Secure Backup and Recovery of SSI Wallets using Solid Pod Technology

Mohammad Farhad*, Gourab Saha*, Masum Alam Nahid†, Fairuz Rahaman Chowdhury†,
Partha Protim Paul*, Mohammed Raihan Ullah*, Md Sadek Ferdous‡
*Institute of Information and Communication Technology, Shahjalal University of Science and Technology, Sylhet, Bangladesh
†Cryptic Consultancy Limited, London, UK
‡Department of Computer Science & Engineering, BRAC University, Dhaka, Bangladesh

Email: wahedfarhad123@gmail.com, gourabsahag@gmail.com, m.a.nahid008@gmail.com, frcshovon@gmail.com,
partha-iict@sust.edu, raihan-iict@sust.edu, sadek.ferdous@bracu.ac.bd

***Abstract*—A new paradigm for digital identity management called *Self-Sovereign Identity (SSI)* has emerged to offer users more control over their identity data. An important component of SSI is a wallet that stores cryptographic keys and other identity data. Secure backup and recovery methods for data stored in such wallets are a crucial feature for its wide-scale adoption, however, such a feature is lacking or implemented casually in the existing SSI wallets. In this research, we propose a secure backup and storage of SSI wallets using a novel technology called *Solid Pod*. Solid Pod is an emerging technology that enables users to securely store data online. Towards this aim, we present the architecture, based on a threat model and requirement analysis, of the proposed approach. We also discuss its implementation details, outline a number of protocol flows highlighting different use-cases and analyse its security, advantages and limitations.**

***Index Terms*—Self-Sovereign Identity (SSI), SSI Wallet, Social Linked Data, Solid Pod, Blockchain, Hyperledger Aries, Hyperledger Indy.**

I. Introduction

Identity Management is an integrated part of online services which helps both users and service providers to manage the identity data of users. The existing identity management systems (IMSs) suffer from a number of issues: i) they are centralised, ii) privacy-invasive and iii) users have less control over their identity data as these data are mostly controlled by the entity holding those data [1]. To address these issues, a new notion of identity, called self-sovereign identity (SSI), has been proposed [2].

The main motivation of SSI is to empower users in a decentralised way so that users can exert more control over their identity data and access services by sharing their identity data in a privacy-friendly way. SSI has a number of entities, such as issuer, holder and verifier, who interact with other using a number of SSI constructs such as SSI connections, Decentralized Identifiers (DIDs) [3], Verifiable Credentials (VCs) [4] and SSI wallets. Among these constructs, an SSI wallet is a crucial component. Indeed, an SSI wallet is used by each entity to store different cryptographic keys, SSI connections, different DIDs and VCs. That is why the security of an SSI wallet is important. Another important feature of an SSI wallet is how such a wallet is backed up and restored (recovered). This is important in situations where someone loses the device on which the wallet was installed. Understandably, a secure back and restore (restore, these two terms will be used interchangeably) is a crucial factor for a wide-scale adoption of SSI. Unfortunately, this crucial feature is either missing in the existing SSI wallets or the mechanism is not very secure.

A novel Web re-decentralisation technology called Solid (Social Linked Data) has been initiated by the World Wide Web creator Tim Berners Lee [5]. Solid is an open-source platform that provides individuals with better control over the storage of their data. Solid allows users to store their data on a decentralised Personal Online Data (POD) server. Solid Pods enable users to store and manage their data independently, without the need for any centralised authority. Other major advantages of using Solid are its strong security and privacy properties [6]. Hence, Solid Pods can be an excellent solution for the secure storage of SSI wallets.

Indeed, in this article, we present a novel approach for secure backup and restoration of SSI wallets using Solid Pods. The main contributions of this research are:

- The proposal of a secure backup and restoration method for SSI wallets.
- A comprehensive architecture for the proposed system, based on a threat model and requirement analysis, along with a discussion of different implementation aspects of the respective Proof-of-Concept (PoC).
- Detailed protocol flow illustrating different use-cases using the PoC.
- A critical analysis of the security and advantage of the proposed approach.

Structure: The concept of SSI and other related topics are briefly introduced in Section II. We review the existing related works and systems in Section III. Then, in Section IV, the core concept is presented along with a threat model and a requirement analysis. The architecture of the proposed system is discussed in Section V along with the details of the implementation of the developed PoC. We illustrate a number

2836-3795/24/$31.00 ©2024 IEEE
DOI 10.1109/COMPSAC61105.2024.00149

of protocol flow highlighting different use-cases using the PoC in Section VI. We analyse the security of the developed system and discuss its advantages, limitation and future works in Section VII. Finally, we conclude in Section VIII.

II. Background

In this section, we briefly discuss about SSI (Section II-A) and solid pod (Section II-B).

A. Self-Sovereign Identity

Identification and authentication are integral processes for accessing online services in a secure and customised way. To streamline these two processes, users first need to register with a Service Provider (SP), creating a digital identity. Since there are many online services that users need to access, they end up creating many more identities where the majority authentication process requires to remember an identifier (e.g. username) and credential (e.g. a password). Unfortunately, the consequence of this leads to a situation in which the identities of a single user are scattered across multiple SPs which are difficult to manage [1]. As data show that the typical corporate user had to remember 191 passwords [7] or that managing usernames and passwords is now the least-liked web experience [8]. To mitigate this issue, different Identity Management Systems (IMSs), such as SAML (Security Assertion Markup Language) [9]), OpenID [10] and OAuth (Open Authorization) [11], have been proposed. Unfortunately, most of them are centralised in nature and users end up having less control over their own data when using these systems.

Self-sovereign identity (SSI) is a new digital identity paradigm on the Internet whose motivation is to empower users so that they can exert more control over their identity data [2]. SSI has a number of entities: such as issuer, holder and verifier. They interact with each other using a number of SSI constructs such as SSI connections, Decentralized Identifiers (DIDs) [3] and Verifiable Credentials (VCs) [4] and SSI wallets. Each entity needs to establish an SSI connection with another entity before they can interact with others. DIDs for each entity are generated from their respective public keys. Each DID is accompanied by a respective DID Document (DID Docs, in short) which is a JSON object containing the linked cryptographic public keys and different metadata. DIDs of two entities are used to establish the SSI connection between them.

VCs are cryptographically signed claims regarding a user-generated by an issuer. The typical SSI flow is as follows. At first, the user (holder) establishes two different SSI connections with the issuer and the verifier respectively. The user then requests for a VC to the user and the issuer issues the VC via the established SSI connection and then the VC is stored in the SSI wallet of the user. Then, when requested, the user releases the VC to the verifier and the Verifier verifies the VC by verifying its digital signature. In this SSI setting, a blockchain can be used as a verifiable data registry for the storage of DIDs and DID Docs as blockchain offers verifiability, persistence and immutability of data [12].

