A Universal System for OpenID Connect Sign-ins with Verifiable Credentials and Cross-Device Flow

Blinded for Review

***Abstract*—Self-Sovereign Identity (SSI), as a new and promising identity management paradigm, needs mechanisms that can ease a gradual transition of existing services and developers towards it. Systems that bridge the gap between SSI and established identity and access management have been proposed but still lack adoption. We argue that they are all some combination of too complex, locked into specific ecosystems, have no source code available, or are not sufficiently documented. We propose a comparatively simple system that enables SSI-based sign-ins for services that support the widespread OpenID Connect or OAuth 2.0 protocols. Its handling of claims is highly configurable through a single policy and designed for cross-device authentication flows involving a smartphone identity wallet. For external interfaces, we solely rely on open standards, such as the recent OpenID for Verifiable Credentials standards. We provide our implementation as open-source software intended for prototyping and as a reference. Also, we contribute a detailed technical discussion of our particular sign-in flow. To prove its feasibility, we have successfully tested it with existing software and realistic hardware.**

***Index Terms*—Self-Sovereign Identity, Identity and Access Management, Verifiable Credentials, OpenID Connect, OAuth**

I. Introduction

Identity management has long been and will likely stay a core pillar of our Internet service ecosystem and economy. From simple social media accounts to rigorous know-your-client (KYC) compliant financial service accounts, identity is everywhere. And while the principles of authentication and authorization seem eternal, their implementation evolves with the times. Isolated centralized solutions have largely given way to federated identity management with hints of user-centric identity, such as data sharing consent prompts after sign-ins [1]. In this age of the single-sign-on (SSO), a handful of large corporations act as Identity Providers (IdP) and keep user account data that any service can integrate, given the IdPs permission. This has created an alarming concentration of data, censorship potential, and, ultimately, power.

At the core of the issues is that the IdP takes an active role in authentication and authorization processes. Thus, it always knows what its users access and when. Denying service to individuals or groups of users—be it by design or by accident—is as easy as taking no action. The concept of Self-Sovereign Identity (SSI) has evolved as the ultimate form of user-centric identity [2], reclaiming the active role for users. Initially described through a set of ten principles by Christopher Allen [2], SSI has become much more tangible through technical standards and open-source implementations.

Naturally, adopting and integrating an identity management approach fundamentally different from today's established Identity and Access Management (IAM) is a challenge. Even though there are initiatives from several sizable actors, including the European Union [1], change will likely need to be gradual and more organic. Existing IAM systems need to be phased out over time, and users must adopt and understand the new paradigm. Simple sign-in procedures arguably cover a majority of today's Internet use cases. Thus, building a system that can—at the very least temporarily—be used to bridge the gap between SSI credentials and established IAM solutions is necessary, and the few existing proposals have not seen significant adoption.

OpenID Connect (OIDC) [3] is arguably one of the most widely used IAM protocols. General implementation support is good[2] and implementations tailored to popular web frameworks are commonly available[3], it is the protocol of choice for today's dominant IdPs[4], and it is possible to integrate into arbitrarily complex IAM setups via Keycloak[5]. This is likely why existing proposals for SSI bridges have mainly focused on bridging to OIDC [4], which we improve upon.

In this work, we present the status quo of SSI bridging and highlight remaining problems. We then propose a simple yet powerful SSI-to-OIDC bridge that service providers can deploy to adopt SSI-based sign-in while still using the familiar OIDC. As part of this, we advocate for the separation of issuer and relying party software. We design and implement such a bridge that we make available as free and open-source software intended for prototyping and as a reference[6]. In the process, we emphasize cross-device protocol flows and the use of the recent standardization efforts belonging to the OpenID for Verifiable Credentials family [5]. Also, we provide a deep technical discussion of one of these flows: the authorization code flow. Finally, we validate our bridge with the involvement of existing software, running everything on separate physical devices.

II. Background

In this section, we provide the minimum necessary background knowledge needed to follow this work. We provide an overview of the concept of Self-Sovereign Identity, introduce OpenID Connect with an emphasis on the core standard, and

[1]https://ec.europa.eu/digital-building-blocks/sites/display/EBSI/EBSI+Verifiable+Credentials

[2]e.g., https://www.npmjs.com/package/oidc-client

[3]e.g., https://www.npmjs.com/package/oidc-react, https://www.npmjs.com/package/next-auth, https://www.npmjs.com/package/angular-oauth2-oidc

[4]https://developers.google.com/identity/openid-connect/openid-connect

[5]https://www.keycloak.org/docs/latest/server_admin/

[6]Code to be released early 2024

finally explore the newest additions to OpenID Connect that go beyond the core specification to provide some support for Self-Sovereign Identity.

A. Self-Sovereign Identity

SSI is a decentralized identity management paradigm that aims to return a higher degree of privacy and control to individual end-users [2] by letting them hold their own user data. As of writing, SSI has gained momentum in research and policy communities, but large-scale adoption is yet to come.

