA136E34E6E6ACBf0a1F9671225c767A"
  },
  "issuer": {
    "id": "did:ethr:0xaa36a7:0
   x2F506eaaFfe39edD456cA74F13c74D6d80768Eb0"
  },
  "type": [ "VerifiableCredential" ],
  "@context": [ "https://www.w3.org/2018/credentials/v1" ],
  "issuanceDate": "2024-03-13T11:08:24.000Z",
  "proof": {
    "type": "JwtProof2020",
    "jwt": "eyJhbGciOiJFUzI1NkstUiIsInR5cCI6IkpXVCJ9eyJ2Y.
   yI6eyJAY29udGV4dCI6WyJodHRwczovL3d3dy53My5vcmcvMjAxOC9j.
   cmVkZW50aWFscy92MSJdLCJ0eXBlGOTx6lSAnLrhLp9iUOdUPl6H...
  }
}
```

Listing 6.9. Verifiable Credential JSON data.

If these checks are passed, the user will be asked to sign a message containing a Nonce, using the private key associated with their account, through the MetaMask wallet. This process does not involve any transactions on the blockchain and therefore does not require payment of fees. As explained earlier, this Nonce serves to increase the level of security and will be automatically sent to the server along with the signed message and the payload extracted from the VC previously provided by the user. After appropriate verification, if successful, the server will return a signed JWT containing the payload received.

Finally, this JWT will be placed in a token in the session cookies and the user will be authenticated. This corresponds to accessing the DApp as a member. In this way, the user can view the entire browser DApp, access its page, and perform all functionality, within the limits of the information in the session cookies. In fact, depending on the functionality the user wishes to perform within the DApp, the session cookie token will be decoded and, based on the information obtained, certain authorizations will be granted or denied to the user. In addition, a button will appear in the navigation bar that allows the user to log out of the account with which they have authenticated.

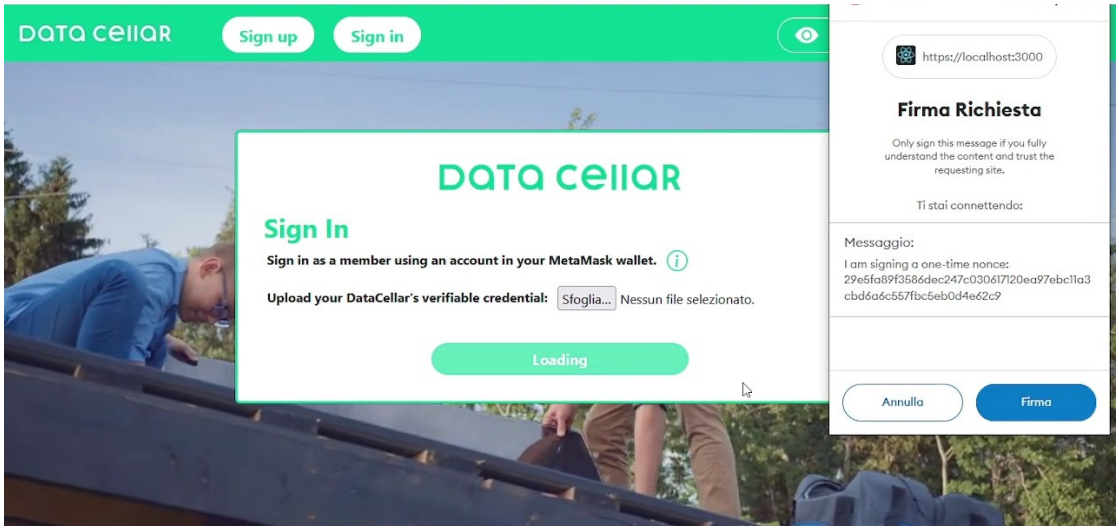

Figure 6.3. Sign In page with sign message pop-up.

Chapter 7

Integration in a Real Project

To demonstrate the real utility and effectiveness of the previously created decentralized identity management SSI framework, it was integrated as an authentication process into an ongoing project at Links Foundation, known as *Data Cellar*. Later, in addition to the integration of this system, the entire GUI of the application was also developed.

In this chapter, we will first deal with what *Data Cellar* is and what the initial state of the project was. Then, we will examine in detail the entire integration process and the final graphical result of the application.