B. Solid (Social Linked Data)

Solid [6] is a standard that allows individuals to store their data securely in decentralised data warehouses known as *Pods*. Data Pods resemble private, protected web servers, which can be hosted on a web server, on a cloud server or even on a personal server. The main motivation is that it will support a decentralised ecosystem for existing social web apps, where users fully control their data and may authorise third parties to use it for certain reasons, such as social networks [13]. Solid refers to a set of protocols and conventions for building decentralised social applications. Some key components of Solid are discussed next.

- **WebID:** A decentralised identity mechanism that allows users to have a single identifier across the Web [14].
- **Pod Provider:** The pod provider hosts the physical or virtual servers on which user data are stored [14].
- **Resource Storage:** The data consisting of both RDF (Resource Description Framework) [15] data and non-RDF data (e.g., jpeg or pdf files) are stored as JSON-LD (JSON Linked Data). All data are stored as files.
- **Linked Data:** Using the principles of Linked Data (a method for structuring, interconnecting and sharing data on the Web), data are represented in a machine-readable format and relationships between entities are established using URIs [14].
- **Web Access Control:** It defines how resources are protected and accessed on the Web. It allows users to control who has access to their data [16], [17].

Solid provides several key features and benefits:

- Facilitating personal data pods which are secure and private storage spaces for an individual's data. The data of each person is stored in their data pod and they have complete control over it.
- Supporting a decentralised data management model where individuals can store their data in a personal data pod, rather than relying on a centralised third party to store it for them. Ultimately, the degree of centralisation or decentralisation achieved in a Solid implementation depends on the specific choices made by users and developers.
- Strong support for secure and privacy-preserving data sharing, enabling individuals to control their data and choose what data they share and with whom they share it. Open standards and protocols within Solid aim to prevent any single entity from controlling the network or user data.
- Offering interoperability with other web-based technologies and data formats, enabling it to work seamlessly with existing infrastructure and systems.

The Solid Pod architecture is illustrated in Figure 1, where each user possesses a Unique Identity on the network represented by a WebID (URL) linked to their Pod (User's Pod). A user interacts with the Application Software which in turn interfaces with their Pods using their respective WebIDs via the Solid Pod Server. Before a user can access their Pod, the

user must be authenticated by the Pod Server. The user can store different types of data within their Pod whereas access to resources and content delivery from the Pod depends on permissions granted to the application, ensuring secure and controlled data sharing.

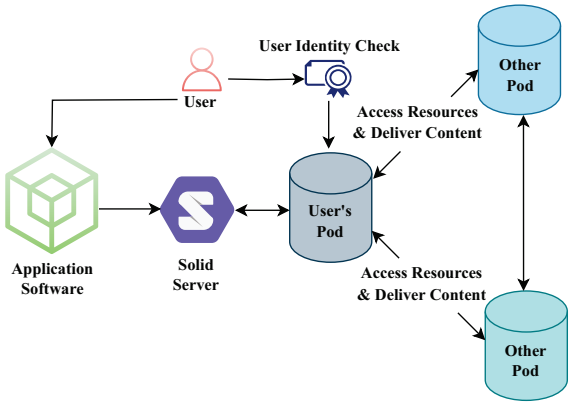

Fig. 1: The Solid Pod Architecture [14].

III. Related Work

In this section, we provide a brief overview of relevant research in the areas of SSI wallets and Solid Pods. To our knowledge, no prior research has explored the possibility of secure backup and restoration of SSI wallets using Solid Pods. However, we have found some relevant researches, which we analyse next.

The authors of [18] have highlighted the weaknesses of traditional digital identity systems, which store user identity details in centralised databases with little control for users over their personal information. It proposes self-sovereign identity (SSI) as a solution for a secure and reliable digital identity system and advises to store personal information in a decentralised manner. In addition, they have suggested the secure backup of their wallets and keeping them in secure digital storage, like a cloud-based provider. The authors of [19] have made a demonstration that illustrates the use of Web-based Verifiable Credentials to grant access to resources stored in a Solid Pod. To maintain the user's privacy, the demonstration uses the BBS + signature scheme (enhanced Boneh, Boyen, and Shachum) [20] for selective disclosure of attributes. In [21], the authors have presented a solution to data over-centralisation, which can lead to tampering and unauthorised sharing of user information. The solution involves combining two technologies, Solid Pods and distributed ledgers, to ensure a complete decentralisation of data with better user controls. The authors in [22] have modelled a digital wallet that uses the secure element of a mobile device to store sensitive information such as identity credentials and cryptographic keys. The wallet can back up and recover secret keys in a distributed and privacy-preserving way. In [23] the authors address the challenges associated with secure backup and recovery processes for private keys, particularly in the context of hardware wallets.

In our research, we have also analysed the backup and restoration of some existing SSI wallets. The Trinsic wallet [24] can backup and restore the wallet in a centralised cloud server, meaning users have no control to dictate the security of these backups. Conversely, the backup generated by the ADEYA wallet [25] and the DIT wallet [26] generate a self-contained wallet file to share and restore without relying on any server. Data wallet [27] has no restoration, just a cloud-based backup option. As we can see, the above-mentioned wallets have various drawbacks, particularly related to security concerns. In this paper, we present a better approach to secure backup and restoration of SSI wallets.

IV. Proposal

In order to address the limitation of existing SSI wallets, we propose a system that integrates Solid Pod technology with an open-source SSI wallet to backup SSI constructs such as SSI connections and VCs. The system demonstrates that it is possible to provide a secure storage solution that reduces the risk of data breaches and unauthorised access to personal data. Before we present the architecture of the proposed system, we identify the underlying security threats (Section IV-A) and formulate a few functional and security requirements (Section IV-B).

A. Threat Modelling

Threat modelling is a structured approach to identify possible security threats to a software system and evaluate the risk associated with these threats. To model threats for this research, we have used the well-known *STRIDE* model [28], which encompasses a variety of security risks as shown below.

- **T1-Spoofing Identity:** This threat with respect to our research involves an attacker creating a false identity as an authorised Pod user or impersonating another person or entity to gain access to sensitive SSI information.
- **T2-Tampering with Data:** It involves an attacker who manipulates, alters, or corrupts the Pod data to achieve a malicious goal.
- **T3-Repudiation:** It refers to the threat of a user denying that they have performed a malicious action.
- **T4-Information Disclosure:** It refers to the situation in which an attacker unintentionally gains access to SSI information stored in a Solid Pod.
- **T5-Denial of Service (DOS):** The goal of a DOS attack is to prevent legitimate users from accessing the functionality of backup and restore, either temporarily or permanently.
- **T6-Elevation of Privilege:** It describes a situation where an attacker gains elevated access rights or permissions, allowing them to perform actions, e.g., receiving the ability to share data from a Solid Pod, that would normally be restricted.