Key to the technical implementation of SSI are the W3C Verifiable Credential (VC) [6] and W3C Decentralized Identifier (DID) [7] standards. The VC standard defines a format for expressing claims, enabling the creation of digitally signed statements about subjects such as individuals, organizations, or machines. These credentials can be cryptographically verified. Complementing VCs, the W3C DID standard establishes a pattern for creating and managing globally unique decentralized identifiers that represent the subjects and issuers of VCs. The actual implementation is up to each specific DID method, allowing for great flexibility. Usually, these work by leveraging asymmetric encryption keys, similar to how blockchain accounts work.

For end-users, the entry into any SSI ecosystem likely begins with a smartphone wallet application. SSI wallets are comparable to crypto wallets in that they store private key material. However, their keys represent DIDs. In addition, SSI wallets also provide means to store VCs, display VCs, and implement protocols to present and receive VCs.

Presenting a VC happens in the form of a Verifiable Presentation (VP) that contains at least one VC and a challenge and is signed by the holder to prevent replay attacks. A relying party (RP) can cryptographically verify the VP and its VC(s) without interacting with the issuer. If designated in a VC, the issuer will also query a status list to ensure the VC is not revoked.

B. OpenID Connect

OpenID Connect (OIDC) [3] is an open standard designed for secure user authentication and authorization. It is built on top of the OAuth 2.0 [8] authorization framework, providing an additional layer of identity verification. OIDC facilitates the exchange of user information between the identity provider (IdP) and the relying party (RP) in a secure and standardized manner.

The OIDC protocol introduces a set of standardized identity flows adapted from OAuth 2.0, such as the Authorization Code Flow, Implicit Flow, and Hybrid Flow, allowing clients operated by an RP to request and obtain identity information about users. To limit what information a client gets, OpenID Connect defines a set of scopes that determine the user attributes transmitted to the client during the sign-in process. Ultimately, the client ends up receiving tokens. An `id_token` is used for identity data, such as an email address. An `access_token` encodes information relevant to access control, such as membership in a user group.

One of the strengths of OpenID Connect is its ability to support single sign-on (SSO) scenarios, enabling users to log in once and access multiple services without the need for separate authentications.

C. OpenID for Verifiable Credentials

Recently, OpenID Connect has been expanded by OpenID for Verifiable Credentials (OID4VC) [5]. Originally, OID4VC was a family of three specifications:

- **OpenID for Verifiable Credential (OID4VCI)** defines an API for VC issuance from a server to a wallet. It essentially covers the download of a signed credential.
- **OpenID for Verifiable Presentations (OID4VP)** specifies how a wallet can present a VP to a server based on an OAuth 2.0 flow [9].
- **Self-Issued OpenID Provider v2 (SIOPv2)** enables SSI wallets to act as OIDC Providers [10].

Since then, more have been added[7], but considering our focus on sign-ins, OID4VP and SIOPv2 are the most relevant specifications of that family. They can be combined to create a standardized way for smartphone wallets to authenticate and authorize a user with claims from VCs.

III. Related Work

With the field of Self-Sovereign Identity (SSI) having enjoyed steady attention from researchers in the past years, several authors have contributed work related to the topic of integrating SSI into established IAM systems. These contributions roughly fit into three categories. We start by mentioning surveys about integrations of SSI into existing IAMs, then go over noteworthy integrations with OIDC, and finally, integrations with other IAMs.

A survey by Kuperberg et al. [4] looking at SSI integrations for established IAM protocols identifies only seven relevant candidates. The majority are commercial, and only 2 of them are available as open-source software. They note a need for code examples and an explanation of implementation specifics.

In another survey, Grüner et al. [11] formalize and compare different interoperability concepts for SSI. They are unable to clearly identify a superior concept. Also focused on interoperability, Yildiz et al. [12] once more emphasize the need for true interoperability in SSI and describe it as a key requirement for further adoption. They introduce an SSI reference model that structures SSI components and functionalities into different layers. They also point out where specific protocols and standards could be applied on these layers, including OIDC technology standards.

A general survey of the state of SSI from 2022 by Schardong et al. [1] notes that protocol integration into established IAMs is vital to pave the adoption for SSI. The authors provide a comprehensive list of existing efforts.

Grüner et al. introduced a system facilitating IAM between the old SSI ecosystems of Jolocom and uPort, as well as the still-existent Hyperledger Aries, and the established IAMs

[7] https://openid.net/sg/openid4vc/specifications/

OIDC and SAML2 over the course of two works [13], [14]. The presented system works as a two-way integration and consequently assumes the role of issuer and relying party simultaneously. An elaborate trust model, in combination with attribute mapping support, enables the forwarding of claims under unified names while marking untrusted claims by renaming them further. The most significant issue with this architecture lies in its complexity and the significant work needed to integrate new blockchain-based SSI ecosystems.

Lux et al. [15] propose an integration architecture that allows OIDC sign-in via SSI based on a blockchain public key infrastructure. Namely, they are looking at Sovrin, which uses Hyperledger Indy. In their implementation, they write attributes to the OIDC `id_token` to avoid reliance on non-default OIDC features. Unfortunately, the solution is exclusive to Hyperledger Indy.

