Designing a Decentralized Identity Verification Platform

Surya Bahadur Kathayat

Norwegian University of Science and technology, Norway
surya.b.kathyat@ntnu.no

Abstract. In today's world, almost every private and public service has either fully embraced digitalization or is in the process of doing so, which is generally a positive trend. However, significant foundational challenges remain. Many services still depend on various centralized physical and digital identities, making it increasingly difficult for users to securely and reliably prove their identity with trusted credentials. Additionally, there are other risks, such as single points of failure, identity theft and fraud, censorship and discrimination, and limited user autonomy. To address some of these issues, the European Union has been developing the decentralized Digital ID Wallet initiative. This initiative functions like a physical wallet, allowing users to store multiple identities in one place at user premises and giving them control over their data, including the ability to decide what information to share, with whom, and when. While the concept is promising, one of the major challenges lies in designing and developing the necessary infrastructure, which involves creating secure digital identities, integrating them with existing systems, and ensuring they are accessible to a broad audience. This paper addresses these challenges by providing architectural guidelines and a formal description of common operations in such ecosystems, based on our own experiences in developing MVPs.

Keywords: Web 3.0 · Decentralized Identity · Verification · Platform

1 Introduction

Ever since Peter Steiner's famous cartoon from 1993 proclaimed, 'On the Internet, nobody knows you're a dog,' the Internet has evolved and revolutionized our lives by offering various services. Most of the modern services rely on various authentication methods, including usernames, passwords, or more secure options like digital or physical IDs. Existing solutions for digital identity verification are not only unsatisfactory for both users and service providers [1, 2, 8], but also with numerous ID providers available, users struggle to manage their ID data effectively.

However, emerging technologies such as blockchain and VC offer a new concept, explored by entities like the European Commission, of a digital ID wallet application [14]. This application would securely store all IDs, ensuring users

have full ownership of their personal data without centralized collection. Similar to a physical wallet, the digital ID wallet would consolidate multiple IDs in one place, granting users complete control over sharing selected data.

Recently, a new paradigm, self-sovereign identity (SSI), for end-users' digital identity management, has gained considerable momentum, likely also owing to blockchain technology's popularity. Although blockchain technology is not strictly needed for SSI, several SSI projects use a blockchain as a publicly shared and immutable registry for trusted organizations [1]. In the case of SSI, users store their identity-related documents in so-called digital wallet apps on their smartphones [2]. Different credentials can be stored and presented in combination through these identity wallets, for instance, a digital ID card, a digital vaccination certificate, and a digital ticket. There are ongoing initiatives (see section 2.9) to realize these concepts however a comprehensive platform to support for such decentralized identity mechanisms is still lacking. Most existing efforts are focused either on backend APIs or wallet solutions, rather than addressing the broader ecosystem. Clear guidelines are necessary to tackle the unique challenges of this space and ensure that all pieces fit together in a puzzle seamlessly. This paper aims to lay the groundwork for realizing this concept by proposing guidelines for architectural components and common necessary operations needed for such ecosystems.

Here's what's coming up: In section 2, we'll examine relevant literature, followed by an overview of our research methodology and experiments conducted in section 3. Section 4 outlines the result, followed by concluding remarks in section 5.

2 Background

This section presents the relevant theoretical concepts and offers a concise overview of the related research.

2.1 Distributed Ledger Technology

Distributed Ledger Technology, DLT, such as blockchain, is a decentralized database managed by multiple nodes, where transactions are recorded as sequence of blocks, B, in a secure, transparent, and immutable manner. Each block contains a set of transactions T_i, a timestamp t_i, and cryptographic hash of the previous block $H(B_{i-1})$. And each transaction consists of an unique id, input data, output data, and transaction signature i.e $T_i \rightarrow (id, input, output, Sign(T_i))$. Then, DLT can be represented as

$$DLT = (B_1, B_2, ...B_n) \rightarrow \{(T_i, t_i, H(B_{i-1})), (T_{i+1}, t_{i+1}, H(Bi)), ...\} \quad (1)$$

In DLT, decisions are made by all the nodes using an agreed consensus mechanism, meaning that a single node cannot solely control the system, preventing

malicious intents towards the system (Moubarak, Chamoun, Filiol, 2020). A decentralized network of nodes also prevents system downtime since a failure in a single node will not prevent the entire system from working, in the same way that one malicious node will not corrupt the network.

