# Escaping from Identity Providers: Protecting Privacy with Verifiable Credentials in Community Solid Server

Ben Macdonald[1], Ross Horne[1] and Biagio Boi[2]

[1]Computer and Information Sciences, University of Strathclyde, Glasgow, United Kingdom
[2] Department of Computer Science, University of Salerno, Italy

**Abstract**

This paper investigates how Verifiable Credentials can be incorporated into authentication and authorisation protocols used in Solid as a solution which can enhance user privacy. A VC-based authentication protocol is proven secure as an alternative to Solid-OIDC. The protocol used in this system is then adapted and integrated into the open-source Community Solid Server.

## 1. Introduction

Solid [1] is a web decentralisation project that aims to give users control over the storage and use of their data by organisations and applications. To achieve this, the Solid architecture provides users with their own Personal Online Data Stores (pods) where they can store their data and determine who has access to it, and to what extent. Instead of organisations storing and processing their users' data in one place, such as databases on their own servers, they would improve transparency by interacting with each individual's pod. As Solid involves managing personal data , protecting privacy becomes paramount.

Solid currently recommends the Solid-OIDC [2] protocol to authenticate users before allowing them to access resources on a pod. This protocol is based on the OAuth 2.1 framework where a person can access a service by logging onto their account with a third-party identity provider such as Google, Microsoft or Facebook. The basic flow, as illustrated in the Solid-OIDC Primer [3], consists of two key interactions: the Authorization Code grant (with PKCE) flow followed by the Request flow. In the Authorization Code grant flow, the user's browser interacts with their chosen *OpenID Provider*, logging in and authenticating themselves and also the app the user would like to use before the app can acquire an ID token. This token is then used by the app to initiate the Request flow, where the token is presented to the Authorization Server (AS) and used to verify the user's identity. An access token is then generated, which can be presented to the resource server in order to access the specified resource.

Recent work [4, 5] has carried out an analysis of the Solid-OIDC protocol and identified several security and privacy issues associated with it. A privacy-related concern with Solid-OIDC is the role of third-party Identity Providers (IDP) in the process, as they are required to be contacted whenever a user grants access to their pod's resources. The IDP is able to collect and store information concerning the applications being used, and over time may potentially use

*SoSy'24: Solid Symposium 2024, May 2–3, 2024, Leuven, Belgium*
benianmacdonald@gmail.com (B. Macdonald)

this to track user behaviour and create profiles of users, which could be monetised via targeted advertising for example [6]. Formally, this is a violation of the privacy property unlinkability [7]. In order to address this privacy challenge in Solid, this paper investigates an alternative to Solid-OIDC which incorporates Verifiable Credentials in a new flow which does not require a third-party IDP to be contected during an authentication session. Through examining existing VC-based flows and architecture, a protocol is presented and then integrated into the open-source Community Solid Server[1] as a working demonstration. The protocols used to present VCs to an authorisation server in Solid in order to obtain a resource are presented in Section 3. Authentication properties of the protocol are formally verified in ProVerif.[2]

## 2. Background on Verifiable Credentials

This section briefly introduces Verifiable Credentials and highlights their suitability for use in authentication protocols. A Verifiable Credential [8] is a knowledge graph issued by a trusted third party, typically used to represent the credentials of its subject in a digitally verifiable manner. Credentials are a collection of one or more claims about a subject. Common examples would include info in passports, driving licenses or university degrees, which are used to demonstrate that an entity is who they claim to be and/or possess the ability to meet certain requirements. As they are useful for proving a subject's identity, VCs can be effectively incorporated into authentication systems.

The VC ecosystem normally consists of three main entities: an Issuer, a Holder and a Verifier, as illustrated on the W3C Verifiable Credentials specifications [8]. The **Issuer** is a trusted authority that can create and issue VCs to a Holder should they meet the required criteria. The VC will contain cryptographic proof of which entity issued it. The **Holder** is the entity that possesses a VC and presents it to a verifier to demonstrate their credentials before they can access their services. This is usually done in the form of a Verifiable Presentation (VP), where information from a VC is encoded into a tamper-evident presentation that allows cryptographic verification along with additional security measures to protect against replay attacks. The **Verifier** checks the presented VC (or VP) to ensure that it is valid before the holder can be authenticated. A simple example of a Verifiable Credential from the W3C specifications [8] is provided in Appendix 1, where a university-issued VC can be used to qualify for an alumni discount. Fields are included to specify the issuer (university), holder (student), issuance date and the information asserted about the holder, along with the cryptographic proof.