B. Requirement Analysis

Next, we present a set of functional and security requirements. Functional requirements ensure that the system can

function as intended, while security requirements are used to mitigate the identified threats.

1) Functional Requirements (FR)

The functional requirements are listed below:

- **F1.** The system should offer solutions to backup and restore of mentioned SSI constructs to a Solid Pod controlled by the respective user. To do this, we must integrate this backup and restore option into an existing SSI wallet.
- **F2.** The proposed should backup and restore a number of SSI constructs such as Decentralized Identifiers (DIDs), Verifiable Credentials and SSI connections as well as the related keys.
- **F3.** The backup and storage method of SSI wallets should be user-friendly so that it is accessible to users with a variety of technical abilities.

2) Security Requirements (SR)

Next, we present a set of security requirements:

- **S1.** The system must ensure that only authenticated and authorised users can access the data stored within a Pod, thereby mitigating the T1 threat.
- **S2.** The system must ensure that in case of tampering with the SSI constructs, the corrupted data can be easily retrieved from the last backup. Thus, the threat T2 can be mitigated by S2.
- **S3.** The system must utilise a digital signature to protect against the repudiation threat (T3).
- **S4.** The system must apply encryption mechanisms at various levels, such as during data in transit as well as data at rest to mitigate the T4 threat.
- **S5.** Measures should be taken so that users can restore data whenever required, even during a DOS attack (mitigating T5 threat).
- **S6.** Users should only have the minimum degree of access necessary to carry out their job responsibilities, according to the principle of least privileges, to mitigate the T6 threat.

V. Architecture and Implementation

The primary objective of the system is to integrate a Solid Pod-based backup and restoration of SSI constructions within an SSI wallet. Next, we discuss the architecture of the proposed system (Section V-A) and its implementation details (Section V-B).

A. Architecture

The architecture of the system is shown in Figure 2. As per Figure 2, there are four main components of the system: User, Wallet, Blockchain, and Solid Pod Server. The user uses the wallet to store SSI constructs such as SSI connections, VCs and cryptographic keys. The wallet interacts with the blockchain, using a verifiable data registry, to retrieve the DID Doc whenever a new SSI connection is established. According to the user's instructions, the wallet backs up the SSI constructs to the Solid Pod which then can be recovered to another wallet installed on another smart device. In Figure 2, SSI communications are represented by dotted arrows, whereas non-SSI direct communications are represented by solid arrows. It is imperative that all SSI constructs are backed up and recovered with strict security procedures so as to fulfil all the security requirements.

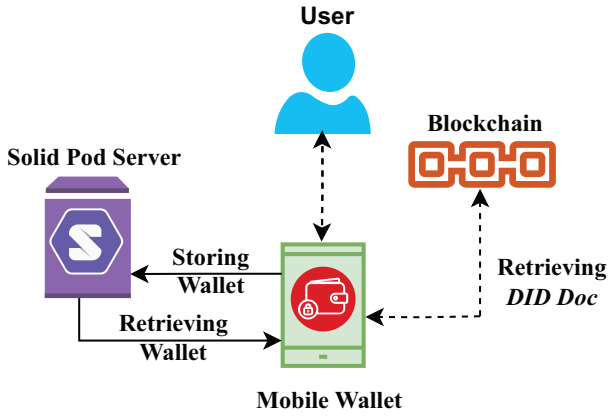

Fig. 2: Architecture of the Proposed System.

B. Implementation

Using a variety of frameworks and technologies, we have developed a Proof of Concept (PoC) using the proposed architecture. Next, we briefly present the implementation details.

For implementation, we needed an open-source SSI wallet so that we could modify the wallet to integrate the required features (interacting with a Solid Pod). We have chosen the Aries-Bifold digital wallet [29] (Bifold in short), which is the only open source wallet available based on the Hyperledger Indy and Aries frameworks. There are a few other Hyperledger Indy and Aries based SSI wallets, such as Trinsic [24], ADEYA [25], DIT [26], however, they are not open-source. The Bifold wallet is written in React-Native, an open-source UI software framework created by Meta Platforms [30].

In addition, we have used an implementation of Hyperledger Aries based SSI framework called, *Hyperledger Aries Cloud Agent - Python (ACA-Py)* [31]. It relies on Hyperledger Indy [32], a purpose-built blockchain for SSI. The Bifold wallet makes use of the ACA-Py based public mediator (Indicio Public Mediator) [33].

For the purposes of storing and backing up the SSI wallet, we have utilised Solid Pod technology [34] in our PoC. In order to manage interactions with the Solid Pod and the wallet, we have incorporated a user interface created by ReactJS [35], a JavaScript library.

VI. Protocol Flow

In this section, we outline the protocol flow, illustrating the interactions between the various parts of the proposed system along with the underlying use-cases utilising the PoC. Towards

this aim, in the following, we present the data model (Section VI-A), algorithm (Section VI-B) and protocol flow (Section VI-C).

A. Data Model

First, we present the mathematical notations in Table I. These notations have been used to create the data model. The notations are self-explanatory and hence, no further explanations are provided.

TABLE I: Cryptographic Notation

Notations	Descriptions
U	User (Playing the role of the holder)
W	SSI Wallet
PS	Pod Server
PP	Pod Provider
$PURL_U$	WebID URL of the Pod for U
$[...]_{https}$	Communication over a https channel
$[...]_k$	Communication over a channel encrypted with key k
h	Hashed user password
$Wallet$	Wallet file consists of SSI constructs
K_U	Secret key for encrypting and decrypting the wallet
$H(M)$	SHA-512 hashing operation of message M
msg	A textual message
$MAC(d, K)$	A message authentication code with data d and key K

Now, we present the data model used in this PoC. The data model is presented in Table II and is discussed next.

TABLE II: Data Model

$podReq \triangleq \langle$type, data$\rangle$
$TYPE \triangleq \langle$registration, backup, restore$\rangle$
$DATA \triangleq \langle$regisData, $PURL_U\rangle$
$regisData \triangleq \langle$userName, name, email, $h\rangle$
$walletData \triangleq \langle\{Wallet\}_{K_U}, MAC(\{Wallet\}_{K_U}, K_U)\rangle$
$resp \triangleq \langle$msg$\rangle$
$loginData \triangleq \langle$userName, $h\rangle$

The proposed system utilises a request and response protocol in which each request has a *type* and *data*. There are three types of requests (registration, backup and restore) as presented in Table II. There are two types of data: $regisData$ and $PURL_U$. $regisData$ and $loginData$ consist of a username and h which denotes the hash of the password, i.e., $h = H(password)$. $PURL_U$ denotes the URL of a specific user Pod. $walletData$ represents a zip file consisting of SSI constructs data (denoted with $Wallet$) encrypted with a user-generated key (K_U) and a Message Authentication Code (MAC) generated with $\{Wallet\}_U$ and the key K_U. There is only one response denoted with $resp$, consists of a message that varies depending on the request type.