2.2 Decentralized identity

Decentralized identities, $DIDs$, are used to uniquely identify subjects. A DID is identified with the format "did:method:method-id", e.g., "did:example:987654321" [2]. Each DID has a DID document (DID_D) that describes it in detail. The DID document contains information like how the owner of the DID can verify their ownership, what kind of services the DID can be utilised and so on. The DID identifier can be extended to a URL with a path (using /), query (using ?) or any combination of these. The URL enables retrieving sections of a DID document and other related resources. The DIDs are stored on verifiable data registries, such as databases or blockchains. These systems need to be able to transmit the required data for generating or resolving a DID document.

2.3 Verifiable credentials and presentations

Verifiable credentials(VC), are a collection of claims, usually from a trusted issuer, regarding one or multiple subjects, which can be verified through cryptographic means [4]. Such claims are given inside the credential subject property of the VC, and can be made regarding anything, e.g., a university degree. Claims are represented as key-value pairs e.g. age:25. Proving the claims is done via the required *proof* property, which can be a JWT along with other types of proofs. This requires information about the issuer such as its public key such that it can be dereferenced to the required information for verification, such as a DID. VCs usually are valid for a defined or limited period of time and are expressed such as access tokens, id tokens and etc. Verifiable credentials can use any format that is able to represent the data structure, such as XML or JSON format.

Verfiable presentations (VP) can be created and combined from one or more VCs. It can be a verifiable subset of data from a VC. E.g., only using the birth date from a VC representing a passport. It can be a verifiable combination of multiple VCs e.g. combining VCs representing job status and financial status when applying for a bank loan. It can also be verifiable data derived from a VC e.g. create a zero-knowledge proof (or selective disclosure) from the subject's birth date in another VC or VP proving that the credential subject is of legal voting age. For proving to the verifier that all VCs used in the VP were issued to the same holder the proof property is recommended. This property also ensures that no unused information from the VCs is exposed to the verifier [4].

VCs and VPs must contain the proof property containing a method for verifying the data. For instance, by using RSA signing, an issuer can send a VC to a wallet with a proof property containing the signed value of the entire VC, the proof method (RSA signing) and the public key used for decrypting the signed value. The wallet can then generate VPs that consists of one or more VCs, that

can be presented to a verifier. The verifier should be able to use the public keys of the issuers contained in the proof property to validate each VC by decrypting the signatures and checking that they equal the content of the VCs [2].

2.4 Authentication

Authentication is the process of verifying the identity (or credentials C_U) of a user or entity(U). It typically involves two main components: credentials provided by the user or en entity and verification(V) is done by an authenticator(A). In our context, the information from user or entity can be in the form VCs, VPs or ZKPs.

$$U \xrightarrow{C_U} A \ni V(C_U) = \text{true}|\text{false} \tag{2}$$

2.5 Cryptographic Hashes

A hash function is used to construct a short fingerprint of some data. That means if the data is altered, then the fingerprint will (with high probability) no longer be valid A cryptographic hash function is one-way deterministic function where the same input it will always result the same output, and different inputs should give different outputs. Given a hash, one should not be able to calculate the input. Hash functions are used in blockchains or DLTs to generate the cryptographic hash of a block's data, which will be used in the next block to create a back-link (Stinson Paterson, 2018).

$$H_{data} = H(data) \ni H(data_1) \neq H(data_2) \quad when \quad data_1 \neq data_2 \tag{3}$$

2.6 Temper-proof

A feature of a distributed network is that it prohibits tampering, as every node in the network has a copy of the blockchain. If a malicious node tries to change or append a malicious block to the blockchain, other non-malicious nodes will not validate or accept the change rendering the attempt useless [3]. That means if any data in block B_{i-1} is tampered, then the block hash H_{i-1} will change, causing H_i to change, and so on, invalidating the entire chain.