7.1 Overview of Data Cellar

Data Cellar is an energy data center, geographically located within the European Union, whose main activities consist of the construction of a federated energy data space in order to enable the creation, growth, and support of local energy communities [6]. That initiative is based on an innovative rewarded private metering approach,, stressing successful integration, simplicity of the interactions, guaranteeing integration with other energy data spaces in the EU and providing the actors with the services and tools they need for their own actions.

As part of this four-year project, Links Foundation endeavors to decentralize various functionalities of the data center, resulting in enhanced security, speed, distribution, and additional benefits inherent to blockchain and decentralized architectures.

Naturally, being a project in development within the company and at the outset of this process, in its initial version, the set of functionalities offered by this

decentralized version is still limited compared to its standard operational version at the European level.

7.2 Initial Project Status

In its decentralized application form, the *Data Cellar* project comprised a set of smart contracts enabling various functionalities (briefly discussed later), invoked through a backend written in *NestJs*, with its interface generated via *Swagger*. *Swagger*, which is an API documentation tool, streamlines the documentation of RESTful APIs, so that the developers could focus on the code while not bothering about the manual creation of documentation.

7.2.1 Development Environment

The execution environment of the project was simulated using *Docker*. *Docker* is an open-source platform that is used for application creation, distribution, and execution, all within containers. *Docker* containers represent a modern form of virtualization that allows developers to bundle applications and all their dependencies (such as libraries, frameworks, and other components) into a self contained unit known as a "container" [13].

Specifically, in this case, three containers were executed:

- **Ganache:** An Ethereum-based private blockchain simulation containing pre-created accounts with visible private keys and a fund of 100 ETH.
- **Postgres:** The off-chain database for storing user information.
- **Redis:** The service for initializing and using queues to handle requests on the blockchain.

As can be seen from these containers, the user identity management system was still completely centralized, using a database implemented through *Prisma*, in which user IDs, accounts and private keys were stored.

7.2.2 Offered Functionalities

The key aspect of the initial version of the project was the digitization of energy data and its exchange among users through the purchase of one-time and periodic licenses. Features offered included:

- User registration, which includes the assignment of an address to the new user enabling them to buy and sell digital assets on the blockchain.

- Visualization of datasets and associated licenses.
- Upload of datasets and associated licenses.
- Purchase of licenses associated with datasets.
- Deletion of licenses associated with datasets and the datasets themselves.
- Visualization of DataCellar Token balance.

7.3 Integration Process

Following the guidelines provided, from the initial version, of the *Data Cellar* project, in the form of a decentralized application, the decision was made to continue using *Ganache* as a local network, running through a *Docker* container, rather than using Ethereum testnets, such as *Goerli* and *Sepolia*, as was done for the creation of the standalone framework for managing SSI.

First, the previously created framework was adapted to work with *Ganache*, and the configuration of MetaMask was then modified. In fact, among other functions, MetaMask allowed the addition of new local networks, such as *Ganache*, and the import of corresponding accounts into the wallet. As a final step regarding the blockchain aspect, the existing script, which loaded all the contracts used to execute the project's functionalities upon *Docker* startup, had the SSI framework smart contract acting as a registry and the one used by *ethr-did* for DID management added.

Subsequently, the code was modified, changing its structure almost completely. This was done because all the APIs, which invoked the functions, defined within the smart contract, had to be moved from the backend, where it was executed only because the users' private keys were stored in plain text in the database, to the frontend, to be executed using MetaMask, which allows transactions to be signed without exposing the users' private keys.

During this code change, the use of the database, thus *Prisma* and *Postgres*, the use of queues, as they are not supported by MetaMask, and the use of *Swagger* were completely eliminated, as a new GUI was created.

7.4 Final Application

The second version of the *Data Cellar* corresponds to the final application that has been built, in the form of a decentralized application that has been integrated

with the SSI framework and is further enriched with a dynamic and user-friendly frontend.