At least two open-source solutions without academic contribution exist. The first one is the Province of British Columbia's Verifiable Credential Authentication with OpenID Connect (VC-AuthN OIDC)[8]. It supports a cross-device flow where the wallet is interfacing via DIDComm and uses custom presentation requests. Information from presented credentials is mapped into the OIDC `id_token`. Unfortunately, the implementation seems to be mostly, if not only, compatible with their BC Mobile Wallet, and the trust model is unclear.

The second open-source effort identified is IdP Kit[9] from walt.id and looks promising. From what the documentation[10] show, the project focuses on OIDC compliance and additionally offers sign-in based on Non-Fungible-Token possession. The biggest downsides seem to lie in the lack of detail regarding wallet interaction implementation and the amount of required configuration. However, public deployments were unavailable, and recent builds were failing, as also acknowledged in the documentation.

Moving on to other IAMs, Hong et al. [16] develop an architecture that allows sign-in through OAuth 2.0, which is also used in OIDC. It is heavily blockchain-focused and requires one identification smart contract to be deployed per user. This is likely too complex for the average user in the near future, and leaves open the question of who will pay user registration costs. Since then, the standardization of DIDs has offered more usable options for mass adoption.

Yildiz et al. [17] present an architecture making SSI credentials available for SAML sign-in. The implementation relies on Hyperledger Indy and Aries. It supports a cross-device flow via DIDcomm. That means that the SSI-facing interface of the architecture is locked into Hyperledger Indy.

IV. Open Challenges

In this section, we identify and examine open challenges in the area of SSI bridges. Our aim is not to highlight the general complexity of a bridge, but rather to focus on aspects that have not, or have not sufficiently, been addressed by other sources.

[8]https://github.com/bcgov/vc-authn-oidc

[9]https://github.com/walt-id/waltid-idpkit

[10]https://docs.walt.id/v/idpkit/getting-started/quick-start

A. Fragmented Presentation Protocols

The exchange of a VP from a wallet to a server has long been fragmented in terms of protocols. Many wallet developers simply built their own. Also, most of these protocols lack standardized support for the cross-device flow that will likely dominate as wallets almost exclusively come in the form of mobile apps. Both of these are currently beginning to change, with the OID4VP and SIOPv2 standards gaining attention and adoption. The former standard features developer considerations that shed some light on cross-device flows [9]. However, the standard is still in development, and especially the hand-off back from phone to desktop is still too abstract. Contemporary wallet apps feature no mechanism to directly redirect a browser running on another device.

B. SSI Ecosystem Dependence

While interoperability is one of the fundamental ideas of SSI, implementers in practice have shown a tendency to build SSI ecosystems with a certain level of lock-in. This can easily happen through using DID methods living on a specific blockchain, a custom solution for trusted issuer management in the absence of a common standard, or custom status list implementations seeking more decentralization than StatusList2021 [18] can provide. That leads to a bridge that is suitable only for a very select group of issuers and, by extension, service providers.

C. High Bridge Complexity

Previous bridging solutions tend to be fairly complex in design and setup, which goes entirely against the idea of simplifying SSI adoption. This can stem from supporting too many SSI ecosystems with custom drivers that increase the necessary amount of configuration, often also relying on multiple configuration interfaces. In practice, bridges are not just desirable for legacy services but also as an easy solution to prototype and build new ones. They need to be simple to run.

Furthermore, designs like the one by Grüner et al. [14] incur additional complexity by including full issuance capability when it arguably is not necessary for many use cases. Embracing SSI does not necessitate issuing every piece of user information. Only key attributes that conceivably have value outside the service of origin must be issued.

D. Lack of Concrete Technical References

As also noted by Kuperberg et al. [4], there is a distinct lack of code examples and practical implementation considerations. Specifications leave room for interpretation or significantly abstract parts of a flow. While seeing a full authorization code flow in less than ten steps is great as a gentle high-level introduction, it is not sufficient to base an implementation upon.

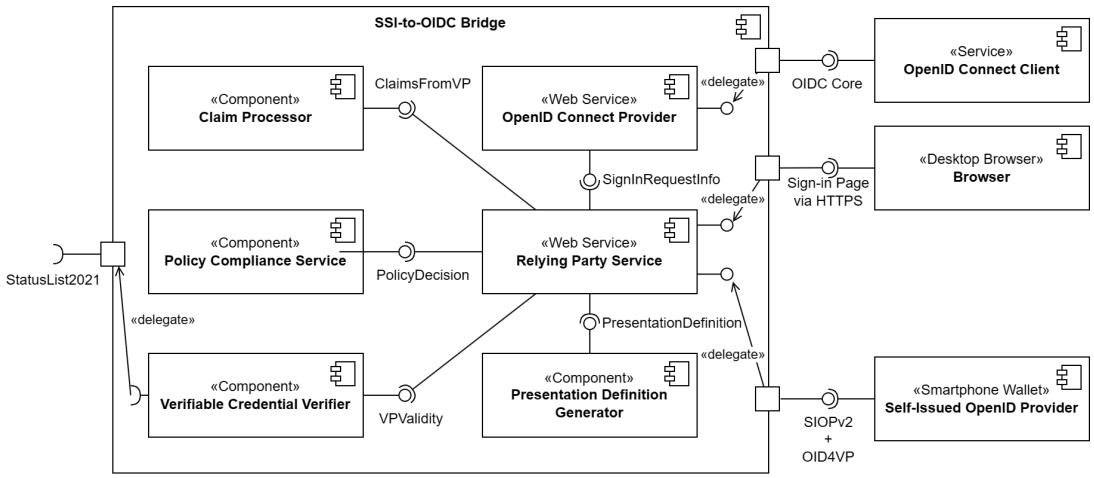

Fig. 1. A component diagram of the SSI-to-OIDC Bridge's conceptual components and its interfaces.