A VC-based protocol can be thought of as having two distinct stages: issuance and provenance. In the first stage, the holder communicates with the issuer in order to acquire a VC. At no point does the issuer need to be informed of how and when the VC will be used. In the second phase, the holder communicates with the verifier, presenting their VC (in the form of a VP) to gain access to their service. Unlike the Identity Provider in the Solid-OIDC protocol, the issuer has no knowledge of the services being accessed by the holder. It can therefore be said that a VC-based approach to authentication protects the unlinkability property.

[1] Community Solid Server: https://github.com/CommunitySolidServer/CommunitySolidServer

[2] Implementations and ProVerif code are made available in a repository: https://github.com/ben3101/CommunitySolidServer and https://github.com/ben3101/CommunitySolidServer/tree/main/FormalVer

```
{
  "@context": [
  "https://www.w3.org/2018/credentials/v1",
  "https://www.w3.org/2018/credentials/examples/v1"
],
"id": "http://example.edu/credentials/1872",
"type": ["VerifiableCredential", "AlumniCredential"],
"issuer": "https://example.edu/issuers/565049",
"issuanceDate": "2010-01-01T19:23:24Z",
"credentialSubject": {
  "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
  "alumniOf": {
    "id": "did:example:c276e12ec21ebfeb1f712ebc6f1",
    "name": [{
      "value": "Example University",
      "lang": "en"
    }]
  }
},
"proof": {
  "type": "RsaSignature2018",
  "created": "2017-06-18T21:19:10Z",
  "proofPurpose": "assertionMethod",
  "verificationMethod": "https://example.edu/issuers/565049#key-1",
  "jws": "eyJhbGciOiJSUzI1NiIsImI2NCI6ZmFsc2UsImNyaX [snip]"
}
}
```

**Figure 1:** Example of a verifiable credential in JSON-LD format.

## 3. Authentication Protocols using Verifiable Credentials

This section introduces two protocols based on Verifiable Credentials. The first is directly adapted from recent Self-Sovereign Identity (SSI) protocols [9, 7] to Solid and assumes that the app is fully controlled and trusted by the user to act to pose as the user. The second is better suited to Solid's decentralized architecture where an app is not fully controlled by the user, since it decouples the app from the user during what we call the *VC-based Request flow*.

### 3.1. A VC-based Protocol for direct resource requests by the user

We adapt here a related SSI protocol designed originally for Hyperledger Aries to Solid [9, 7]. The protocol provides a strong proof-of-concept for VC-based flows as a means of authenticating users while clearly decoupling an Identity Provider from actions of the app.

Our contribution is to modify the Community Solid Server (CSS) to implement a variant of the Aries Present Proof Protocol [10]. In what follows we refer to CSS as the Verifier and Authorisation Server interchangeably. As with the Aries Present Proof Protocol, the protocol assumes the holder has already interacted with an issuer to acquire the necessary VC. Here communications take place over secure HTTPS connections, which allows the app to