The functionalities of the second version of the *Data Cellar* can be divided into functionalities that relate to the SSI framework, those designed for visitors, and those for registered members.These functionalities include:

- **SSI functionalities:**
  - Connection to *Data Cellar* (access as a visitor)
  - Registration to *Data Cellar* (sign up as a member)
  - Authentication in *Data Cellar* (sign in as a member)
  - Delete your *Data Cellar* account
- **Visitor functionalities:**
  - View all datasets available in *Data Cellar*
  - View all available licenses for each dataset
- **Member functionalities:**
  - View the balance of ETH and DataCellar tokens
  - Convert ETH to DataCellar tokens
  - Add new datasets in *Data Cellar*
  - Create new licenses, single-use, or periodic, for your datasets
  - View, edit, and delete your own datasets
  - View, edit, and delete your own licenses
  - Buy licenses for datasets added by other users
  - View purchased licenses and reference datasets
  - Consume purchased licenses

7.4.1 Backend Implementation

To follow the guidelines, provided by the first version of this project, the backend remained written in *NestJs*, using the structure proposed by the language, consisting of Module, Controller and Service files. On the logical level, however, within the backend, which acts as a server, only the two features present in the framework for SSI management remained, namely:

- Verification of the user's signature and generation of an access token (which will be placed in the session cookie to verify user authentication)
- Generation of a VC demonstrating the sign-up to DataCellar (the user must provide it in order to access the dApp)

7.4.2 Frontend Development

As in the previous case, the frontend, realized using *React*, covers the most substantial part of the application. It contains not only the entire graphical interface but also the authentication process of the SSI framework, explained earlier, enriched by the de-registration functionality, along with all the APIs invoking the functionalities defined in the smart contracts of the *Data Cellar* project.

The GUI, in order to be user-friendly, has been enriched with confirmation modals for the most important operations, which the user can perform, and automatically disappearing error and success alerts, which cover all possible outcomes, of the various features the user can perform, within the application.

Finally, let us then briefly examine the two main pages that make up the *Data Cellar* project in its second version in the form of a browser DApp.

Home Page

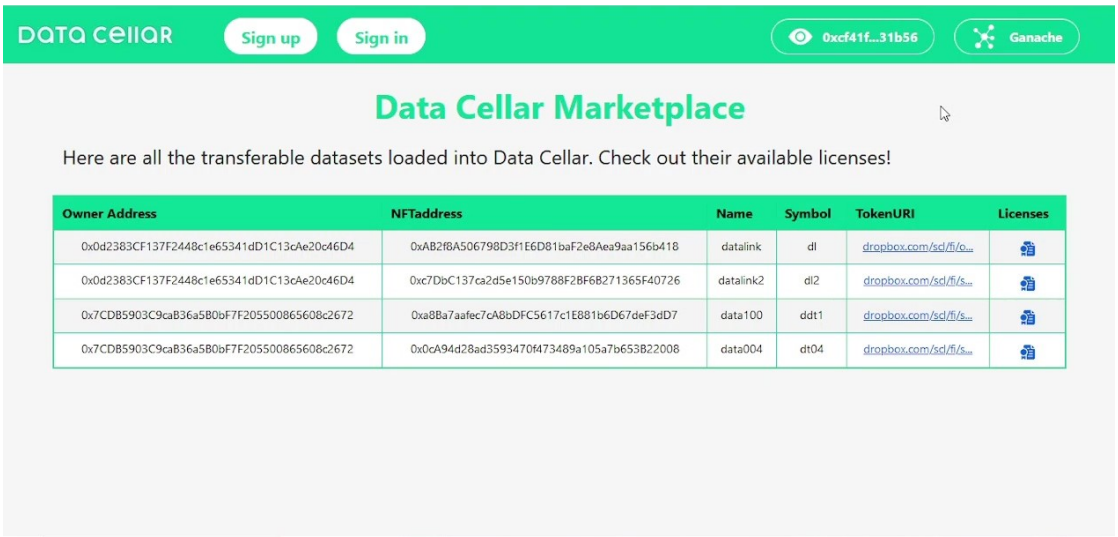

Figure 7.1. The marketplace in the Data Cellar Home page.

The initial page shows the *Data Cellar* marketplace, where all transferable datasets created by other users are visible. Various information is provided for each dataset, from the address of the owner, to the URI of the Token, which actually contains the energy data.

Clicking on the license symbol takes you to the corresponding page, which shows for the referenced dataset all available licenses, indicating for each of them various information, including the price in DataCellar Token, i.e., the currency used within the application.

