This work has been accepted and presented at the 5th International Congress on Blockchain and Applications 2022. To be included in the proceedings published by Springer under LNCS.

SSI4Web: A Self-sovereign Identity (SSI) Framework for the Web

Md Sadek Ferdous[1,2], Andrei Ionita[1], and Wolfgang Prinz[1]

[1] Fraunhofer Institute for Applied Information Technology, Schloss Birlinghoven, 53757 Sankt Augustin, Germany
md.sadek.ferdous@fit-extern.fraunhofer.de, {andrei.ionita,wolfgang.prinz}@fit.fraunhofer.de
[2] BRAC University, 66 Mohakhali, 1212 Dhaka, Bangladesh

Abstract. The traditional protected web services rely on a user authentication process. The utilisation of an identifier (e.g. username, email address and so on) and credential (e.g. password) still remains the most widely deployed user authentication process, even though such an authentication process is one of the major sources of security breaches. Moreover, in this traditional setting, the management and sharing of user identity information is cumbersome with limited user controls over their identity data. In recent times, SSI has emerged as a new mechanism for managing and exchanging identity information in a more user-centric and privacy-friendly way. There are many explorations of SSI in different application domains, however, its utility for the web mostly remains unexplored. In this work, we present *SSI4Web*, a framework for integrating Self-sovereign Identity (SSI) for providing web services in a secure passwordless manner with much more user control and greater flexibility. We provide its architecture, discuss its implementation details, sketch out its use-case with an analysis of its advantages and limitations.

Keywords: Self-sovereign Identity · SSI · Blockchain · Web · Hyperledger Indy · Hyperledger Aries.

1 Introduction

With the growing popularity of web services, we have experienced a plethora of web services offering a wide range of online services such as social networking, online banking, e-commerce and so on. Most of these services require a user authentication process which traditionally relies on a knowledge based mechanism utilising an identifier (e.g. username email address) and a corresponding credential (e.g. a password) [1]. Unfortunately, such a password based mechanism is regarded as a major source of numerous security breaches and has strong usability issues [2]. There have been numerous efforts to replace such a mechanism, however, none of these efforts have been successful enough to replace passwords yet on a large scale [2].

Introduced with Bitcoin [3], the concept of blockchain has received significant attention from all over the world. A blockchain is an example of a distributed ledger (an ordered data structure) which is shared and maintained using a distributed consensus algorithm by a number of Peer to Peer nodes which could be scattered across different geographical locations of the world [4]. Similar to blockchain, Self-sovereign identity (SSI) is recently emerging as a new paradigm to manage one's digital identity, that the user creates and controls throughout its life-cycle [5, 6]. It is in contrast to the traditional setting of identity management which is mostly controlled by different service providers (SPs) [7]. SSI aims to disrupt this notion by giving more controls to

the users to handle their identity life-cycle. Understandably, SSI has generated a great deal of interest among enthusiasts from industries, academics and the Governments. Therefore, there are a number of researches exploring how SSI could be deployed in different application domains such as healthcare [8], IoT [9] and so on. There are other researches which have explored how SSI can be integrated with OAuth [10], a widely deployed access delegation method and with SAML (Security Assertion Markup Language) [11], a widely used standard for creating and managing identity federations. However, none of the existing works considered how SSI could be utilised for the web in such a way which could eventually replace the traditional method of the password-based authentication. Hence, the utility of SSI for the web mostly remains unexplored.

In this work, we present **SSI4Web**, a novel framework for accessing restricted web services using SSI with the motivation to facilitate a passwordless, secure user authentication mechanism for the web with more privacy controls.

Contributions: The main contributions of the paper are presented below:

- We present SSI4Web, a framework for integrating SSI in order to access restricted web services.
- We explain the motivation behind SSI4Web and discuss its architecture and design choices which are based on a rigorous threat modelling and requirements analysis.
- We also discuss its implementation details and sketch out a use-case with a protocol flow.
- Finally, we analyse how SSI4Web satisfies the formulated requirements as well as its advantages, limitations and future work.

Structure: Section 2 outlines the threats and the corresponding functional, security and privacy requirements for SSI4Web. We present the architecture, implementation details and a use-case protocol flow in Section 3. We analyse how SSI4Web satisfies different formulated requirements in Section 4 along with a highlight of its advantages, limitations and possible future work. Finally, we conclude in Section 5.