B. Algorithm

In order to integrate the backup and recovery features in the Bifold wallet, we have modified its codebase. The pseudocode of the newly added functionalities are presented in Algorithm 1. Next, we present a brief explanation of the pseudocode.

According to Algorithm 1, there are two functions $backupFunc$ and $restoreFunc$ that represent the newly added pseudocode for the backup and restore functionalities in the Bifold wallet. During the backup process, the user needs to submit a password that needs to be repeated for security. These two passwords are fed into $backupFunc$ where these passwords are compared (lines 3-5). If they match, then an AES key (K_U) is generated using the password that is used to encrypt the SSI constructs of the wallet, denoted by $\{Wallet\}_{K_U}$. In addition, a MAC (denoted $MAC(\{Wallet\}_{K_U}, K_U)$) is generated. Then, this function prepares a new zip called $walletData.zip$, which consists of $\{Wallet\}_{K_U}$ and $MAC(\{Wallet\}_{K_U}, K_U)$) and then this zip file is saved in the local storage of the mobile phone. Then a $podReq$ is prepared and sent to the Pod Server (lines 11-12).

Algorithm 1 Algorithm for new functionalities in Bifold

```
1:  ...
2:  function backupFunc(data)
3:      keyPass1 = data.password1;
4:      keyPass2 = data.password2;
5:      if keyPass1 == keyPass2 then
6:          Generate K_U using keyPass1;
7:          Encrypt the wallet using K_U as {Wallet}_{K_U}
8:          Generate MAC using {Wallet}_{K_U} and K_U;
9:          Prepare walletData.zip using {Wallet}_{K_U} and MAC;
10:         Save the walletData.zip file in the local storage.
11:         podReq = ⟨backup, PURL_U⟩;
12:         Send podReq to Pod Server;
13:     else
14:         "Show an error message";
15:     end if
16: end function
17: function restoreFunc(walletData.zip)
18:     keyPass = Inputted password by the user;
19:     Generate K_U with keyPass;
20:     W interacts with U to get walletData.zip;
21:     Unzip walletData.zip;
22:     Retrieve {Wallet}_{K_U} and MAC from walletData;
23:     Verify MAC using {Wallet}_{K_U} and K_U;
24:     Wallet = Decrypted {Wallet}_{K_U} using K_U;
25:     Retrieve all SSI constructs from Wallet;
26:     Restore them in the Bifold wallet;
27: end function
28: ...
```

The $restoreFunc$ function is invoked when the user clicks on a particular button discussed later. When this function is invoked, the user provides a password which is used to generate K_U. The user then selects the $walletData.zip$ file and then the wallet unzips the file. From the unzipped file, $\{Wallet\}_{K_U}$ and $MAC(\{Wallet\}_{K_U}, K_U)$ are retrieved. K_U is used to decrypt $\{Wallet\}_{K_U}$ and verify the MAC. Once verified, SSI constructs are retrieved from the decrypted data which are then restored in the Bifold wallet.

Next, we present Algorithm 2 in which the pseudocode for the Pod server is presented.

According to Algorithm 2, there are four functions: $podRegFunc$, $backupFunc$, $restoreFunc$ and $podLoginFunc$, each representing different functionalities.

The $podRegFunc$ function is used to register a user on the Pod server by passing the required username and its respective hashed password. The $podRegFunc$ function internally uses

Algorithm 2 Algorithm for Pod Server

```
1: ...
2: function podRegFunc(req)
3:     userName = req.regisData.userName;
4:     name = req.regisData.name;
5:     email = req.regisData.email;
6:     h = req.regisData.h;
7:     PURL_U = registerIntoPP(userName,name,email,h);
8:     return PURL_U;
9: end function
10: function backupFunc(req)
11:     Retrieve userName and h by interacting with U;
12:     PURL_U = req.PURL_U;
13:     if podLoginFunc(userName,h) then
14:         Interact with the user to receive walletData.zip file;
15:         Store walletData.zip file to the Pod with PURL_U;
16:         return "Wallet backed up successfully!";
17:     else
18:         return "An error occurred!";
19:     end if
20: end function
21: function restoreFunc(req)
22:     Retrieve userName and h by interacting with U;
23:     URL = req.PURL_U;
24:     if podLoginFunc(userName,h) then
25:         Retrieve walletData.zip file from the Pod using URL;
26:         Allow the user to download walletData.zip file;
27:     else
28:         ''Show an error message";
29:     end if
30: end function
31: function podLoginFunc(userName, h)
32:     h' = getFromPP(userName);
33:     if h == h' then
34:         return TRUE;
35:     else
36:         return FALSE;
37:     end if
38:     end
39: end function
40: ...
```

the $registerIntoPP$ function that returns the user-specific URL of the Pod (denoted with $PURL_U$). On the other hand, the $podLoginFunc$ function is used to authenticate a user to the Pod Server. In this function, the stored hashed password is retrieved from the system using the built-in $getFromPP$ function. Then the retrieved h is compared with the supplied hash. If they match, a *TRUE* value is returned; otherwise, a *FALSE* is returned. The other two functions ($backupFunc$, $restoreFunc$) use the $podLoginFunc$ internally to authenticate a user to the Pod server.

The $backupFunc$ is invoked when the Pod server receives a backup request. At first, $PURL_U$ is retrieved from req. Consequently, the Pod server interacts with the user to receive the username and hashed password and the user is authenticated to the pod server. Then, the $walletData.zip$ file is received from the user. Finally, the $walletData.zip$ file is stored on the Pod server identified by $PURL_U$.

The $restoreFunc$ is invoked during the recovery process. Like before, $PURL_U$ is retrieved from req, the username and hashed password are received from the user and then the user is authenticated to the pod server. Once authenticated, the $walletData.zip$ file is retrieved from the respective user Pod. This zip file is used to prepare $walletResp$ which is then returned.

C. Use-case & Protocol flow

In this section, we present a number of use-cases to illustrate how a user would interact with the system using a number of protocol flows. We consider three different use-cases: Registration, Backup and Recovery. We discuss each of the use-cases in the following.

Registration: Each user must register to the Pod server before using it. In this use-case we present the respective registration process for a user. The protocol for this use-case is illustrated in Figure 3. There are three entities in this use-case: User (U), Pod Server (PS) and the Pod Provider (PP). When a user interacts with the Pod Server to register, it sends a registration page. The screenshot of the registration is shown in Figure 4. We have excluded this interaction from the protocol flow for brevity. Next, we present the subsequent flows.