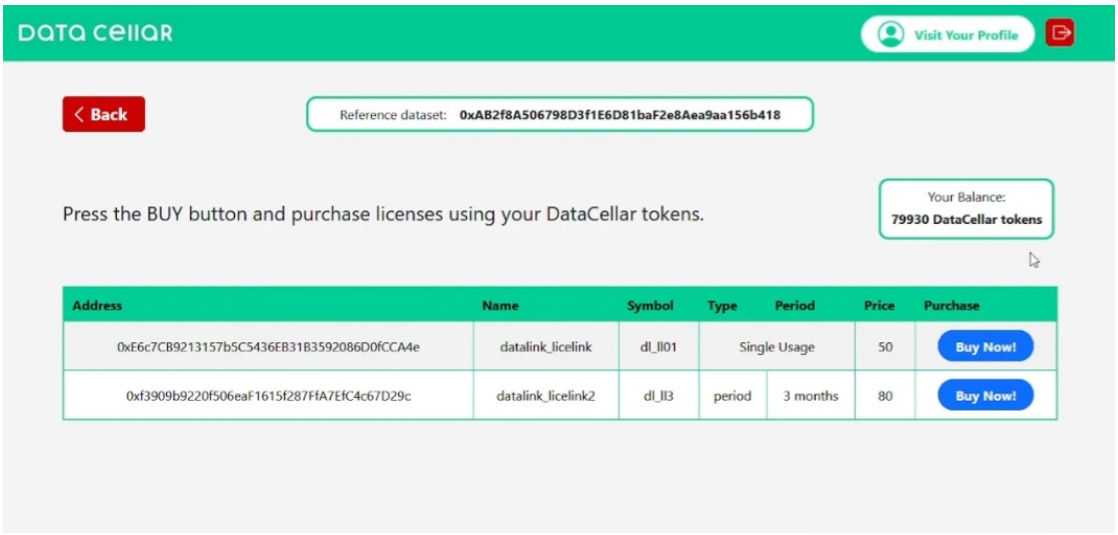

Figure 7.2. List of licenses for a specific dataset, after authentication process.

Authenticated members can purchase these licenses, either for a period or for single use, in the latter case, a modal allowing the definition of the quantity of licenses to purchase together is provided.

Profile Page

By clicking on the "visit your profile" button in the navigation bar, users can access their profile. Within it are various sections covering all the personal functionality executable by the user:

- **General Information:** Contains the user's information obtained from the token placed in the session cookie.

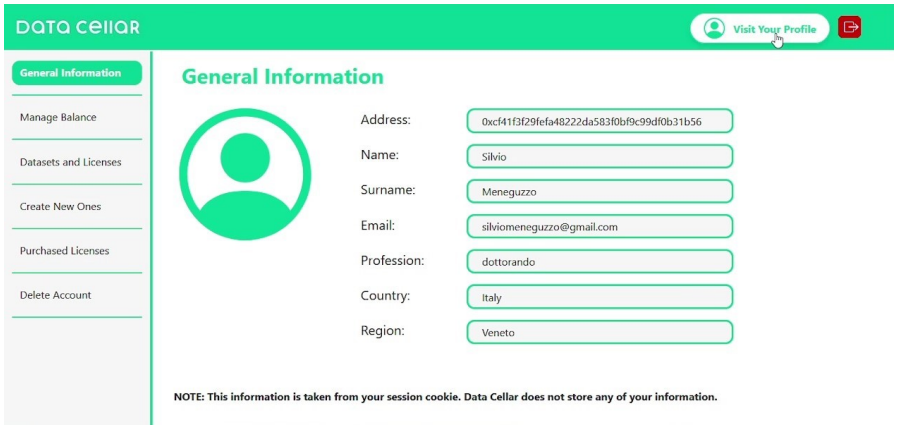

Figure 7.3. Page to view user's personal information.

- **Manage Balance:** Displays the user's DataCellar Token and Ethereum balance, also allowing conversion of new ETH to DataCellar tokens.

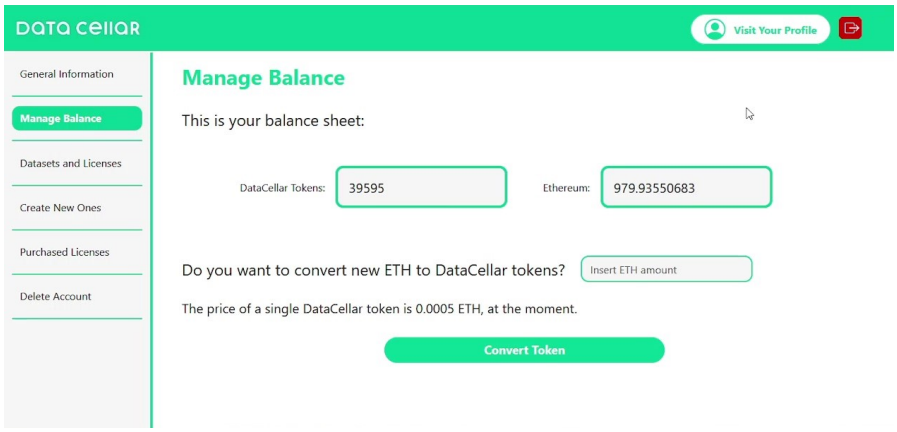

Figure 7.4. Page to manage user's DataCellar tokens.