2 Threat Modelling & Requirement Analysis

In this section, we present a threat model (Section 2.1) and analyse a number of functional, security and privacy requirements (Section 2.2) for SSI4Web.

2.1 Threat Modelling

To model threats with respect to this work, we have chosen a well established threat model called STRIDE [12], which encapsulates different security threats as presented below.

T1. **Spoofing Identity:** This threat implies that an attacker can access an online service by spoofing someone else's identity.
T2. **Tampering with Data:** An attacker can modify crucial information, e.g. claims in a Verifiable Credential (VC, explained later) for malicious purposes.
T3. **Repudiation:** This threat implies that a corresponding entity can repudiate certain invalid and illegal actions while accessing online services.
T4. **Information Disclosure:** Sensitive data are revealed to an attacker unintentionally.
T5. **Denial of Service (DoS):** The online service can be the target of a DoS attack.
T6. **Elevation of Privilege:** An attacker might use other attack vectors to elevate their access privilege within the online services.

In addition to these, we have considered an additional threat which is crucial for accessing restricted web services securely.

T7. **Replay:** An attacker might capture an request/response and submit it afterwards, thus launching a replay attack.

The privacy threats mostly emerge from the lack of any privacy control by any user. Based on this assumption, the identified threats are as follows.

T8. **Lack of Consent:** A VC is released without the consent of a user.
T9. **Lack of control:** Users have little control over how individual claims are released to an SP.

2.2 Requirement analysis

In this section, we present a set of functional, security and privacy requirements. The functional requirements capture the core functionalities of the system while security and privacy requirements ensure that they mitigate the identified threats.

Functional Requirements (FR): The requirements are presented below.

F1. The system must integrate SSI mechanisms into its online services so that a user can access online services following the SSI steps.
F2. The system must be accompanied by a corresponding SSI wallet so that users can manage their self-sovereign identities using the wallet.
F3. Users should be able to access the services provided by an SP using different devices in a seamless manner.

Security Requirements (SR): Next, we present a set of security requirements to address the identified security threats.

S1. The system must ensure that users can be authenticated using a VC based SSI and utilise the VC to securely access a service. This will mitigate the *T1* threat.
S2. The system must guard against any unauthorised modification of VC data to mitigate the *T2* threat.
S3. The system must utilise digital signature in order to mitigate the *T3* threat.
S4. Any crucial data must be transmitted in encrypted format via networks so as to ensure the confidentiality of the data related to a VC. This can mitigate the *T4*.
S5. The system must take measures against any DoS attack so as to mitigate the *T5* threat.
S6. The system should utilise an access control mechanism (e.g. RBAC [13]) to ensure that an attacker cannot elevate their access privilege and thereby mitigating the *T6* threat.
S7. The system must take protective measures against any replay attack in order to mitigate the *T7* threat.

Privacy Requirements (PR): Privacy requirements are important to mostly mitigate the privacy threats. Only one privacy requirement is identified as presented below:

P1. The system must ensure that every VC is released only after the user has checked and consented. This mitigates threats *T8* and *T9*.

3 SSI4Web Framework

In this section, we summarise the concept of SSI (Section 3.1) and then discuss the architecture of the SSI4Web framework & its implementation details (Section 3.2) and protocol flow (Section 3.3).

3.1 SSI Concept

Plans for the next generation of digital identity concentrated on introducing user autonomy over the digital identity and the possibility of sharing it across any number of services. This gave rise to the notion of Self-sovereign Identity (SSI). There are three major entities in SSI (Figure 1): issuer, holder and verifier.

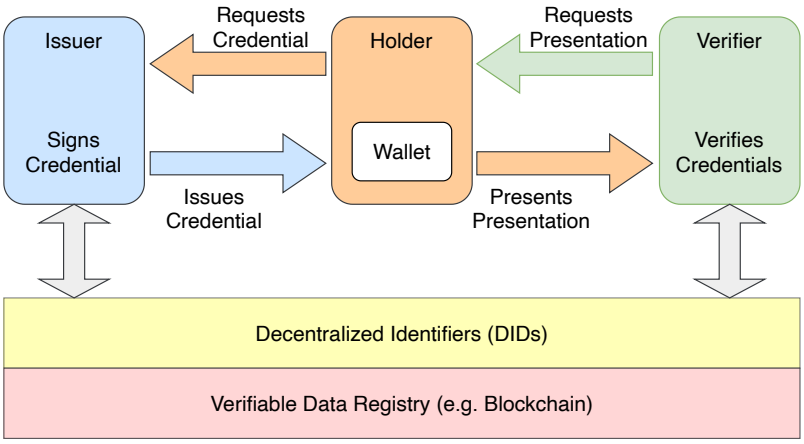


Fig. 1: SSI entities and their relations (add reference).