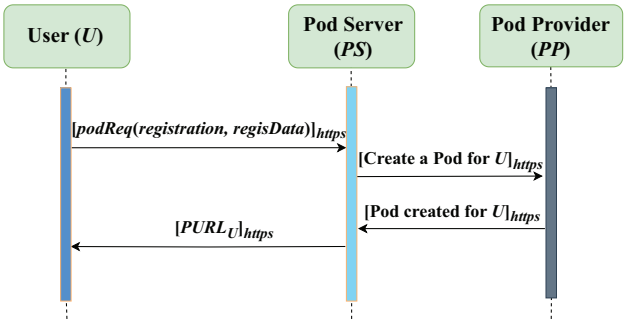

Fig. 3: Registration Protocol Flow.

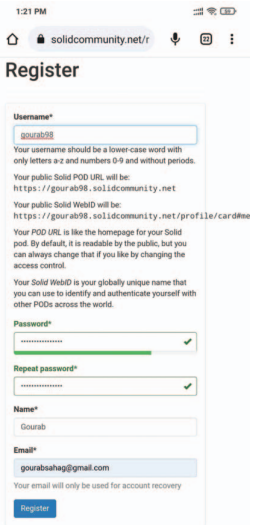

Fig. 4: Pod Registration.

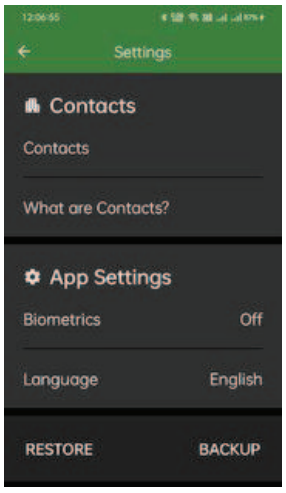

Fig. 5: Backup/Restore.

M1 : The user submits the requested username and password which are then transformed into a $podReq$ consisting of $registration$ as the type and $regisData$ as the data. The $podRegFunc$ of PS is invoked which retrieves the username, hashed password and other data.
M2 : PS interacts with PP to create a Pod for U.
M3 : PP creates a Pod for U and returns its URL $PURL_U$ to PS.
M4 : PS returns $PURL_U$ to the user.

Backup: Next, we discuss the backup use-case along with its respective protocol flow. This use-case is initiated whenever the user would like to create a secure backup of their SSI constructs. We have added two additional options to the Bifold wallet for secure backup and recovery to facilitate this (Figure 5). The backup protocol is illustrated in Figure 6. Before engaging in this use-case, it is assumed that the user has interacted with different SSI entities (e.g. issuers and verifiers) to create SSI connections with them and to receive VCs which are stored in the wallet. With this assumption, we discuss the steps in this protocol flow below.

M1 : The user clicks the *Backup* option in the Bifold wallet.
M2 : The wallet shows a form to submit a password for the backup, as well as a repeat of the password (Figure 7), denoted as $keyPass1$ and $keyPass2$.
M3 : The user submits $keyPass1$ and $keyPass2$.
M4 : At this point, the $backupFunc$ function of Algorithm 1 is invoked and lines 6-12 of Algorithm 1 are executed to send $podReq$ to PS.
M5 : After receiving $podReq$, PS invokes the $backupFunc$ function from Algorithm 2. Since the user's credential is required to continue with the rest of the functionalities, PS sends the login form to the user.
M6 : The user submits their username and password (Figure 8). The submitted password is hashed first and then the username and the hashed password are returned to PS. PS executes line 12 of Algorithm 2 to authenticate the user to the Pod server.
M7 : PS sends an HTML form to U to upload the wallet to be stored in the user Pod (Figure 10).
M8 : U uploads $walletData.zip$ to PS.
M9 : PS requests PP to store $walletData.zip$ in a pod identified by $PURL_U$.
M10 : PP stores those data to to the $PURL_U$ Pod and sends success message back to PS.
M11 : A successful message is returned to W.
M12 : W forwards the successful message to U.

Recovery: Next, we explore the Recovery use-case along with its respective protocol flow. This use-case is initiated whenever the user would like to restore the backed-up wallet from the Solid Pod. The underlying reason could be that the user has lost the mobile where the previous wallet was installed and hence, the backed up SSI constructs need to be restored into a new SSI wallet in a new phone. The protocol for this use-case is illustrated in Figure 9. The protocol steps are presented below.

M1 : The user clicks the *Restore* option in the Bifold wallet.
M2 : The wallet prepares a $podReq$ and sends it to PS.
M3 : After receiving $podReq$, PS invokes the $restoreFunc$ function from Algorithm 2. Since the user's credential is required to continue with the rest of the functionalities, PS sends the login form to the user.
M4 : The user submits their username and password (Figure 8). The submitted password is hashed first and then the username and the hashed password are returned to PS. PS executes line 21 of Algorithm 2 to authenticate the user.
M5 : PS executes lines 21-23 of Algorithm 2 to request the PP to retrieve the $walletData.zip$ file from the Pod identified by $PURL_U$.
M6 : PP returns $walletData.zip$ file to PS.
M7 : PS returns $walletData.zip$ file to the user that is downloaded (Figure 11) and stored to the local storage of the device.
M8 : The user clicks the *Continue* button from the wallet.
M9 : The wallet shows a form to enter a password (denoted as $keyPass$) that is used during the backup use-case.
M10 : The user submits $keyPass$.
M11 : W shows a file explorer for the user to select the downloaded $walletData.zip$ file.
M12 : At this point, W invokes the $restoreFunc$ function of Algorithm 1 and executes lines 17-23 to restore the SSI constructs back to the wallet.
M13 : Finally, a successful message is shown to U.

VII. Discussion

In this section, we examine how our proposed system has satisfied its different requirements (SectionVII-A), discuss its protocol validation (Section VII-B), advantages and limitations (Section VII-C), comparative analysis between Solid Pod and Cloud (Section VII-D),comparative analysis between Bifold and other wallets (Section VII-E) and highlight possible future works (Section VII-F).

A. Analysing Requirements

At first, we explore the functional requirements. The system PoC effectively satisfies all functional requirements, as explained next. The system enables the user to backup and restore a number of SSI constructs such as DIDs, VCs and SSI connections, thereby satisfying *F1* and *F2*. The backup and restore processes are quite easy to follow. The user is guided through each process and they do not need to engage in complex interactions. The complexities of the interactions are effectively hidden from the user. That is why we can say that the system satisfies *F3*.