2.7 Layer-2 solutions

Layer 2 solutions and parachains are designed to address the scalability, efficiency, and interoperability issues faced by base blockchains. Parachains are introduced in the Polkadot blockchain and can be seen as an independent chains connected to a central "relay chain". Each parachain can have its own specific use case and logic while benefiting from the relay chain's robust consensus mechanism, enabling efficient cross-chain communication and transaction throughput [5]. While in layer-2 solutions, especially in ethereum ecosystems, transactions are processed off-chain and then batch them on the main chain, reducing congestion and lowering transaction fees.

2.8 Smart contract

A smart contract is a predefined and automatically executable script stored on a blockchain, as Decentralised applications (DApps). When a smart contract has been deployed on a blockchain, it can be triggered by sending a transaction to the contract's on-chain address. A triggered contract will execute likewise on every node in the network, the difference being the output which may vary depending on the data that was sent with the transaction. A byproduct of being stored on a blockchain, is that smart contracts can never be removed, likewise for their content. This means that any actions towards an on-chain smart contract cannot be reverted. Therefore, sending sensitive data to the smart contract's storage might be dangerous if there is no "delete" function predefined in the contract [6].

2.9 Related works

Soltani et.al, in [16], presented a novel SSI-based eKYC onboarding design and evaluated their solution against Allen's principles of SSI. Feulner et. al, in [11], on the other hand presented an architecture on one very specific technology stack, namely Hyperledger. They emphasize the degrees of freedom in blockchain-based SSI from a technical perspective, such as what data needs to be stored on a blockchain, also regarding nascent standards that are being actively developed by the World Wide Web Consortium (W3C), and take this degree of freedom into discussions with experts. Liu et. al identified 12 design patterns for blockchain-based SSI [10], addressing key management, decentralized identifier management, and credential design. Yet, SSI-based solutions do not necessarily need to be based on blockchain. For instance, Nauta et al introduced the IRMA project, representing a solution that implements the principles of SSI without using a blockchain in its technology stack [17]. In contrast to existing approaches, we propose design guidelines for a complete, comprehensive, and generic SSI ecosystem that can be applied with or without the use of blockchain.

3 Methodology

Our research methodology draws inspiration from agile principles and design science research, focusing on tackling one problem at a time and iterating in small steps. This approach prioritizes flexibility, adaptability, and continuous improvement, enabling us to effectively address changing requirements and uncertainties. By integrating experimentation, hypothesis testing, and data-driven decision-making, the conducted approach fosters curiosity, exploration, and learning. This allows for continuous prototyping, and shorter user feedback loops to iterate, refine solutions and drive innovation. We've undertaken a series of experimental projects, each distinct in nature some of them are described below.

Academic Diploma Verification Platform: In this experient [12], simple decentralized app is developed resembling a Web 3.0 wallet! It is designed to revolutionize academic credential storage and transparent verification. Users can

import verified ID documents, such as academic diplomas, into a web wallet and share them with potential employers during recruitment. While the technology functions well, we've encountered challenges. The primary issue is the performance of storing and retrieving large documents, along with associated gas fees for blockchain transactions. Additionally, storing documents transparently in a decentralized network somewhat limits data ownership aspects, despite the preservation of document integrity by Web 3.0 technology.

Identity Wallet Platform: In this experiment [9, 13], we have focused on the whole ecosystem for a cutting-edge identity platform, to transcend the boundaries of conventional digital identity based authentication methods. The idea behind was based on self-sovereign identity (SSI) where user identity verification (e.g BankID, Google) is done once, stored securely in the wallet in the form of verified credential (VC) and later can be used by the user for verification in the form of verifiable presentations (VP). This not only offers ownership of identity data back to the users, but also has large economical advantages along with increased privacy and security. The platform included both mobile and web-based interfaces, and seamlessly integrated web 3.0 into a sophisticated Web 2.0 backend infrastructure. With this fusion of traditional web 2.0 functionalities with the immutable, trustless architecture of web 3.0 technology, we have drawn some Web 3.0. architectural guidelines and design principles ensuring seamless interoperability and enhanced security. The guidelines are comprehensively presented later section in this paper.