In realisation of SSI, the W3C community developed *Decentralized Identities* (DIDs) and *Verifiable Credentials* (VCs) [14] as central constructs for SSI. DIDs are user-generated strings linked to an SSI entity's cryptographic public key. DID Documents (DID Docs in short) are JSON objects containing the DID, its linked cryptographic public keys and further metadata. On the other hand, *Verifiable Credentials* (VCs) are cryptographically-signed claims (attribute values) made by an issuer over a subject. Since the VCs are signed by the issuers, they can be verified by any other party (Verifier) that resolves their DID and checks the signature with the linked public key. The holder of the VCs stores them in a digital wallet and is able to present these, either individually or as a combination, to a verifier that requires the information in exchange for an action in the holder's favour (e.g accessing a service). Within this setting, blockchain is used as a verifiable registry for storing DIDs or DID Docs because of its immutability, persistence and decentralisation properties.

3.2 Architecture & Implementation

The core motivation of the SSI4Web Framework (or simply SSI4Web in short) is to facilitate mechanisms for integrating SSI for offering protected web services by a Service Provider (SP). Since protected web services require, among other things, the authentication of the users, one of the principal use-cases within this setting is to use SSI for a passwordless user authentication mechanism for accessing protected web services, which is also the primary focus of this work.

In SSI4Web, a user utilises an SSI wallet to interact with the SP. In contrast to the SSI setting where different entities play different roles of issuers and verifiers, the SP within SSI4Web plays a dual role of an issuer and verifier. Within this setting, the SSI4Web must accommodate the following four crucial SSI functionalities.

– **Establishing the connection:** An SSI connection needs to be established between the wallet of the user and the SP.
– **Providing VC:** The SP can release a VC, when requested, to the user using the previously established connection. This process is analogous to the user registration process in the traditional web setting.
– **VC Storage:** The received VC must be stored securely in the user wallet.
– **Requesting Proof:** The SP requests a proof of the previously released VC from the user using the same established connection. This process is analogous to the authentication process in the traditional web setting.

Figure 2 illustrates the architecture of SSI4Web. The architecture has been designed in such a way that it can facilitate the stated functionalities discussed previously. From Figure 2, SSI4Web has five major entities: SP, user, wallet, mediator and blockchain. The SP is responsible for offering SSI based web services and utilises an SSI agent for managing SSI interactions. The user utilises the wallet to interact, via a mediator, with the SP and to store received VCs. The responsibility of the mediator is to forward messages back and forth between the wallet and SP. In addition, a mediator could potentially store messages destined for the wallet in case the wallet is offline and supply it when the wallet becomes online. A blockchain is used as a verifiable data registry. Both the wallet and the SP Agent interact with the blockchain as part of the SSI protocol flow (discussed later). The solid arrows in Figure 2 represent a non-SSI direct communications between two entities whereas the dotted arrows represent the SSI communications.

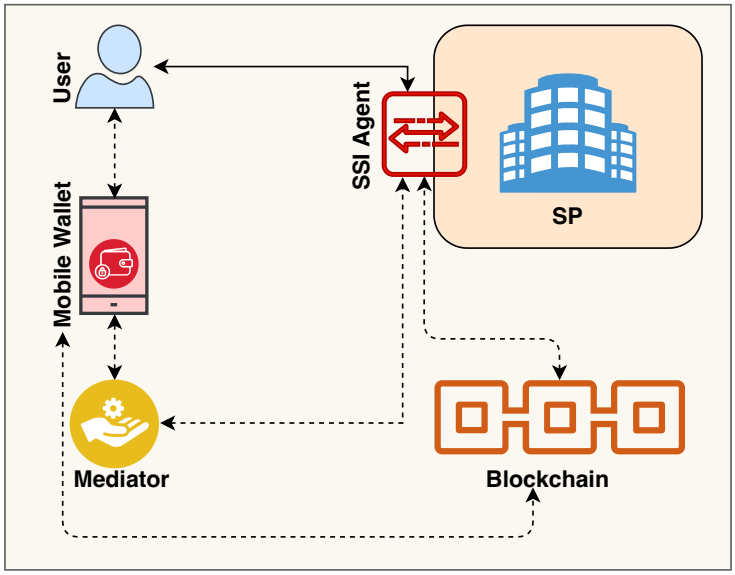


Fig. 2: Architecture of SSI4Web.