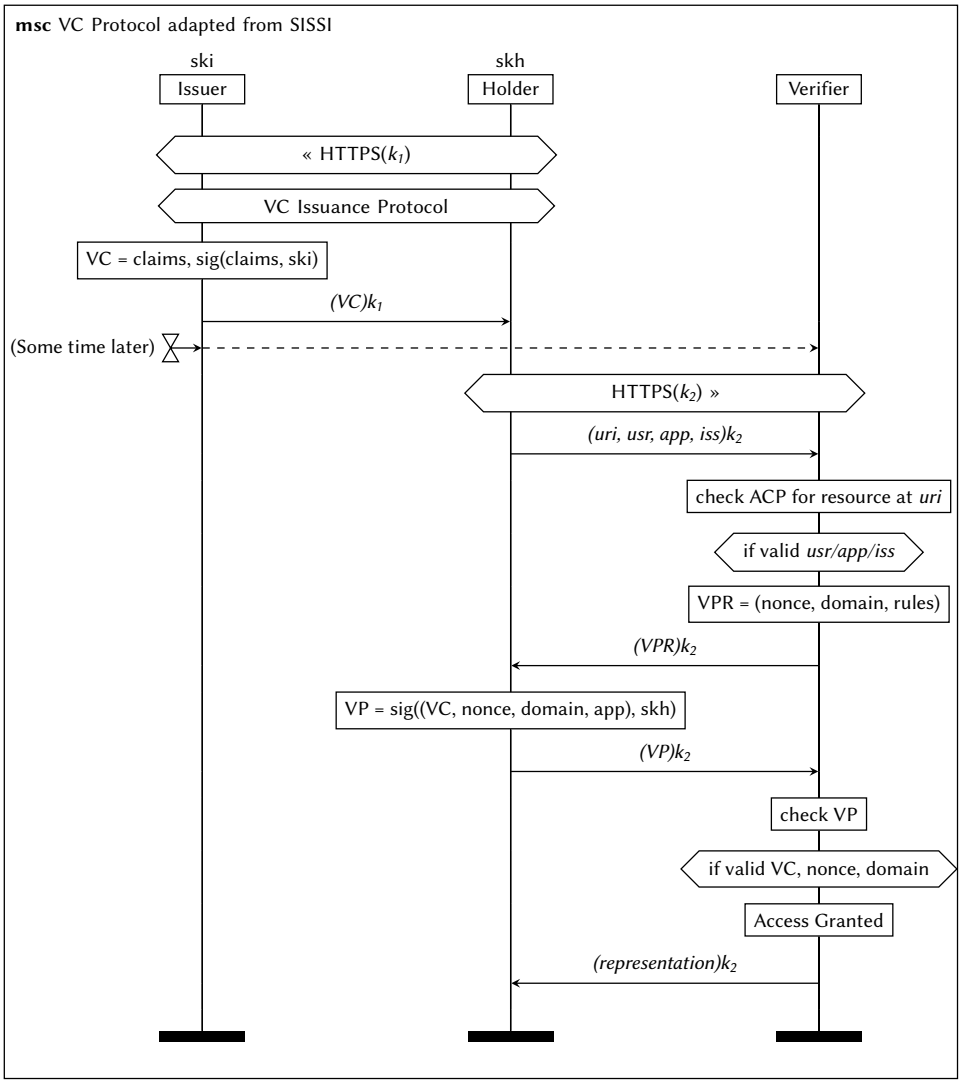


**Figure 2:** A VC-Based Protocol adapted from Hyperledger Aries framework, with communications taking place over HTTPS connections.

authenticate the authorisation server and to establish a symmetric session key and using it to encrypt messages. Our protocol consists of the following five steps, illustrated by the message sequence chart on the left of Fig. 2. (1), an app is used to request a resource hosted on a pod on the CSS, with a message that indicates a user, app and VC issuer. (2), the AS checks the access control policy to ensure it matches the indicated user/app/issuer combination. (3), if the combination is permitted, a 401 HTTP response is returned along with a Verifiable Presentation Request (VPR) that contains a nonce, and domain to protect against replay attacks. A nonce is a random number that is fresh with high probability. (4), the user creates a Verifiable Presentation (VP) using a VC from the appropriate issuer. The nonce and domain, previously sent by the server, are included inside the VP and sent to the authorisation server. (5), the authorisation

server ensures that the nonce and domain are correct. It then verifies the VP is valid and the issuer and user's signatures match. If everything is correct, access is granted to the resource.

### 3.2. Adaptation of Protocol to respect separation between apps and users

The protocol as described above does come with some limitations. As the Holder represents both the user and the application which interacts with the authorisation server, a secure flow requires that the application must be fully trusted by the user to create and use VPs without their explicit approval each time. To address this problem, we can separate the Holder into two distinct entities: a User and an App that communicate through HTTP requests and redirects. The User is responsible for acquiring and storing the VC, and has the ability to determine whether it gives consent to the App for its use in Verifiable Presentations with the authorisation server. The App initiates the request with the authorisation server and eventually accesses resources approved by the User. We describe next the implementation of this flow, which we call the VC-based Request Flow.