V. System Architecture

From a high-level perspective, the architecture we propose is two OIDC Providers with one nested inside the other. While the first one, which is realized through SIOPv2 with OID4VP, could, in theory, function as a standalone one, the reliance on the non-OIDC-Core `vp_token` makes it unsuitable for use with most clients. To remedy this, the second OIDC Provider encapsulates this first one and communicates entirely based on the OIDC Core that any current OIDC client supports.

In the remainder of this section, we discuss how that design translates into conceptual components and present, as well as explain, some further decisions we had to make.

A. Deployment Considerations

Before diving into the architectural details, it is important to recognize how the artifact must be deployed, as that impacts the architecture. Acting as an OIDC Provider towards service clients, the bridge has full authority over transmitted user information with no accountability. Thus, the only feasible deployment option is for every service provider to deploy one themselves. Prior work has also reached this conclusion [11].

B. Login Policy

To simplify setup, the whole system is configured with one main configuration file: the login policy. This defines what credentials to ask wallets for, which issuers to trust, what claims to accept, and how to transform those claims. While there are existing standards for specifying some of these, unifying everything into one file and format eliminates data duplication and avoids confusion about what files are security critical.

An example is shown in Listing 1. It defines a sign-in requesting one credential that contains a verified email address and is issued by one of two distinct issuers that organize this claim differently. We will refer back to this login policy and discuss how exactly different components use it.

```
[{
  "credentialID": "expected_credential_for_name",
  "patterns": [{
    "issuer": "did:example:123",
    "claims": [{
        "claimPath": "$.credentialSubject.e_mail",
        "newPath": "$.email",
        "token": "id_token"
    }]
  },
  {
    "issuer": "did:example:456",
    "claims": [{
        "claimPath": "$.credentialSubject.email",
        "token": "id_token"
    }]
  }]
}]
```

Listing 1: An example of a login policy for a service depending on the user's email address.

C. Conceptual Components

In this section, we describe our proposed system's conceptual architecture, as shown in Figure 1. They represent units of functionality and do not necessarily need to be implemented separately. The components are:

1) OpenID Connect Provider: An OpenID Connect Provider (OIDC Provider) is a service or software component that implements the OIDC protocol to enable secure and standardized user authentication and authorization. In our architecture, it is the interface for OIDC clients. At a minimum, it should support the common OAuth2.0 Authorization Code flow [8]. The more robust and fully compliant this OIDC Provider is, the better the interoperability with existing OIDC clients.

It would ordinarily be possible for a client to request specific user data via OIDC scopes. For the user, that creates transparency about the transferred data. With our use of VCs, users actively choose what they share and give consent inside the wallet application when presenting VCs. That renders a consent step in the top-level OIDC flow unnecessary. Similarly,

using OIDC scopes is superfluous because the Relying Party Service communicates what kind of data is expected to the wallet. This is feasible because the OIDC client and bridge are operated by the same party. We only expect clients to send the mandated default `openid` scope. Further scopes are ignored.

2) Relying Party Service: The Relying Party (RP) Service adheres to OID4VP with SIOPv2 to request and receive a Verifiable Presentation from a wallet. OID4VP dictates that a presentation definition from the DIF Presentation Exchange (PEX) [19] specification is used to inform the wallet how many credentials of what kind are expected. This is not security critical in our system, but it is necessary to improve the user experience because wallets will usually pre-select suitable credentials based on the presentation definition.

3) Presentation Definition Generator: The Presentation Definition Generator creates a presentation definition based on a login policy, such as the one in Listing 1. The policy specifies one or more credentials it expects, giving each one a unique `credentialID`. For each expected credential, at least one pattern defines expected claims on a per-issuer basis. All patterns from the policy are turned into input descriptors requesting the presence of the claim without specifying filters. The Submission Requirement Feature [19] is used to mirror that only one of the patterns from each expected credential needs to be fulfilled. Should a service provider require more control, a custom array of input descriptors can be supplied optionally, overriding the Presentation Definition Generator.

4) Verifiable Credential Verifier: The Verifiable Credential Verifier's job is the cryptographic and structural verification of W3C VCs and VPs according to specification. In addition to that, revocation following the W3C's StatusList2021 [18] proposal is supported. The status list server is optional and operated by the respective issuer that wishes to support revocation.

Our verifier also mandates that all VCs inside a VP are issued to the DID that the VP was signed by. This is a simple way to ensure a holder can only present VCs that make claims about him, and it is sometimes referred to as a holder binding.