We have developed a Proof of Concept (PoC) of our proposal using a number of tools and frameworks. The core tools utilised to develop the PoC are Hyperledger Aries [15] and Hyperledger Indy [16]. Hyperledger Aries provides a set of libraries to develop SSI applications which are available in different programming languages. For our PoC, we have utilised the Hyperledger Aries Cloud

Agent Python (ACA-Py) library [17]. The SSI agent has been developed using NodeJS [18] which depends on the ACA-Py library to serve different SSI functionalities. As the Mobile Wallet we have utilised an open source Aries-compliant mobile App facilitated by the Aries Mobile Agent React Native project [19]. It relies on the Indicio Public Mediator [20], a public mediator based on ACA-Py. The web services have also been developed with NodeJS. Hyperledger Indy is the blockchain platform that is being increasingly used in SSI applications. In our PoC we have utilised an Indy testbed called *BCovrin Dev* (`http://dev.greenlight.bcovrin.vonx.io/`).

3.3 Use-case & Protocol Flow

Next, we present a use-case with protocol flows to show how SSI4Web could be utilised for accessing restricted web services. However, at first, we introduce mathematical notations and data model in Table 1 and Table 2.

Table 1: Cryptographic Notations

Notations	Description
SP	Service provider (Playing the role of an issuer & verifier)
U	User (Playing the role of a holder)
M	Mediator
K_{SP}^{U}	Public key of SP for U
$K_{SP}^{-1\|U}$	Private key of SP to be used for U
K_{U}^{SP}	Public key of U for SP
$K_{U}^{-1\|SP}$	Private key of U to be used for SP
DID_{SP}^{U}	Decentralised identifier of SP for U
DID_{U}^{SP}	Decentralised identifier of U for SP
VC_{SP}^{U}	A verifiable credential issued by SP to U
N_i	A fresh nonce
$\{\}_K$	Encryption operation using a public key K
$\{\}_{K^{-1}}$	Signature using a private key K^{-1}
$H(M)$	SHA-512 hashing operation of message M
$[...]_K$	Communication over an channel encrypted with key K
[...]	Communication over an unencrypted channel
[[...]]	Optional data

Data Model: The developed PoC is designed as a request and response system (denoted with *req* and *resp* respectively in Table 2). A *req* can be of four types: *serviceReq*, *inviteReq*, *credReq* and *proofReq* which represent a service request, SSI invitation request, credential request and proof request respectively, where the last three requests correspond to different SSI functionalities. Similarly, there are four responses which correspond to the four requests.

The *serviceReq* contains the URL of the requested service and optionally an invite cookie (denoted with *inviteCookie*) and a service cookie (denoted with *serviceCookie*). The functionalities of these cookies will be explained when we present the protocol flow. On the other hand, an SSI invite request (*inviteReq*) consists of the URL of the SP SSI agent (url_{SP}) and a nonce. There are two corresponding responses, *inviteResp1* and *inviteResp2*. The former response contains the previously

Table 2: Data Model

$req \triangleq \langle serviceReq, inviteReq, credReq, proofReq \rangle$
$resp \triangleq \langle serviceResp, inviteResp[1-2], credResp, proofResp \rangle$
$serviceReq \triangleq \langle url_{s_i}, [[inviteCookie, serviceCookie]] \rangle$
$inviteReq \triangleq \langle url_{sp}, N_i \rangle$
$inviteResp1 \triangleq \langle N_i, DID_U^{SP} \rangle$
$inviteResp2 \triangleq \langle N_i, DID_{SP}^{U} \rangle$
$credReq \triangleq \langle$"Requesting Credential"$\rangle$
$attrReq \triangleq \langle a_1, a_2, ..., a_n \rangle$
$attrResp \triangleq \langle (a_1, av_1), (a_2, av_2), ..., (a_n, av_n) \rangle$
$credResp \triangleq \langle VC_{SP}^{U} \rangle$
$VC_{SP}^{U} \triangleq \langle \{(a_1, av_1), (a_2, av_2), ..., (a_n, av_n)\}_{K_{SP}^{-1\|U}} \rangle$
$initiateProofReq \triangleq \langle$"Initiating Proof Request"$\rangle$
$proofReq \triangleq \langle a_1, a_2, ..., a_n \rangle$
$proofResp \triangleq \langle VC_{SP}^{U} \rangle$

used nonce and a DID of the user to be used with SP, denoted with DID_U^{SP}. Similarly, the later response contains the same nonce and a DID of SP to be used with U, denoted with DID_{SP}^{U}.