- **Datasets and Licenses:** Displays all datasets created by the user; each can be edited or deleted, using the corresponding buttons that open the relevant modals. Also, by clicking on the license symbols, the user can view all the licenses he has created for that dataset. By accessing the dedicated page, the user can also edit and delete each license through the corresponding buttons and modals, as in the previous case.

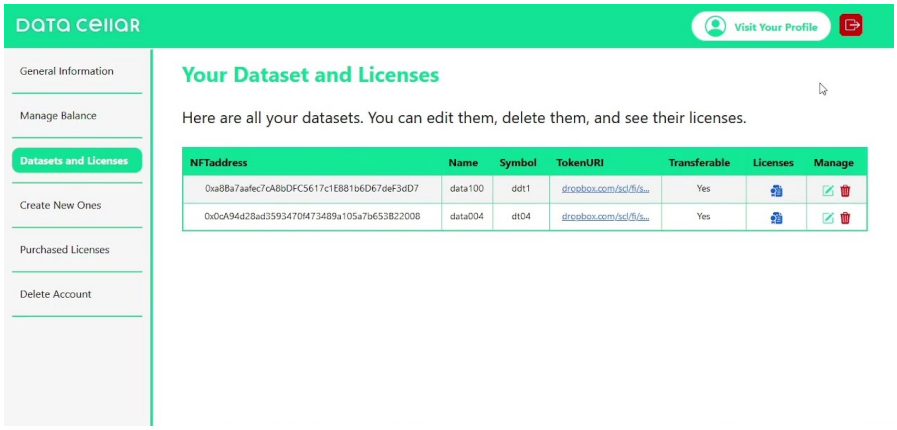

Figure 7.5. Page to view user's dataset and access their licenses.

- **Create new Ones:** Here the user can create new datasets, i.e., add new datasets to the marketplace, or create new licenses related to an existing dataset among its available ones. Datasets marked as transferable will be on the Home page of other users, allowing them to purchase their respective licenses.

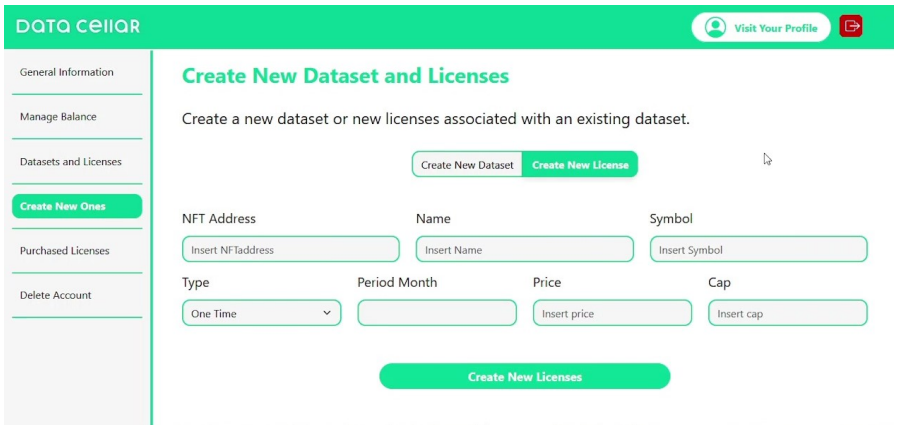

Figure 7.6. Page to create a new license or switch for new dataset.

- **Purchased Licenses:** Shows the list of datasets for which the user has purchased at least one license. These licenses are visible by clicking on the respective symbol leading to the dedicated page. Here the user can consume licenses, i.e., use them; periodic licenses can be used as many times as desired within the validity period, while single-use licenses can be used a number of times equal to the amount of tokens available for it.