5) Policy Compliance Service: To ensure that presented credentials contain the required claims and are issued by trusted issuers, the Policy Compliance Service evaluates a VP with respect to the login policy. An example policy is given in Listing 1. Each VC in a submitted VP needs to be matched to exactly one of the expected credentials in the policy. For each expected credential, the acceptable patterns are iterated, starting with the first one. Each pattern lists claims by JSONPaths following the Presentation Exchange specification [19]. A VC matches a pattern if the issuer matches and if all required claims are present. By default, every claim is assumed to be required. Our example policy expects one credential and declares two different patterns for it.

6) Claim Processor: Claims from the VP a user presents must be made available via OIDC in a way that is readable by current OIDC clients. The new `vp_token` defined in OID4VP [9] could be used, but it is infeasible to rely on the quick adoption of a draft specification that will likely never be supported by many legacy systems depending on OIDC when we design for interoperability. Thus, claims need to be transferred into the established `id_token` and `access_token` [3] to work with all OIDC clients. The Claim Processor's task is to turn a VP into token payloads according to a login policy.

Because existing services incorporating an OIDC client likely depend on standard OIDC claim names, transferring claims might not be sufficient, leading us to support renaming. This is also necessary if two trusted issuers use different claim names for the same piece of information, as can be seen in our example login policy in Listing 1. Every claim has a `claimPath`, which is a JSONPath. Optionally, `newPath` provides a new JSONPath for the claim value inside the corresponding OIDC token. It defaults to the last element in the `claimPath` and, if explicitly specified, must always point to exactly one location. The target token is specified using the `token` property, which defaults to the `access_token` if not present. In summary, this allows arbitrary remapping of claim data.

In the case that a `claimPath` points to more than one value, a `newPath` is required. All claims in the original path are aggregated as a JSON object and indexed only by their ultimate JSONPath element. That object is written to the new path.

D. DID Method Support

The challenges posed by the DID standard's flexibility are numerous. The one mainly relevant for this work is the software—and perhaps protocol—support needed to resolve DIDs. We choose a pragmatic solution for this by recommending the use of simple lightweight DIDs, as defined in [20]. Every other DID method type would require another interface implementation custom to the used storage solution. Using `did:pkh` [21] specifically has the added benefit of leaving an upgrade path for blockchain-based DIDs. For example, a `did:pkh:eth` could also be used as the feature-richer `did:ethr` [22].

E. Limitations

While we think that a bridge is a great option for fast prototyping and upgrading existing services, there are limitations to be aware of. First, services only have access to user attributes, not an interactive wallet connection. So if a service requires a wallet interaction to, for instance, obtain a user signature, relying on OIDC might not be advisable.

Next, we have chosen to keep our bridge as generic and simple as reasonably possible. As mentioned earlier, that entails support for a limited set of DID methods and for only one status list specification. However, future implementations could simply choose to support what is deemed appropriate for a given use case, provided expertise and development resources are sufficient. Some changes might be relatively easy.

```
openid-vc://
?client_id=<BRIDGE_DID>
?request_uri=<BACKEND_URL>/api/presentCredential
%3Flogin_id=<UUID>
```

Listing 2: QR code contents for starting a credential exchange process with a wallet via SIOPv2 and OID4VP.

Finally, the simple holder binding our bridge enforces makes it impossible for users with multiple VCs legitimately issued to different DIDs to present them in one VP and be accepted.

VI. Implementation

For our implementation, we were looking for a robust and configurable open-source IDP. We ended up choosing Ory Hydra[11], which is a free and open-source OIDC Provider. Its compliance is officially verified by the OpenID Foundation. Setting up Hydra is reasonably convenient via Docker[12] and the sign-in flow is by default incorporating redirects to custom web services for login and consent[13]. That lets us build our proof of concept without touching Hydra's source code.

Our custom implementation called *vclogin* will take on the role of the login and consent services for Hydra. Additionally, we bundle all other remaining conceptual components into vclogin to keep deployment complexity down. For this reason, we chose to build with Next.js[14], as it allows us to combine user-facing web pages with API routes in a single code base.

Our code base comes with a Docker Compose file that contains all the configuration to run the entire bridge. That includes a PostgreSQL[15] Database for Hydra and a Redis[16] for vclogin. The required configuration is minimal and documented. The code is intended for prototyping and as a reference. It has not undergone a security audit, and it is not intended to be used in production.

The Redis store only acts as temporary storage, and all data is set to expire within minutes. For our use case, the PostgreSQL database can also be reset without effect. Thus, the entire bridge is essentially stateless and can be migrated without a data and database migration.

A. Detailed Protocol Flow

The biggest challenge in engineering the bridge lies in the specifics of the cross-device sign-in flow. As an example, we will now examine the adaption of perhaps the most common OAuth 2.0 flow: the Authorization Code flow [8]. Our protocol flow can be seen in Figure 2. We have slightly simplified some parts while keeping the depiction close to the technical reality. For example, we have omitted responses for Redis interactions.

At the start of the depicted flow, the user is assumed to have opened a service website hosted by the OIDC client web server in their browser. The flow starts normally until the user is redirected to the login page. All it features is a QR code that contains the necessary information to involve the smartphone wallet in the flow, which is depicted in Listing 2.