Next, we explore the security requirements. The Pod Server utilises an authentication method to ensure that only the authenticated user can access their respective Pod data. This satisfies *S1*. Using MAC, we can ensure if the backed up wallet has been corrupted or not. If it is corrupted, the recovery process can be used to easily restore the wallet, thereby

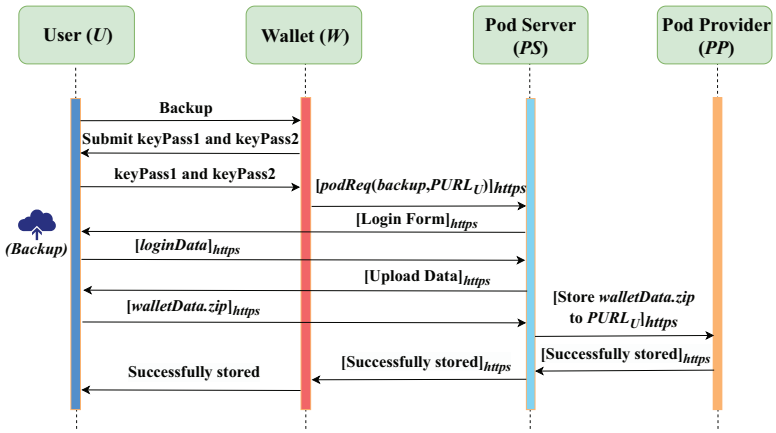

Fig. 6: Backup Protocol Flow.

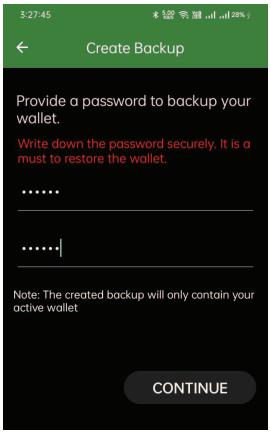

Fig. 7: Preparing Backup.

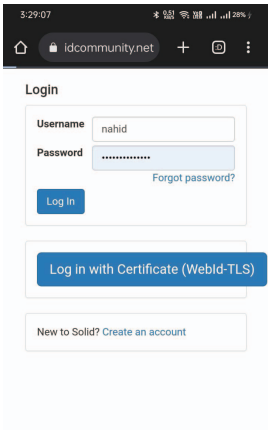

Fig. 8: Pod Server Login.

satisfying *S2*. The system transfers data with the Pod Server over an HTTPS channel and stores the wallet in an encrypted format. Thus, *S4* is satisfied. The user cannot access any other Pods within the Pod Server as the Solid Pod server strictly maintains the principle of least privileges, satisfying thus *S6*. To satisfy *S5*, we need additional mechanisms, for example, using blockchain as a Pod Server. There is no such framework. Finally, the system does not satisfy *S3* since it does not employ any digital signature mechanism. This is because we do not consider the user as the attacker as the attacker would not gain anything by backing up the wallet to the Solid Pod or restoring the wallet from the Solid Pod and then non-repudiating their action.

B. Protocol Validation

To evaluate the security of the system, we formally verify the protocol using a state-of-the-art protocol verification tool, ProVerif [36]. Primarily, we focus on the protocol's secrecy and authentication objectives. To accomplish this, we first formalised the protocol using ProVerif. This process allows ProVerif to evaluate the security properties of the protocol and identify any potential security vulnerabilities. Subsequently, we performed a model-checking procedure, a computational technique that verifies whether the system satisfies the specified security properties. Using this method, we were able to verify whether our protocol met the secrecy and authentication objectives, thereby ensuring the system's overall security.

The secrecy objective focuses on maintaining confidentiality while data are transmitted between entities. It aims to guarantee that only the intended recipient can access the secret information. Conversely, the authentication objective assesses the legitimacy of the two entities involved in the data exchange. Correspondence and injective correspondence assertions ensure the authenticity of the communication between entities. Correspondence assertions establish event relationships within protocols, maintaining proper event sequencing without mandating a one-to-one relationship. Injective correspondence, on the other hand, enforces a strict one-to-one link between events.

In conclusion, by employing the ProVerif tool to assess the security of the system protocol, we can confirm that the system meets the desired secrecy and authentication objectives. This comprehensive evaluation ensures the security and robustness of the system, ultimately protecting it from potential threats.

To illustrate the validation process, we have attached screenshots of the core backup and recovery protocols in Figure 12 and Figure 13, indicating the successful validation of our protocols. The ProVerif scripts for all protocol validation can be accessed from the URL added in the footnote and verified independently. [1]

C. Advantages and Limitations

Our proposed system offers a number of advantages, as discussed below:

- This is the first system to integrate secure backup and restore features within open source SSI wallet by leveraging the Solid Pod technology.

[1] https://drive.google.com/file/d/1-rkPDBOhWokEtXERt3xepU4vxQ7R3Prm/view?usp=sharing

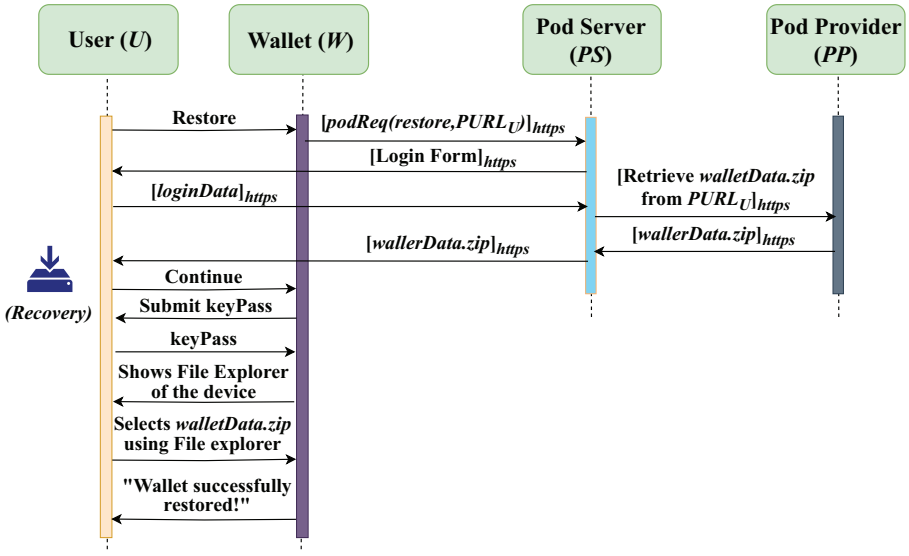

Fig. 9: Recovery Protocol Flow.

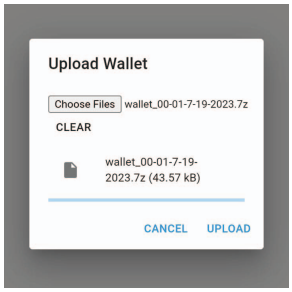

Fig. 10: Wallet Upload.

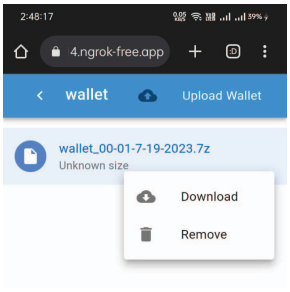

Fig. 11: Download Option.