Next, a credential request is denoted with *credReq* which just contains a string as shown in Table 2. The *attrReq* contains the list of attribute names and *attrResp* contains the attribute name-value pairs. A credential response (*credResp*) corresponds to a *credReq* and consists of a VC provided by SP to U and hence denoted with VC_{SP}^{U}. Such a VC is simply represented with a digitally signed set of attribute name-value pairs and modelled as shown in Table 2. Finally, an invitation proof request is denoted with *inviteProofReq* which just contains a string as shown in Table 2. A proof request (*proofReq*) contains the list of attribute names whose values must be present in the released VC (VP) and the corresponding proof response (*proofResp*) contains the released VC.

Algorithm 1: Algorithm Snippet for `handleInviteResp` function

```
1  Start
2      ...
3      function handleInviteResp(inviteResp1)
4          DID_U^SP := inviteResp1.DID_U^SP;
5          Retrieve the corresponding DID Doc using DID_U^SP from the blockchain;
6          Retrieve K_U^SP from the DID Doc;
7          Store K_U^SP and DID_U^SP in the agent Wallet;
8          Generate K_SP^U and K_SP^{-1|U}, create a DID Doc using K_SP^U;
9          Push the newly created DID Doc in the blockchain;
10         Prepare inviteResp2 using K_SP^U;
11         return inviteResp2;
12     ...
```

Algorithm: Now, we present the algorithm snippets of three important functions for the SP SSI Agent, namely handling an invitation response (`handleInvi- teResp` function in Algorithm 1), generating a VC (`handleInviteResp` function in Algorithm 2) and handling a proof response (`handleProof` function in Algorithm 2). The `handleInviteResp` function is responsible for han-

dling *inviteResp1*. Within the function, the DID (e.g. DID_U^{SP}) is retrieved and then the DID is used to retrieve the corresponding DID Doc from the blockchain (line 4 and 5 in Algorithm 1). Then, K_U^{SP} is retrieved from the DID doc. In response, the agent at first generates a new key pair (K_{SP}^U and $K_{SP}^{-1|U}$) and then generates a new DID Doc (line 8 in Algorithm 1). This DID Doc is pushed to the blockchain (line 9). Finally, *inviteResp2* is created using K_{SP}^U which is then returned back (line 10-11).

The `generateVC` function is used to generate and return a new VC by using the retrieved attributes from *attrResp* (line 4 to 6 in Algorithm 2). Finally, the `handleProof` function is used to handle a proof response. For this, at first the user presented VC (VC'^U_{SP}) is retrieved (line 8) which is then used to compare the previously supplied VC to the user (VC_{SP}^U). If they match, a `TRUE` response is returned, otherwise a `FALSE` false is returned (line 10-14).

Algorithm 2: Algorithm snippet for `credResp` & `handleProof`

```
1  Start
2  ...
3  function generateVC(attrResp)
4      atValSet := Retrieve attributes and their values from attrResp;
5      VC^U_SP := createVC(atValSet);
6      return VC^U_SP;
7  function handleProof(proofResp)
8      VC'^U_SP := proofResp.VC'^U_SP;
9      Retrieve VC^U_SP from the wallet for U;
10     if VC'^U_SP == VC^U_SP then
11         return TRUE;
12     else
13         return FALSE;
14     end
15 ...
```

Protocol flow: Now, we present a combined protocol flow for showcasing how SSI4Web facilitates the three SSI functionalities (creating a new connection, releasing a credential and releasing a proof). The flow is sketched in Table 3 and discussed next.