The `request_uri` tells the wallet where to fetch a presentation request. Notably, we add a custom query parameter called `login_id` to use as a challenge for the later presentation exchange and to allow the vclogin backend to know which browser and wallet are involved in the same sign-in procedure. Generating a UUID as the `login_id` in step 4 is necessary because the login challenge generated by Hydra is so large it cannot fit into a readable QR code. To keep track of this mapping between UUID and challenge, it is written to Redis. When the wallet asks for a presentation request in step 9, it also automatically provides vclogin with the `login_id`. It is used as the VP challenge during the following standardized Presentation Exchange flow.

After step 17, vclogin has received a VP. Following successful processing, vclogin exchanges the UUID challenge from the VP back into the login challenge to confirm the authentication and authorization to Hydra. The subject is identified by its used DID. Additionally, all processed claims from the VP are written to Redis to be accessible during the next phase.

At this point, the active role in the sign-in process needs to be handed back to the browser. To do so, vclogin has written the `redirect_uri` for the browser to Redis. Using the original login challenge, the browser periodically requests the redirect to see if the VP submission has happened yet. When it succeeds, it executes the redirect in step 29.

Now, the consent phase of the authorization code flow would take place, but as we do not need to ask for consent a second time, vclogin automatically completes it. Using the new `consent_challenge`, vclogin can request metadata on this consent process from Hydra in step 31. Most importantly, that includes the subject DID, which is used to retrieve the processed claims that were previously written to Redis in step 23. Then, vclogin confirms the user's consent to Hydra and provides the claim data for tokens in step 34.

Finally, the sign-in procedure concludes as normal for an authorization code flow from step 35 and on. The only noteworthy exception is that the bridge never provides a `refresh_token` because the bridge must not depend on caching critical identity data and should remain stateless in the larger picture.

VII. Evaluation

We evaluated our design and implementation in two ways: practically by performing a realistic sign-in and conceptually by recalling the open challenges we identified in the beginning.

A. Practical Testing

For real-world testing, we used the Altme Wallet[17]. It is open-source, actively worked on, and regularly updated to its

[11] https://github.com/ory/hydra
[12] https://www.docker.com/
[13] https://www.ory.sh/docs/oauth2-oidc/custom-login-consent/flow
[14] https://nextjs.org/
[15] https://www.postgresql.org/
[16] https://redis.io/
[17] https://github.com/TalaoDAO/AltMe