Consider the flow in Fig. 3. The App initiates the flow by making a request to the authorisation server for a resource and then receiving a request for a VP in the response. A request is then made to a server run by the User (possibly a Wallet app) containing the necessary information from the VPR – an indication of the user, app name, issuer, and the nonce and domain. Along with this, a URI is included to redirect back to the App. The User checks the provided information to ensure it is able to make an appropriate VP, and that the App's provided redirect URI is listed among its trusted URIs (and essential step for avoiding dishonest apps impersonating honest apps). The user themselves, via a browser or perhaps an app on the phone then redirects to an HTML page where it must be confirmed that the User wishes to provide a VP to the App. Following a successful confirmation, a VP is generated and returned to the App via the callback URI checked above. The App can then make another request to the authorisation server including the VP and gain access to the resource.

The separation between the user and the app in this flow allows a user to have control over how their VC is used by the app, by preventing the app using the VC without without the explicit involvement of the user. It also enables the user to enforce additional security measures and policies. For example, they may want to include an 'exp' field in a VP to ensure it is only valid for a specified amount of time.

We have verified multi-party authentication properties of this protocol in ProVerif [11]. Specifically, if any of the entities complete the protocol with honest participants, then the honest participants agreed on all the messages sent and received. Authentication is verified against a threat model where honest participants may also engage in sessions with compromised agents.

## 4. Integrating the Protocol into Community Solid Server

The Community Solid Server is an open-source implementation of the Solid specifications built using a dependency injection framework called Components.js [12] to determine how its various components are connected through JSON-LD configuration files. The server's architecture