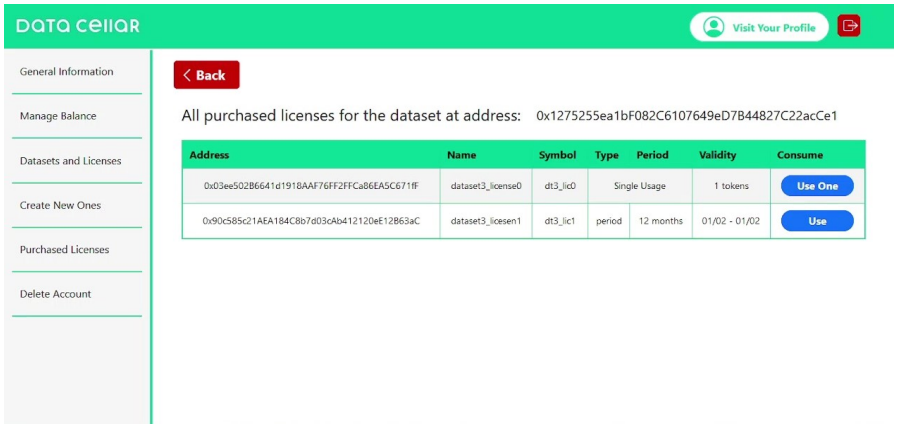

Figure 7.7. Page to view purchased licenses of specific dataset and use them.

- **Delete Account:** This last page developed the latest smart contract function created for the SSI management framework, which gives the user the ability to delete their account, i.e., deregister. However, this function was not originally designed for this specific project, so it has limitations. In fact, by deleting an account, the datasets, licenses and DataCellar tokens related to it and defined in the other smart contracts of the project are not deleted.

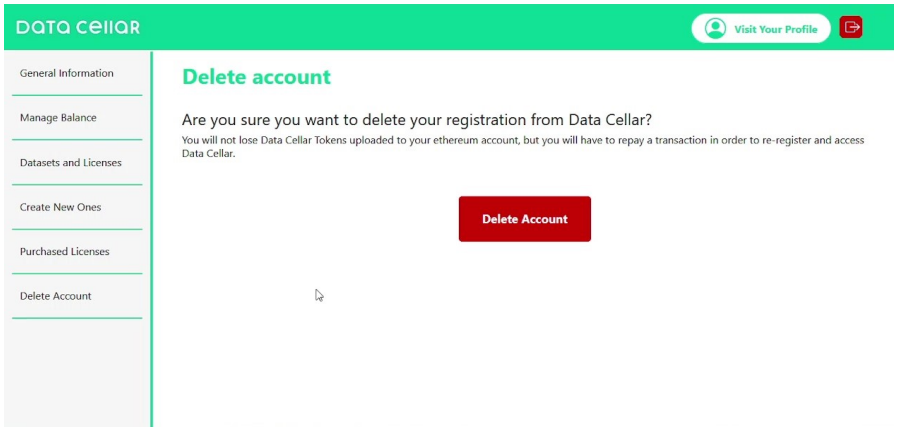

Figure 7.8. Page to delete user's account, performing de-registration.

Chapter 8

Conclusions and Future Works

In conclusion, this thesis delves into the concept of Self Sovereign Identity (SSI), presenting an approach, to handling digital identities. It is evident that SSI shows potential in transforming how identities are controlled in the world, offering several benefits, over conventional centralized systems. Nevertheless, it is crucial to recognize that both, SSI and the underlying blockchain technology, are still in their development stages.

This thesis outlines a framework for managing SSI throughout this exploration. By using Decentralized Identifier (DID), Verifiable Credential (VC) and blockchain technology this system delivers a decentralized solution for managing user identities. Designed as a decentralized browser application, it can be easily incorporated into various systems as an authentication tool.

Moreover, the integration of this framework into a real-world project has proven its usefulness, accompanied by the development of a user-friendly interface. However, it is important to note that given the stage of this field both the SSI framework and the project its integrated into show areas that could be developed in the future.

During the implementation phase of the SSI framework identified limitations highlight two advancements that can support this application and similar ones; establishing standardized verifiable credentials issued by accredited entities and creating non-proprietary wallets capable of securely storing and managing these credentials for selective information disclosure. These changes would enhance system autonomy and reliability while boosting user privacy and security.

Regarding integration with the real project, future progress could focus on using the information contained within the VCs, released by the application, for concrete

purposes. In addition, it is critical to improve the process of de-registering users from the application, to ensure proper management of the assets associated with users.

In summary, although this thesis has provided the groundwork for future work on identity management, through the creation and integration of an SSI framework, there is much more that needs to be done to boost this emerging field. Recognizing the identified limitations and exploiting the research opportunities, will take us a step closer towards a secure and privacy preserving digital identity ecosystem.