i The user submits a request to an SP to access one of its services at url_{s_i}.
ii The SP returns a web-page having different options for SSI (e.g. creating invitations, release credentials and request a proof).
ii The user choose the "New Invitation" option.
iv The SP (its SSI agent) generates a new invitation which is encoded as a QR code and is displayed to the user.
v The user utilises their wallet to scan the QR code. The wallet then generates a new key pair (K_U^{SP} and $K_U^{-1|SP}$) and creates a new DID (DID_U^{SP}) and DID Doc utilising the generated key pair. The DID Doc is pushed to the blockchain. The wallet then prepares *inviteResp1* using DID_U^{SP} and returns it back to the SP via the mediator M, steps M5 and M6 in Table 3.
vi The SP utilises its `handleInviteResp` function (Algorithm 1) to handle *inviteResp1* and prepare *inviteResp2* which is then returned to the user wallet via mediator M (steps 7 and 8 in Table 3). The user wallet retrieves required data from *inviteResp2* and combines it with data from DID_U^{SP} to save as a new connection to the wallet (*Test* in Figure 3a).

Table 3: Web-SSI Protocol

$M1$	$U \rightarrow SP : [N_1, serviceReq(url_{s_1})]_{HTTPS}$
$M2$	$SP \rightarrow U : [N_2, \text{SSI Option Page}]_{HTTPS}$
$M3$	$U \rightarrow SP : [N_2, \text{New Invitation Option}]_{HTTPS}$
$M4$	$SP \rightarrow U : [N_3, inviteReq]_{HTTPS}$
$M5$	$U \rightarrow M : [SP, N_3, inviteResp1]$
$M6$	$M \rightarrow SP : [N_3, inviteResp1]$
$M7$	$SP \rightarrow M : [U, N_3, inviteResp2]$
$M8$	$M \rightarrow U : [N_3, inviteResp2]$
$M9$	$SP \rightarrow U : [N_4, \text{Credential Option Page}, inviteCookie]_{HTTPS}$
$M10$	$U \rightarrow SP : [N_4, \text{Credential Reuqest}]_{HTTPS}$
$M11$	$SP \rightarrow U : [N_5, attrReq]_{HTTPS}$
$M12$	$U \rightarrow SP : [N_5, attrResp]_{HTTPS}$
$M13$	$SP \rightarrow M : [U, \{N_5, credResp\}_{K_U^{SP}}]$
$M14$	$M \rightarrow U : [\{N_5, credResp\}_{K_U^{SP}}]$
$M15$	$U \rightarrow SP : [N_6, \text{Initiate Proof Request}]_{HTTPS}$
$M16$	$SP \rightarrow M : [U, \{N_6, proofReq\}_{K_U^{SP}}]$
$M17$	$M \rightarrow U : [\{N_6, proofReq\}_{K_U^{SP}}]$
$M18$	$U \rightarrow M : [SP, \{N_6, proofResponse\}_{K_{SP}^{U}}]$
$M19$	$M \rightarrow SP : [\{N_6, proofResponse\}_{K_{SP}^{U}}]$
$M20$	$SP \rightarrow U : [N_1, url_{s_1}, serviceCookie]_{HTTPS}$

vii Once the connection is established, the SP shows a page to the user to request for credentials along with an *"inviteCookie"* which holds metadata about the new invitation.

viii The user submits a request to release a credential.

ix The SP returns a page where the user can submit different values for the required attributes, steps M11 and M12 in Table 3.

x The SP prepares a VC (VC_{SP}^{U}) and a *credResp* with the VC which is then returned back to the user via M, steps M13 and M14 in Table 3. After receiving, the wallet validates the signature of the VC using the public key of the SP for this particular connection as stored in the wallet. If it is successful, the VC is stored in the wallet (Figure 3b).

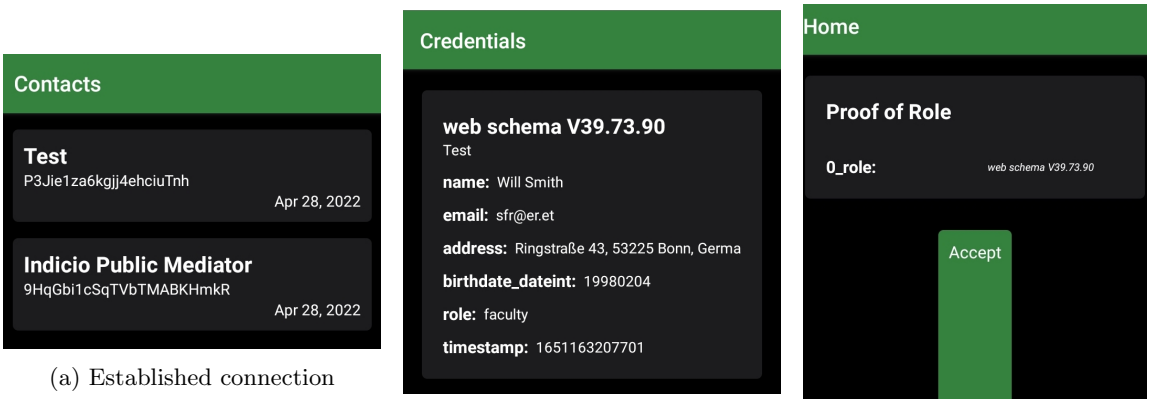
xi Next, the user is forwarded to the SSI option page again where the user chooses a proof request option.