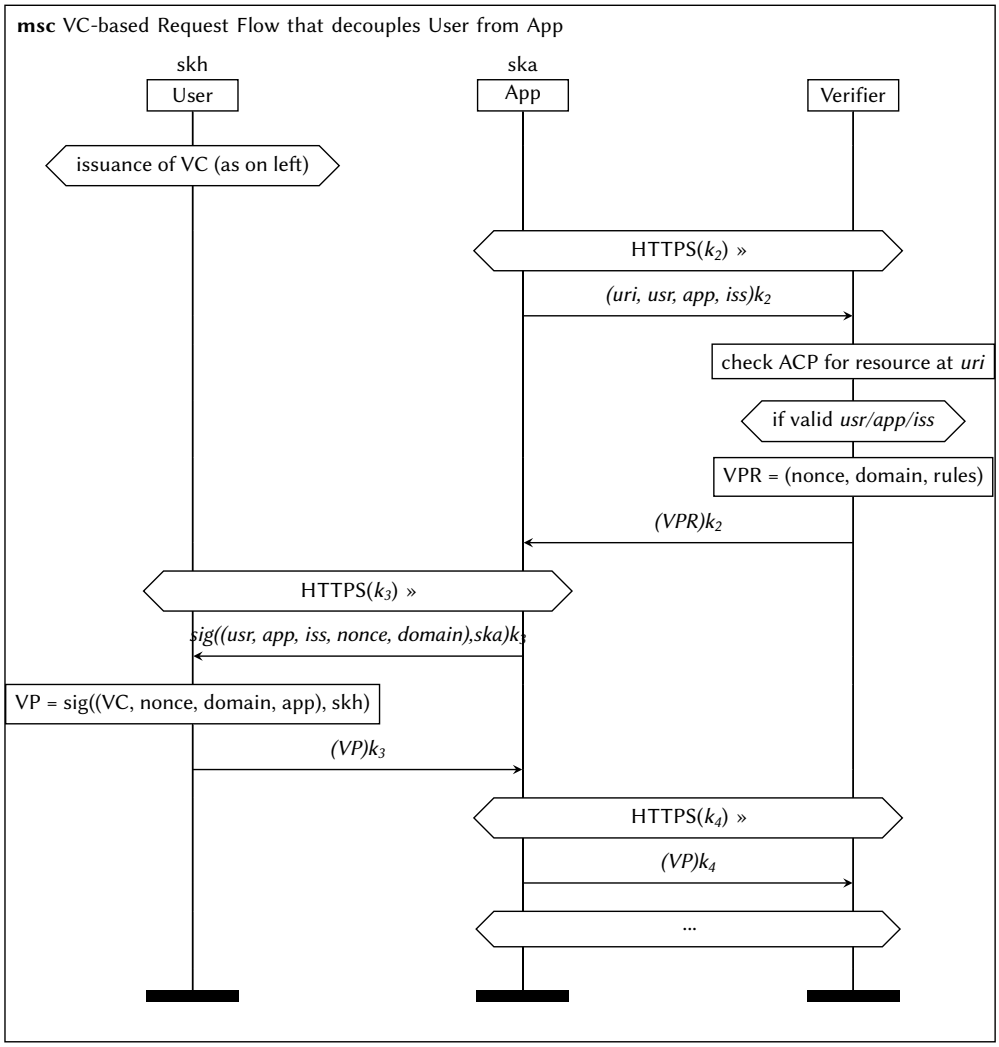


**Figure 3:** A VC-Based Protocol adapted for Solid where the app and user are decoupled. Issuance is as previously.

documentation[3] explains how its components are configured to respond to requests when its default config file is used. To implement the proposed protocol, changes are made to the server's HTTP, Authentication and Authorisation components and configuration files.

**HTTP:** The server needs to recognise when HTTP requests intend to follow this new protocol. To achieve this, a new **VcHttpHandler** component is created and included among the server's existing HTTP handlers. This new handler listens for and responds to requests which contain a `vc` header indicating it is the initial request for a resource, or a `vp` header indicating it is the secondary message in the protocol and a VP is present; and responds accordingly.

**Authentication and Authorisation:** Access Control Policy [13] is used instead of the

[3]Community Solid Server Architecture: https://communitysolidserver.github.io/CommunitySolidServer/7.x/architecture/overview/

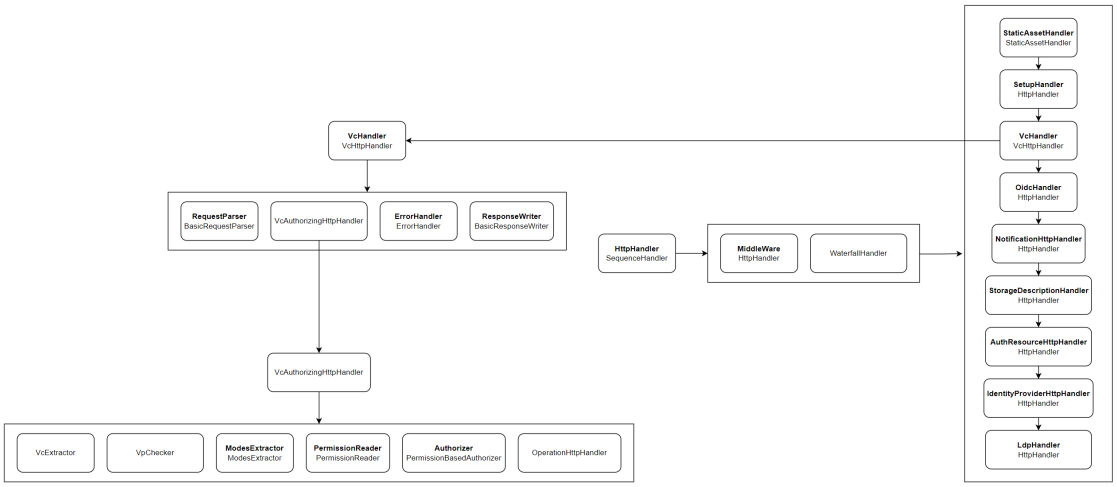


**Figure 4:** VcHttpHandler, VcAuthorizingHttpHandler components represented as architectural diagrams in a similar format to the CSS architecture documentation, with the updated HTTP component structure.

default Web Access Control (WAC) for authorisation. This is a newer method of determining resource access in Solid and is seen as a more flexible approach that is more suited to VCs. This is implemented by swapping out existing configuration files, with `webacl.json` and `acl.json` replaced with `acp.json` and `acr.json` respectively.

A **VcAuthorizingHttpHandler** component is created in order to perform the necessary steps for authorisation. During the initial request, it uses a simple component called **VcExtractor** to retrieve the user/app/issuer from the body of the HTTP request and returns them as a credentials object. Using this, it checks that the given user/app/issuer combination is listed in the ACP policies for the requested resource. If so, the VcHttpHandler responds with a Verifiable Presentation Request.