- Due to the use of Solid Pod technology, the system ensures that an SSI wallet is securely stored considering a number of security requirements.
- The proposed solution also offers better privacy. The access control techniques of Solid Pod technology limit access to resources depending on the roles and permissions of the user, limiting the risk of privacy breaches in the wallet. To further minimise this risk, the wallet is stored in an encrypted format.
- Solid Pods offer a number of use cases and may be used to store a broad variety of data types. Users can host their Pod on different storage servers, thus providing enough flexibility.
- The technology behind Solid Pod is built on open protocols and standards, making it simpler to adopt and modify as required.

However, there are also some limitations to consider, including:

- The usage of a Pod server might create a single point of failure, e.g. amid a DoS attack towards the Pod Server. Therefore, it is important to investigate how to decentralise the approach so that the solution can function even if a single Pod server is not functional.
- Since SSI and Solid Pod are novel technologies, it is crucial to investigate the usability of the proposed system.

D. Comparative Analysis between Solid Pod and Cloud

There are several advantages to using a Solid Pod server in comparison to a cloud-based server for backing up and restoring SSI wallets. We highlight a few of them below.

- **Data Ownership and Control:** Solid Pods provide allow users to exert better control for storing their data, as users can decide where to host their Pods and enforce their own rules for storing such data. On the contrary, when users store data on a cloud server, they are subject to their terms of service and privacy policies. Users have limited control over who can access the data and have little or no knowledge of how their data are used.
- **Privacy and Security:** Solid Pod users can define their own access control rules and decide who can access their data. Open standards and protocols are used for data access, reducing the risk of any vulnerability. Cloud providers generally have access to user data and may be subject to government requests for data access. In addition, cloud servers can be more vulnerable to hacking and data breaches.
- **Interoperability and Flexibility:** Pods can interoperate with different applications and services that adhere to open standards. This allows users to easily move data between different authorised users/domains. Cloud servers often use proprietary formats and protocols, making it difficult to move data to other platforms/domains.
- **Decentralisation:** Solid Pods contribute to a decentralised web where data is not controlled by large corporations. This can lead to greater resilience and less

```
Verification summary:
Query not attacker(loginData[]) is true.
Query not attacker(walletData[]) is true.
Query not attacker(podReq[]) is true.
Query not attacker(successMsg[]) is true.
Query event(endBackup(successMsg_1)) ==> event(beginBackup(backup_1)) is true.
Query event(endUpload(walletData_1)) ==> event(beginUpload(uploadData_1)) is true.
Query inj-event(endUpload(walletData_1)) ==> inj-event(endLogin(loginData_1)) is true.
Query inj-event(endPodRequest(successMsg_1)) ==> inj-event(endSubmitKeys(keyPass[])) is true.
```

Fig. 12: Backup Protocol Validation Result.

```
Verification summary:
Query not attacker(loginData[]) is true.
Query not attacker(walletData[]) is true.
Query not attacker(podReq[]) is true.
Query event(endRestore(successMsg_1)) ==> event(beginRestore(backup)) is true.
Query event(endSubmitKeys(keyPass_1)) ==> event(beginSubmitKeys(submitKeys_1)) is true.
Query event(endPodRequest(continue_1)) ==> event(beginPodRequest(podReq_1)) is true.
Query event(endLogin(loginData_1)) ==> event(beginLogin(loginForm_1)) is true.
Query inj-event(endWalletDataReq(walletData_1)) ==> inj-event(endLogin(loginData_1)) is true.
Query inj-event(endSubmitKeys(keyPass[])) ==> inj-event(endPodRequest(continue_1)) is true.
```

Fig. 13: Recovery Protocol Validation Result.

TABLE III: Comparison among SSI Wallets

	Trinsic [24]	ADEYA [25]	DIT [26]	Data [27]	Base Bifold [29]	Updated Bifold
Backup	●	●	●	●	○	●
Recovery	●	●	○	○	○	●
Storage Option	Cloud & Local	Local	Local	Cloud	○	POD Server
Sec. Option	Seed Phrase	Seed Phrase	?	?	?	Password
Encryption	●	●	?	?	?	●
User Controllability	○	○	○	○	○	●

censorship. On the other hand, cloud services reinforce the centralised model, where a few large companies control large amounts of data.

E. Comparative Analysis between Updated Bifold and other wallets

In this section, we present a comparative analysis between existing SSI wallets that have backup and restoration methods with our work (the updated Bifold) with respect to some of the features relevant to this research. The result of our analysis is presented in Table III. We also include the previous Bitfold (denoted as the *Base Bifold*) to underline the difference with our updated version. We have used the "●" symbol to denote a particular feature is present in the respective wallet, whereas the symbol "○" is used to imply that the feature is missing. The symbol "?" is used to indicate that the respective feature is not explicitly specified.

For analysis, we have used six features: backup, recovery, storage option, security (denoted as sec. in Table III) option, encryption and user-controllability. The first two features are to imply if the wallet has backup and restore options respectively. Even though all wallets (except the base BIfold) have backup options, surprisingly DIT and Data wallets did not have the restore options. Most of the wallets used either local (mobile) storage or cloud storage. Only we have used Solid Pod for wallet recovery and backup.

The security option indicates which security feature is required during the backup and restore process. Trinsic and ADEYA used seed phrases, also known as a recovery phrase, which are a series of typically 12, 18, or 24 randomly generated words. These words are selected from a predefined list of words and are ordered in a specific sequence. In our updated Bifold wallet, we used a user defined password that contains the combination of digit, character, special character which is used for encrypting and decrypting the wallet, acts like a barrier to prevent unauthorised access to the information. It is unclear as well as not specified what other wallets used.

The encryption option implies that the wallet is backed up in encrypted formats. Like before, only Trinsic, ADEYA and our solution back up the wallet in encrypted formats. It is not specified for other wallets.

The user controllability feature emphasises whether the user has any control during the backup and restore process. All other wallets do not provide any flexibility regarding where the backup is kept. In our solution, the user has full control to choose any Solid Pod server, providing much better control than any other wallets.

F. Future Work

There are several potential future works within the scope of the current work.

- Introducing an automatic syncing mechanism of SSI construct would facilitate interesting use-cases where the user would be able to access their SSI constructs from multiple devices. This is currently not possible with any SSI wallet. Automating the synchronisation of SSI wallets would improve the usefulness of the system.
- Investigating how blockchain could be utilised to host Solid Pods would also open up novel research avenues. We would also like to explore this in the future.