xii The SP generates a *proofReq* which is then sent to the user wallet via M, steps 16 and 17 in Table 3.

xiii The wallet parses the requested attributes and any condition from the *proofReq* and matches with the stored credentials in the wallet. The matched VC is shown to the user for their approval (Figure 3c). Once the user approves, the wallet prepares a *proofResponse* using the approved VC which is then returned back to the SP, steps 18 and 19 in Table 3.

xiv The SP validates the VC as before and once validated, the user is forwarded to the requested url (url_{s_i}) with a session cookie (*"serviceCookie"*). The user can then can access the service as long as they are not logged out.

The information from the connection metadata in the wallet can be utilised to access SP services from any browser of any devices using SSI4Web. For example, if a user would like to access the services from the SP from a different device with the previously established connection, the user can do so using a secure two-factor method. At first, the user needs to collect the connection name from



(a) Established connection (b) VC Attributes (c) Proof Request

Fig. 3: Establishing a connection and receiving a VC

the wallet (*Test* in Figure 3a) and click the *'Active Connection"* from the SSI Option Page. Then, the user will have the option to search for the connection using the connection name. When an active connection is found, an one time password will be sent to the previously saved email address (supplied while creating a new connection). Once the code from the email address is supplied, the user can utilise the previous connection in the browser of the new device and the similar flow can be used to access the requested service.

4 Discussion

In this section, we examine how our framework has satisfied its different requirements (Section 4.1), discuss its advantages and limitations (Section 4.2) and highlight possible future works (Section 4.3).

4.1 Analysing Requirements

Functional Requirements: The SSI4Web implementation effectively satisfies all functional requirements. For example, SSI4Web requires that the user can access restricted services by following the SSI steps as outlined in the protocol flow, thereby satisfying *F1*. The user needs to utilise the SSI Wallet to access any SP service and thus satisfying *F2*. The implementation allows a user to access restricted services from multiple devices in a seamless manner which satisfies *F3*.

Security Requirements: It is evident from the discussion of the use-case that SSI4Web satisfies *S1*. *S2* and *S3* are satisfied as well since each VC is digitally signed by the issuer (SP in our case). As evident from the protocol flow, every interactions with the SP and all SSI interactions between the user and SP where sensitive data are transmitted take place either via HTTPS or encrypted channels, thereby satisfying *S4*. Finally, we have utilised nonces extensively in all interactions to satisfy *S7*. Regarding *S5*, every single web service is prone to DoS attacks and SSI4Web is not an exception in this regard. Additional steps are required to mitigate these threats which is beyond the scope of the presented work. Similarly, satisfying *S6* would require to deploy an access control mechanism. In the current implementation of SSI4Web our goal was to showcase if SSI could be utilised for simple web authentication. That is why S6 has not been considered in the presented work.

Privacy Requirements: The wallet utilised in the SSI4Web enables a user to check what attributes are released to the SP (Role attribute in Figure 3c). Only after the user's consent, the VC is released to the user. This satisfies *P1*.

4.2 Advantages & Limitations

The advantages of SSI4Web are summarised below:

- SSI4Web provides a fully passwordless way of accessing restricted online services. This is also advantageous for the service providers as no additional security measures are needed to ensure password security at their ends.
- Users can access such services from multiple devices in a seamless manner without compromising their security. This is much better than accessing services using passwords across multiple devices in different browsers.
- Users have the ultimate control to determine how they want to release their credentials to whom and only with their explicit consent.