In the secondary request, it uses a **VpChecker** component to verify the VP and VC using libraries from the Decentralised Identity Foundation. Features such as the signatures of the issuer and holder, formatting, fields for expiration and issuance dates are checked to ensure it is valid. The necessary data is then used to create a credentials object which is passed to the VcAuthorizingHttpHandler, where it is again checked against the ACP policy to determine whether the request can be authorised.

## 5. Conclusion & Related Work

This paper presents an alternative approach to authentication and authorisation on Solid, with a VC-based protocol that can protect user privacy by avoiding the need for interaction with a third-party identity provider; in favour of an ecosystem involving issuers that require far less information. The protocol is adapted from an protocol for the Hyperledger Aries framework after considering security analysis and compatibility with the current development of the Community Solid Server. In particularly, the holder is decomposed into a separate app and a user so that the app never handles the VC. This adapted protocol is incorporated into a modified

version of the CSS, allowing for a demonstration that outlines its privacy-enhancing potential. Future work includes expanding the ACP policy used in the protocol in order to have access control rules specify attributes of a Verifiable Credential, allowing for more varied use-cases.

Regarding related work, Inrupt's Enterprise Solid Server currently has an experimental VC-based protocol used in their Access Grant service.[4] The ESS uses an authorisation mechanism based on access requests and grants. When an access request is received, it is serialized into a VC before the resource owner can decide whether or not to grant access. An access grant with 'approved' or 'denied' status is then also serialized as a VC which can be checked by the server's */verify* endpoint. It appears that the VCs involved in this protocol are created by the ESS server itself, rather than through a completely separate third-party issuer. Consequently, the unlinkability property of privacy cannot be protected for the user in this case.

## References


[1] S. Capadisli, T. Berners-Lee, R. Verborgh, K. Kjernsmo, Solid protocol, 2023. URL: https://solidproject.org/ED/protocol.

[2] A. Coburn, E. Pavlik, D. Zagidulin, Solid-OIDC, 2022. URL: https://solidproject.org/TR/oidc.

[3] J. Morgan, A. Coburn, M. Bosquet, Solid-OIDC primer, 2022. URL: https://solidproject.org/TR/oidc-primer.

[4] G. Schaus, Verifying multi-party authentication of Solid OIDC and VC protocols, 2023. MSc thesis, University of Luxembourg.

[5] C. Esposito, R. Horne, L. Robaldo, B. Buelens, E. Goesaert, Assessing the Solid protocol in relation to security and privacy obligations, Information 14 (2023) 411. doi:10.3390/info14070411.

[6] J. van Dijck, B. Jacobs, Electronic identity services as sociotechnical and political-economic constructs, New Media & Society 22 (2020) 896–914. doi:10.1177/1461444819872537.

[7] C. H.-J. Braun, R. Horne, T. Käfer, S. Mauw, SSI, from specifications to protocol? formally verify security!, ACM, 2024. doi:10.1145/3589334.3645426.

[8] M. Sporny, D. Longley, D. Chadwick, Verifiable Credentials data model v1.1, 2022. URL: https://www.w3.org/TR/vc-data-model/.

[9] C. H.-J. Braun, V. Papanchev, T. Käfer, SISSI: An architecture for semantic interoperable self-sovereign identity-based access control on the web, ACM, 2023, pp. 3011–3021. doi:10.1145/3543507.3583409.

[10] N. Khateev, S. Curran, Aries RFC 0454: Present proof protocol 2.0, 2021. URL: https://github.com/hyperledger/aries-rfcs/blob/main/features/0454-present-proof-v2/README.md.

[11] C. Cremers, S. Mauw, Operational Semantics and Verification of Security Protocols, Information Security and Cryptography, Springer, 2012. doi:10.1007/978-3-540-78636-8.

[12] R. Taelman, J. Van Herwegen, M. Vander Sande, R. Verborgh, Components.js: Semantic dependency injection, Semantic Web Journal (2022). URL: https://linkedsoftwaredependencies.github.io/Article-System-Components/.

[13] M. Bosquet, Access control policy (ACP), 2022. URL: https://solid.github.io/authorization-panel/acp-specification/.


[4] Access Grant Service `/verify` Endpoint: https://docs.inrupt.com/ess/latest/services/service-access-grant-verifier/