Bibliography

[1] Morteza Alizadeh, Karl Andersson, and Olov Schelén. "Comparative Analysis of Decentralized Identity Approaches". In: *IEEE Access* 10 (2022), pp. 92273–92283. DOI: 10.1109/ACCESS.2022.3202553.

[2] Yirui Bai et al. "Decentralized and Self-Sovereign Identity in the Era of Blockchain: A Survey". In: *2022 IEEE International Conference on Blockchain (Blockchain)*. 2022, pp. 500–507. DOI: 10.1109/Blockchain55522.2022.00077.

[3] Bitcoin.com. *How Bitcoin Transactions Work*. URL: https://www.bitcoin.com/get-started/how-bitcoin-transactions-work/ (visited on 02/14/2024).

[4] Business Reporter. *The History of Digital Identity*. URL: https://www.business-reporter.co.uk/technology/the-history-of-digital-identity (visited on 02/20/2024).

[5] Cisco FPIE. *Decentralized Identities Demystified*. URL: https://medium.com/cisco-fpie/decentralized-identities-demystified-49a65159196c (visited on 02/22/2024).

[6] *Data Cellar Project*. URL: https://datacellarproject.eu/ (visited on 03/06/2024).

[7] Andrea De Salve et al. "AlgoID: A Blockchain Reliant Self-Sovereign Identity Framework on Algorand". In: *2023 IEEE Symposium on Computers and Communications (ISCC)*. 2023, pp. 1162–1168. DOI: 10.1109/ISCC58397.2023.10218198.

[8] *Decentralized Identifiers (DIDs) v1.0*. World Wide Web Consortium (W3C). URL: https://www.w3.org/TR/did-core/ (visited on 02/22/2024).

[9] Blockchain for Decentralized Identity. *Blockchain for Decentralized Identity: Conceptual Architecture*. URL: https://medium.com/blockchain-for-decentralized-identity/blockchain-for-decentralized-identity-conceptual-architecture-982c41e446d9 (visited on 02/26/2024).

[10] decentralized-identity. *did-jwt-vc GitHub Repository*. URL: https://github.com/decentralized-identity/did-jwt-vc (visited on 03/08/2024).

[11] decentralized-identity. *Ethr-did-resolver GitHub Repository*. URL: https://github.com/decentralized-identity/ethr-did-resolver (visited on 03/06/2024).

[12] Dock. *Decentralized Identity and Self-Sovereign Identity: What's the Difference?* URL: https://www.dock.io/post/decentralized-identity#decentralized-identity-and-self-sovereign-identity-whats-the-difference (visited on 02/22/2024).

[13] *Docker*. URL: https://www.docker.com/ (visited on 03/08/2024).

[14] *Ethereum*. URL: https://ethereum.org/it/ (visited on 03/06/2024).

[15] Ethereum Foundation. *Proof of Stake (PoS)*. URL: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos (visited on 02/14/2024).

[16] Ethereum Foundation. *The Merge*. URL: https://ethereum.org/en/roadmap/merge (visited on 02/14/2024).

[17] Md Sadek Ferdous, Farida Chowdhury, and Madini O. Alassafi. "In Search of Self-Sovereign Identity Leveraging Blockchain Technology". In: *IEEE Access* 7 (2019), pp. 103059–103079. DOI: 10.1109/ACCESS.2019.2931173.

[18] GeeksforGeeks. *How does the Blockchain Work?* URL: https://www.geeksforgeeks.org/how-does-the-blockchain-work/ (visited on 02/08/2024).

[19] Varun Chandra Gupta et al. "An Intrinsic Review on Securitization using Blockchain". In: *2021 International Conference on Computational Performance Evaluation (ComPE)*. 2021, pp. 971–976. DOI: 10.1109/ComPE53109.2021.9752154.

[20] Humanizing the Singularity. *A Brief History of Digital Identity*. URL: https://medium.com/humanizing-the-singularity/a-brief-history-of-digital-identity-9d6a773bf9f5 (visited on 02/20/2024).

[21] IBM. *Blockchain Technology*. URL: https://www.ibm.com/topics/blockchain (visited on 02/06/2024).

[22] *Infura*. URL: https://www.infura.io/ (visited on 03/06/2024).

[23] Yue Jing et al. "The Introduction of Digital Identity Evolution and the Industry of Decentralized Identity". In: *2021 3rd International Academic Exchange Conference on Science and Technology Innovation (IAECST)*. 2021, pp. 504–508. DOI: 10.1109/IAECST54258.2021.9695553.