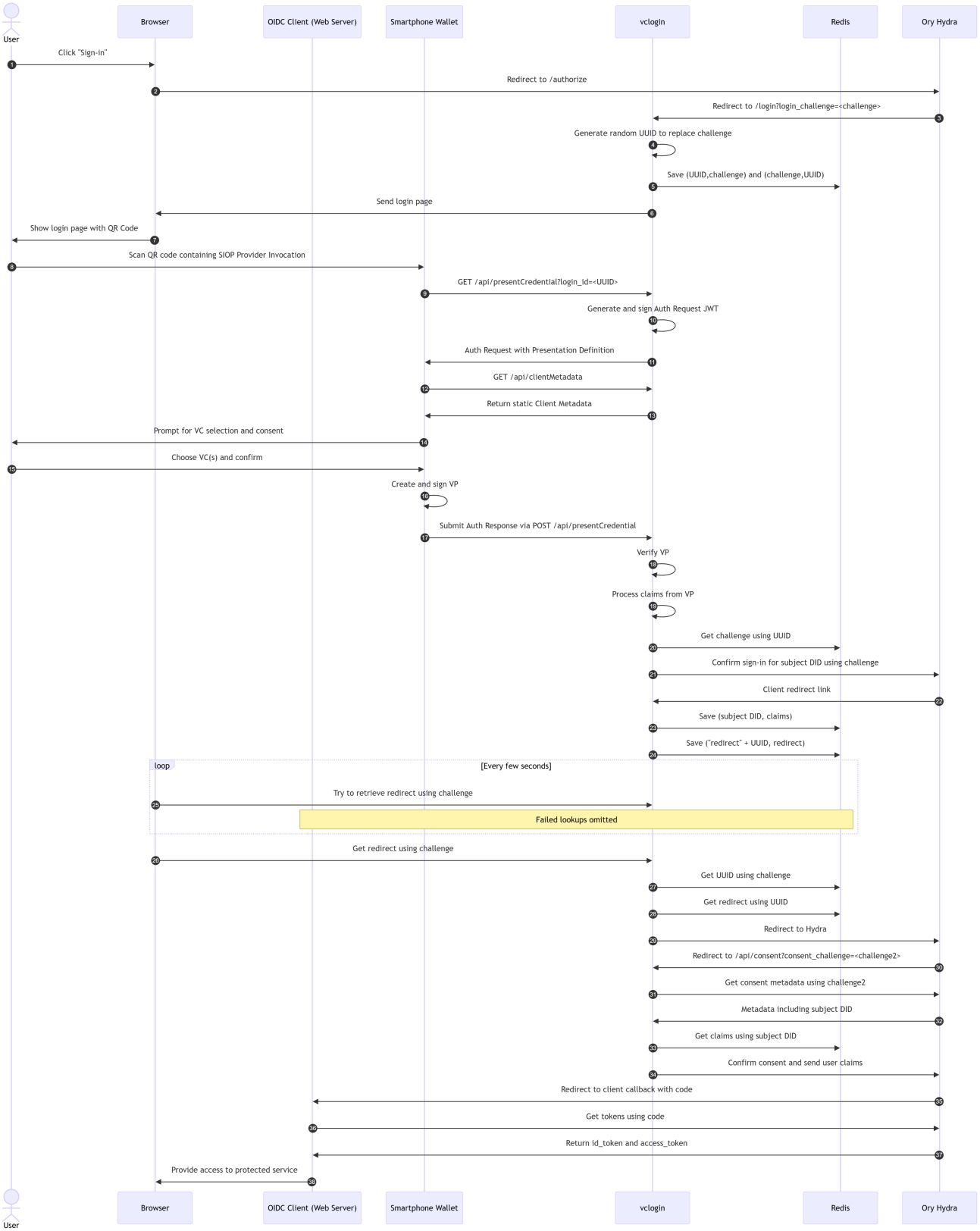

Fig. 2. A simplified sign-in procedure using the authorization code flow with our bridge. At the start of the presented sequence, we assume that the user has accessed the web page provided by the OIDC client.

```
"credentialSubject": {
  "id": "did:key:z6M...",
  "email": "name@example.com",
  "type": "EmailPass",
  "issuedBy": { "name": "Altme" }
}
```

Listing 3: The `credentialSubject` key in an Altme Proof of email VC with shortened `id` field.

```
DID_KEY_JWK=<jwk>
EXTERNAL_URL=https://examplebridge.com
LOGIN_POLICY=./acceptEmailFromAltme.json
PEX_DESCRIPTOR_OVERRIDE=./descrEmailFromAltme.json
```

Listing 4: An example `.env` file used to configure the SSI-to-OIDC bridge with a JWK placeholder.

newest version on Apple's App Store and Google's Play Store for convenient installation. This particular wallet combines the capabilities of a crypto wallet and an SSI wallet. For our testing, it is essential that the wallet supports OID4VP with SIOPv2. In Altme, a *Proof of email* VC is offered, which is one of the VCs we used in our tests. The relevant claims in that VC are shown in Listing 3. It is signed by a `did:web` and contains a StatusList2021 entry.

As an OIDC client, we will use the client Ory includes in Hydra's command line interface (CLI) for testing[18]. It hosts a website with a sign-in button and shows session metadata upon successful sign-in.

Before starting the SSI-to-OIDC bridge, we need to ensure a fitting configuration. The environment variables seen in Listing 4 need to be set, where `DID_KEY_JWK` is the key used as a `did:key` to authenticate to the wallet with. While `PEX_DESCRIPTOR_OVERRIDE` is optional, we use it here to combat a crash likely stemming from Altme's implementation of the Presentation Exchange specification by only requesting the credential type. The login policy we used can be found in Listing 5.

After starting the bridge, we can use Hydra's CLI to register a new client and then start our test client. Now, we perform these steps:

1) Click "Authorize application" and get redirected.
2) Scan the QR code below the text "Scan the code to sign in!" with the wallet.
3) Select one email VC from the list and press "Present" in the wallet.
4) Confirm the choice with the biometrics of the smartphone.
5) Website redirects to a page showing the received tokens, with the `id_token` being as configured. It can be seen in Listing 6.

B. Addressed Challenges

Looking at all of the overall challenges we identified, we have addressed all of them in various ways:

[18]https://www.ory.sh/docs/cli/ory-perform-authorization-code

```
[{
  "credentialID": "one",
  "patterns": [{
    "issuer": "did:web:app.altme.io:issuer",
    "claims": [{
      "claimPath": "$.credentialSubject.email",
      "token": "id_token"
    }]
  }]
}]
```

Listing 5: A login policy accepting a VC issued by Altme, containing an email.

```
{
  ...
  "email_verified": "name@example.com",
  "sub": "did:key:z6M..."
}
```

Listing 6: A shortened `id_token` payload containing an email.

Fragmented Presentation Protocols Our bridge relies fully on SIOPv2 and OID4VP, which show considerable promise to become the quasi-default, learning from and being associated with established IAM.

SSI Ecosystem Dependence Until further standards are established, we avoid custom protocols, stick to simple lightweight DID methods, and support revocation via just the established StatusList2021 spec.

High Bridge Complexity Our design has one configuration file with powerful defaults. A local test instance of the bridge can be run within minutes.

Lack of Concrete Technical References We have contributed reference code and an in-depth look at a bridged authorization code flow.

VIII. Conclusion

We have proposed an architecture that can simplify the adoption of SSI for sign-ins and described it in detail. To prove its feasibility, we have fully implemented this SSI-to-OIDC bridge and successfully tested it with existing software and realistic hardware. Our complete code repository is slated for open-source release in early 2024[19] and will be further evaluated as part of a Gaia-X project for decentralized business-to-business cooperation.

In the long term, a mechanism to standardize VC types and claim names is needed. Otherwise, any bridge needs to remap claims as soon as more than one issuer is involved, which is almost inevitable, given that SSI has decentralization at its core. Remapping claims complicates configuration and could introduce security-critical errors. This is a governance issue but also a technology deficit: relying on VC contexts and their types is not secure. Similarly, an easier way to organize collections of trusted issuers would simplify the configuration and administration of a bridge.