The current implementation of SSI4Web has some limitations:

- The wallet plays a central role for the user in any SSI setting. As all the VCs and connections are stored in the wallet, a secure backup mechanism is crucial. The wallet used in our PoC does not have any backup functionality.
- Using SSI4Web to access restricted websites presents a novel way to access any website. There are multiple new steps involved in comparison to the traditional way of accessing restricted websites and it represents a new learning curve for any user. The wide-scale adoption of such SSI based mechanism will depend on the proper training of the user and their willingness to adopt this method.

4.3 Future Work

In future we would like to explore the following:

- Developing a secure backup and synchronisation mechanism for the SSI wallet so as to provide a seamless user experience for any user across multiple devices.
- Studying the usability of proposed approach to understand how difficult it is for the user to adopt this approach.
- Utilising SSI4Web to adopt in other forms of identity management models (e.g. Federated Identity Management [21]). In this current version, we have only explored the SILO model [21]. This will showcase that such a model can be utilised in advanced use-cases as well.

5 Conclusion

In this work, we have presented SSI4Web, a Self-sovereign Identity (SSI) based framework for accessing restricted web services. The main objective of the framework is to facilitate a passwordless, secure user authentication mechanism with better user control over their identity data. We have presented its architecture which is based on a threat model and requirement analysis, discussed its implementation details and highlighted a use-case to show its applicability. With these contributions, we strongly believe that SSI4Web will usher a new domain of research in this respective domain.

References

1. C. Katsini, M. Belk, C. Fidas, N. Avouris, and G. Samaras, “Security and usability in knowledge-based user authentication: A review,” in Proceedings of the 20th Pan-Hellenic conference on informatics, 2016, pp. 1–6.

2. J. Bonneau, C. Herley, P. C. Van Oorschot, and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in IEEE Symposium on Security and Privacy, 2012, pp. 553–567.
3. S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Manubot, Tech. Rep., 2019.
4. 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.
5. A. Mühle, A. Grüner, T. Gayvoronskaya, and C. Meinel, "A survey on essential components of a self-sovereign identity," Computer Science Review, vol. 30, pp. 80–86, 2018.
6. 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.
7. M. S. Ferdous, "User-controlled identity management systems using mobile devices," 2015, PhD. Thesis. University of Glasgow.
8. M. Shuaib, S. Alam, M. S. Alam, and M. S. Nasir, "Self-sovereign identity for healthcare using blockchain," Materials Today: Proceedings, 2021.
9. N. Kulabukhova, A. Ivashchenko, I. Tipikin, and I. Minin, "Self-sovereign identity for iot devices," in International Conference on Computational Science and Its Applications. Springer, 2019, pp. 472–484.
10. S. Hong and H. Kim, "Vaultpoint: A blockchain-based ssi model that complies with oauth 2.0," Electronics, vol. 9, no. 8, p. 1231, 2020.
11. H. Yildiz, C. Ritter, L. T. Nguyen, B. Frech, M. M. Martinez, and A. Küpper, "Connecting self-sovereign identity with federated and user-centric identities via saml integration," in 2021 IEEE Symposium on Computers and Communications (ISCC). IEEE, 2021, pp. 1–7.
12. A. Shostack, Threat modeling: Designing for security. John Wiley & Sons, 2014.
13. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," Computer, vol. 29, no. 2, pp. 38–47, 1996.
14. "Verifiable Credentials Data Model 1.0," Accessed: 2022-04-27. [Online]. Available: https://www.w3.org/TR/vc-data-model/
15. "Hyperledger Aries". Accessed: 2021-11-10. [Online]. Available: https://www.hyperledger.org/use/hyperledger-aries
16. "Hyperledger Indy". Accessed: 2021-11-10. [Online]. Available: https://www.hyperledger.org/use/hyperledger-indy
17. "Hyperledger Aries Cloud Agent - Python ". Accessed: 2021-12-05. [Online]. Available: https://github.com/hyperledger/aries-cloudagent-python
18. "Node.js". Accessed: 2021-11-10. [Online]. Available: https://nodejs.org/en/
19. "Aries Mobile Agent React Native ". Accessed: 2021-11-01. [Online]. Available: https://github.com/hyperledger/aries-mobile-agent-react-native
20. "Indicio Public Mediator". Accessed: 2021-11-01. [Online]. Available: https://indicio-tech.github.io/mediator/
21. A. Josang, M. AlZomai, and S. Suriadi, "Usability and privacy in identity management architectures," in ACSW Frontiers 2007, 2007, pp. 143–152.