[24] Jayana Kaneriya and Hiren Patel. "A Comparative Survey on Blockchain Based Self Sovereign Identity System". In: *2020 3rd International Conference on Intelligent Sustainable Systems (ICISS)*. 2020, pp. 1150–1155. DOI: 10.1109/ICISS49785.2020.9315899.

[25] Rajesh Kumar Kaushal et al. "Immutable Smart Contracts on Blockchain Technology: Its Benefits and Barriers". In: *2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*. 2021, pp. 1–5. DOI: 10.1109/ICRITO51393.2021.9596538.

[26] Kyung-Hoon Kim et al. "Analysis on the Privacy of DID Service Properties in the DID Document". In: *2021 International Conference on Information Networking (ICOIN)*. 2021, pp. 745–748. DOI: 10.1109/ICOIN50884.2021.9333997.

[27] Seungjoo Lim et al. "A Subject-Centric Credential Management Method based on the Verifiable Credentials". In: *2021 International Conference on Information Networking (ICOIN)*. 2021, pp. 508–510. DOI: 10.1109/ICOIN50884.2021.9333857.

[28] *Links Foundation*. URL: https://linksfoundation.com/ (visited on 03/06/2024).

[29] *Masca Documentation*. URL: https://docs.masca.io/ (visited on 03/06/2024).

[30] *MetaMask - Crypto Wallet for Ethereum and ERC-20 Tokens*. URL: https://metamask.io/ (visited on 03/06/2024).

[31] Nitin Naik, Paul Grace, and Paul Jenkins. "An Attack Tree Based Risk Analysis Method for Investigating Attacks and Facilitating Their Mitigations in Self-Sovereign Identity". In: *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2021, pp. 1–8. DOI: 10.1109/SSCI50451.2021.9659929.

[32] Nitin Naik and Paul Jenkins. "Governing Principles of Self-Sovereign Identity Applied to Blockchain Enabled Privacy Preserving Identity Management Systems". In: *2020 IEEE International Symposium on Systems Engineering (ISSE)*. 2020, pp. 1–6. DOI: 10.1109/ISSE49799.2020.9272212.

[33] Bharti Pralhad Rankhambe and Harmeet Kaur Khanuja. "A Comparative Analysis of Blockchain Platforms – Bitcoin and Ethereum". In: *2019 5th International Conference On Computing, Communication, Control And Automation (ICCUBEA)*. 2019, pp. 1–7. DOI: 10.1109/ICCUBEA47591.2019.9129332.

[34] Sampath S et al. "Decentralized Digital Identity Wallet using Principles of Self- Sovereign Identity Applied to Blockchain". In: *2022 IEEE 7th International Conference on Recent Advances and Innovations in Engineering (ICRAIE)*. Vol. 7. 2022, pp. 337–341. DOI: 10.1109/ICRAIE56454.2022.10054286.

[35] Sheetal Sinha, Kumkum, and Ruchika Bathla. "Implementation of Blockchain in Financial Sector to Improve Scalability". In: *2019 4th International Conference on Information Systems and Computer Networks (ISCON)*. 2019, pp. 144–148. DOI: 10.1109/ISCON47742.2019.9036241.

[36] Mustafa Takaoğlu et al. "The Impact of Self-Sovereign Identities on Cyber-Security". In: Apr. 2023.

[37] *Truffle Suite*. URL: https://archive.trufflesuite.com/ (visited on 03/08/2024).

[38] uport-project. *Ethr-did GitHub Repository*. URL: https://github.com/uport-project/ethr-did (visited on 03/06/2024).

[39] uport-project. *Ethr-did-registry GitHub Repository*. URL: https://github.com/uport-project/ethr-did-registry (visited on 03/06/2024).

[40] *Verifiable Credentials Data Model*. World Wide Web Consortium (W3C). URL: https://www.w3.org/TR/vc-data-model/ (visited on 02/26/2024).

[41] W3C. *Verifiable Credentials Data Model 1.1*. URL: https://w3c.github.io/vc-data-integrity/ (visited on 02/28/2024).

[42] Xuesen Zhang et al. "Research on blockchain consensus algorithm for large-scale high-concurrency power transactions". In: *2022 9th International Forum on Electrical Engineering and Automation (IFEEA)*. 2022, pp. 1221–1225. DOI: 10.1109/IFEEA57288.2022.10037907.