[19]Link will be inserted in camera-ready version

REFERENCES

[1] F. Schardong and R. Custódio, "Self-Sovereign Identity: A Systematic Review, Mapping and Taxonomy," *Sensors*, vol. 22, no. 15, p. 5641, Jan. 2022.

[2] C. Allen, "The path to self-sovereign identity," 2016, (Accessed 21-11-2023). [Online]. Available: http://www.lifewithalacrity.com/2016/04/the-path-to-self-soverereign-identity.html

[3] OpenID Foundation, "OpenID Connect Core 1.0 incorporating errata set 1," https://openid.net/specs/openid-connect-core-1_0.html, 2014, (Accessed 02-12-2023).

[4] M. Kuperberg and R. Klemens, *Integration of Self-Sovereign Identity into Conventional Software Using Established IAM Protocols: A Survey*. Gesellschaft für Informatik e.V., 2022.

[5] K. Yasuda, T. Lodderstedt, D. Chadwick, K. Nakamura, and J. Vercammen, "Openid for verifiable credentials," https://openid.net/wordpress-content/uploads/2022/06/OIDF-Whitepaper_OpenID-for-Verifiable-Credentials-V2_2022-06-23.pdf, 2022, (Accessed 02-12-2023).

[6] M. Sporny, D. Longley, and D. Chadwick, "Verifiable Credentials Data Model v1.1," https://www.w3.org/TR/vc-data-model/, 2022, (Accessed 03-12-2023).

[7] M. Sporny, D. Longley, M. Sabadello, D. Reed, O. Steele, and C. Allen, "Decentralized Identifiers (DIDs) v1.0," https://www.w3.org/TR/did-core/, 2022, (Accessed 03-12-2023).

[8] D. Hardt, "The OAuth 2.0 Authorization Framework," RFC 6749, Oct. 2012. [Online]. Available: https://www.rfc-editor.org/info/rfc6749

[9] O. Terbu, T. Lodderstedt, K. Yasuda, and T. Looker, "OpenID for Verifiable Presentations - draft 18," https://openid.net/specs/openid-4-verifiable-presentations-1_0-ID2.html, 2023, (Accessed 03-12-2023).

[10] K. Yasuda and M. Jones, "Self-Issued OpenID Provider v2," https://openid.net/specs/openid-connect-self-issued-v2-1_0-ID1.html, 2022, (Accessed 03-12-2023).

[11] A. Grüner, A. Mühle, and C. Meinel, "Analyzing Interoperability and Portability Concepts for Self-Sovereign Identity," in *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. Shenyang, China: IEEE, Oct. 2021, pp. 587–597.

[12] H. Yildiz, A. Küpper, D. Thatmann, S. Göndör, and P. Herbke, "A Tutorial on the Interoperability of Self-sovereign Identities," Aug. 2022.

[13] A. Grüner, A. Mühle, and C. Meinel, "An Integration Architecture to Enable Service Providers for Self-sovereign Identity," in *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*. Cambridge, MA, USA: IEEE, Sep. 2019, pp. 1–5.

[14] ——, "ATIB: Design and Evaluation of an Architecture for Brokered Self-Sovereign Identity Integration and Trust-Enhancing Attribute Aggregation for Service Provider," *IEEE Access*, vol. 9, pp. 138 553–138 570, 2021.

[15] Z. A. Lux, D. Thatmann, S. Zickau, and F. Beierle, "Distributed-Ledger-based Authentication with Decentralized Identifiers and Verifiable Credentials," Jun. 2020.

[16] S. Hong and H. Kim, "VaultPoint: A Blockchain-Based SSI Model that Complies with OAuth 2.0," *Electronics*, vol. 9, no. 8, p. 1231, Aug. 2020.

[17] 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, Sep. 2021.

[18] D. Longley and M. Sporny, "Status List 2021 — w3.org," https://www.w3.org/community/reports/credentials/CG-FINAL-vc-status-list-2021-20230102/, 2023, (Accessed 01-12-2023).

[19] D. McGrogan, G. Cohen, O. Steele, W. Chang, D. Chadwick, J. Hensley, N. Klomp, and A. Kesselman, "DIF Presentation Exchange 2.0.0," https://identity.foundation/presentation-exchange/spec/v2.0.0/, 2022, (Accessed 03-12-2023).

[20] F. Hoops, A. Mühle, F. Matthes, and C. Meinel, "A taxonomy of decentralized identifier methods for practitioners," in *2023 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*. IEEE, 2023, pp. 57–65.

[21] W. Chang, C. Lehner, J. Caballero, and J. Thorstensson, "did:pkh method specification," https://github.com/w3c-ccg/did-pkh/blob/main/did-pkh-method-draft.md, 2023, (Accessed 03-12-2023).

[22] Veramo Core Team, "Ethr did method specification," https://github.com/decentralized-identity/ethr-did-resolver/blob/master/doc/did-method-spec.md, 2022, (Accessed 03-12-2023).