VIII. Conclusion

In this research, we have proposed the use of Solid Pod technology for the secure backup and recovery of SSI wallets. The proposed system is based on a threat model and a number of requirements. We have developed a PoC and used it to illustrate several use cases that outline the applicability of the system. Our developed PoC provides a secure and efficient way to store and restore SSI wallets, and it can be integrated with existing SSI frameworks to enhance the security and usefulness of SSI applications. We believe that our work provides a valuable contribution to the field of SSI and paves the way for further research in this area.

References

[1] M. S. Ferdous, "User-controlled identity management systems using mobile devices," 2015.

[2] M. S. Ferdous, F. Chowdhury, and M. O. Alassafi, "In search of self-sovereign identity leveraging blockchain technology," *IEEE Access*, vol. 7, pp. 103 059–103 079, 2019.

[3] "Decentralized Identifiers (DIDs) v1.0," accessed: 2023-03-15. [Online]. Available: https://www.w3.org/TR/did-core/

[4] "Verifiable Credentials Data Model 1.0," accessed: 2023-03-15. [Online]. Available: https://www.w3.org/TR/vc-data-model/

[5] J. Werbrouck, P. Pauwels, J. Beetz, and L. van Berlo, "Towards a decentralised common data environment using linked building data and the solid ecosystem," in *36th CIB W78 2019 Conference*, 2019, pp. 113–123.

[6] "Web inventor tim berners-lee's next project: a platform that gives users control of their data," accessed: August 5, 2023. [Online]. Available: https://www.csail.mit.edu/news/web-inventor-tim-berners-lees-next-project-platform-gives-users-control-their-data

[7] N. Djosic, B. Nokovic, and S. Sharieh, "Machine learning in action: securing iam api by risk authentication decision engine," in *2020 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2020, pp. 1–4.

[8] A. Preukschat and D. Reed, *Self-sovereign identity*. Manning Publications, 2021.

[9] J. Hughes and E. Maler, "Security assertion markup language (saml) v2. 0 technical overview," *OASIS SSTC Working Draft sstc-saml-tech-overview-2.0-draft-08*, vol. 13, 2005.

[10] D. Recordon and D. Reed, "Openid 2.0: a platform for user-centric identity management," in *Proceedings of the second ACM workshop on Digital identity management*, 2006, pp. 11–16.

[11] D. Hardt *et al.*, "The oauth 2.0 authorization framework," 2012.

[12] M. J. M. Chowdhury, M. S. Ferdous, K. Biswas, N. Chowdhury, A. Kayes, M. Alazab, and P. Watters, "A comparative analysis of distributed ledger technology platforms," *IEEE Access*, vol. 7, no. 1, pp. 167 930–167 943, 2019.

[13] E. Mansour, A. V. Sambra, S. Hawke, M. Zereba, S. Capadisli, A. Ghanem, A. Aboulnaga, and T. Berners-Lee, "A demonstration of the solid platform for social web applications," in *Proceedings of the 25th international conference companion on world wide web*, 2016, pp. 223–226.

[14] A. V. Sambra, E. Mansour, S. Hawke, M. Zereba, N. Greco, A. Ghanem, D. Zagidulin, A. Aboulnaga, and T. Berners-Lee, "Solid: a platform for decentralized social applications based on linked data," *MIT CSAIL & Qatar Computing Research Institute, Tech. Rep.*, 2016.

[15] "Rdf (resource description framework)," accessed: August 5, 2023. [Online]. Available: https://www.w3.org/RDF/

[16] "W3c: (webaccesscontrol)," accessed: August 5, 2023. [Online]. Available: https://www.w3.org/wiki/WebAccessControl

[17] M. Van de Wynckel and B. Signer, "A solid-based architecture for decentralised interoperable location data," in *12th International Conference on Indoor Positioning and Indoor Navigation, CEUR Workshop Proceedings*, 2022.

[18] M. Shuaib, N. H. Hassan, S. Usman, S. Alam, S. Bhatia, P. Agarwal, and S. M. Idrees, "Land registry framework based on self-sovereign identity (ssi) for environmental sustainability," *Sustainability*, vol. 14, no. 9, p. 5400, 2022.

[19] C. H.-J. Braun and T. Käfer, "Attribute-based access control on solid pods using privacy-friendly credentials," in *Proceedings of Poster and Demo Track and Workshop Track of the 18th International Conference on Semantic Systems co-located with 18th International Conference on Semantic Systems (SEMANTiCS 2022) Ed.: U. Şimşek*, 2022, p. 5.

[20] " T. Looker, O. Steele, BBS+ Signatures 2020, Draft CG Report, W3C Credentials CG, 2022," accessed: August 5, 2023. [Online]. Available: https://w3c-ccg.github.io/ldp-bbs2020/#the-bbs-signature-suite-2020

[21] M. Ramachandran, N. Chowdhury, A. Third, J. Domingue, K. Quick, and M. Bachler, "Towards complete decentralised verification of data with confidentiality: different ways to connect solid pods and blockchain," in *Companion Proceedings of the Web Conference 2020*, 2020, pp. 645–649.

[22] R. Soltani, U. T. Nguyen, and A. An, "Practical key recovery model for self-sovereign identity based digital wallets," in *2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*. IEEE, 2019, pp. 320–325.

[23] H. Rezaeighaleh and C. C. Zou, "New secure approach to backup cryptocurrency wallets," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.

[24] "Trinsic," accessed: August 5, 2023. [Online]. Available: https://trinsic.id

[25] "Adeya wallet," accessed: August 5, 2023. [Online]. Available: https://blockster.global/

[26] "Dit wallet," accessed: August 5, 2023. [Online]. Available: https://play.google.com/store/apps/details?id=com.dit&hl=en_IN&gl=US

[27] "Data wallet," accessed: August 5, 2023. [Online]. Available: https://igrant.io/datawallet.html

[28] A. Shostack, *Threat modeling: Designing for security*. John Wiley & Sons, 2014.

[29] "Aries mobile agent react native," accessed: August 5, 2023. [Online]. Available: https://github.com/hyperledger/aries-mobile-agent-react-native

[30] "React native," accessed: August 5, 2023. [Online]. Available: https://reactnative.dev/

[31] "Hyperledger aries," accessed: August 5, 2023. [Online]. Available: https://wiki.hyperledger.org/display/ARIES

[32] "Hyperledger indy," accessed: August 5, 2023. [Online]. Available: https://wiki.hyperledger.org/display/INDY

[33] "Indicio public mediator," accessed: August 5, 2023. [Online]. Available: https://indicio-tech.github.io/mediator/

[34] "Solid pod," accessed: August 5, 2023. [Online]. Available: https://solidproject.org/

[35] "Reactjs," accessed: August 5, 2023. [Online]. Available: https://react.dev

[36] B. Blanchet, B. Smyth, V. Cheval, and M. Sylvestre, "Proverif 2.00: automatic cryptographic protocol verifier, user manual and tutorial," *Version from*, pp. 05–16, 2018.