4 Result

This section presents best practices for designing decentralized and Web 3.0 based identity verification platform. These guidelines provide developers a road map for building resilient, future-ready applications in decentralized technologies. The results are divided into two main categories: the architectural structure (section 4.2), and the operations (section 4.3 to 4.12). The key results are summarised using a scenario in section 4.1.

4.1 Illustrating scenario

When the wallet app starts for the first time, a primary DID and corresponding DID document is created using the process described in section 4.3, with Ed25519 as authentication method and empty set of service endpoints. At this point, user can verify the ownership of this DID only by using a public-key signature mechanism, Ed25519.

In order to have wider acceptance and usage of DID, an user can link the primary DID to the identities from existing identity providers such as Google, BankID, etc. When the user goes through Google, BankID (or other OAuth2, OpenID based) authentication, information like *id_ token*, *access_ token* and *refresh_ token* are returned. While the *id_ token* verifies the user's identity, the *access_ token* manages access to server-protected resources, and the *refresh_ token*

allows for seamless re-authentication without requiring the user to log in repeatedly. In order to linkup the user information from these ID providers, we need a VC. However, at the time of writing, these ID providers does not issue user information a format of verified credentials (VC). Therefore, we implemented a VC issuer on our own VC issuer and call it *issuer agent*, in section 4.2, that not only issues VCs but also helps integration with existing ID providers.

When VC's are linked with primary DID, the corresponding DID document gets updated, especially the authentication methods, as described in section 4.3. Foe example, after successful user authentication with Google and BankID, the updated DID document will have authentication methods as given in equation (8). The DID owner can then verify the identity ownership using Ed25519, Google, and BankID authentication methods. That means that user DID can be used with any services that allow user authentication with these auth methods. Similarly, the user can add other identities to the wallet, such as scan an machine readable passport, drivers licence etc and link them to the primary DID.

Depending on user preferences or contexts, an user can have multiple DIDs, one primary and rest secondary DIDs. For example an user can have university account, and a job account as secondary DIDs. In such cases, the secondary DIDs shall be linked the primary one according to the linking mechanisms given in section 4.10. Similarly, the user can also add personal identities, such as passport, university diplomas, driving licences and so on as verified credentials, to the wallet and link them to a DID of proper context.

When the user has primary (and optionally secondary) DIDs in place, and are linked to various user identities in terms of verifiable credentials. The user can use them to login to and access other services that support decentralized identities, according to the mechanisms presented in section 4.7. While using such services, the user presents identities to the verifier services in terms of verifiable presentations, as described in in section 4.6. When the user logs in and uses a particular service successfully, the corresponding services associated with DID are updated in DID document, as represented in equation (9).

4.2 Architectural structure

The entire ecosystem consists of 3 main components, the wallet, the backend orchestration server and the *DLT*.

The wallet is a software application, both mobile and web application, that allows users to store variety of cryptographic data and credentials, including DIDs, DID documents, VCs, VPs, authentication tokens, and other cryptographic keys and metadata [15]. The wallet is essential to ensures secure, private, and verifiable digital identity management and interactions with various centralized and decentralized systems.

$$wallet = \{DID, DID_D, SK_U, PK_U, VC_I, VP, encryptionKeys, hashValues, nonces, seedPhrases, backupData, metadata : \{trustedEntities, connectionInfo, revocationInfo, preferences\}\} \quad (4)$$

Where, $SK_u \rightarrow GeneratePrivateKey()$, are private (or secure) keys for the users, $PK_u \rightarrow DerivePublicKey(SK_u)$ are the public keys for the users and are derived from corresponding private keys. The $encryptionKeys$ are used for data privacy and security. The $seedPhrases$ are used to back up and restore cryptographic keys. The $nonces$ ensure uniqueness and prevent replay attacks. The $hashValues$ has various usages in wallets including data integrity, anonymity, deduplication, etc. The $trustedEntities, connectionInfo, revocationInfo$ contain information about trusted entities like identity providers, issuers, registries, etc. The $preferences$ include variety of settings and configurations that the user can customize for example default identity and settings for privacy, security, backup, recovery, notifications and so on.

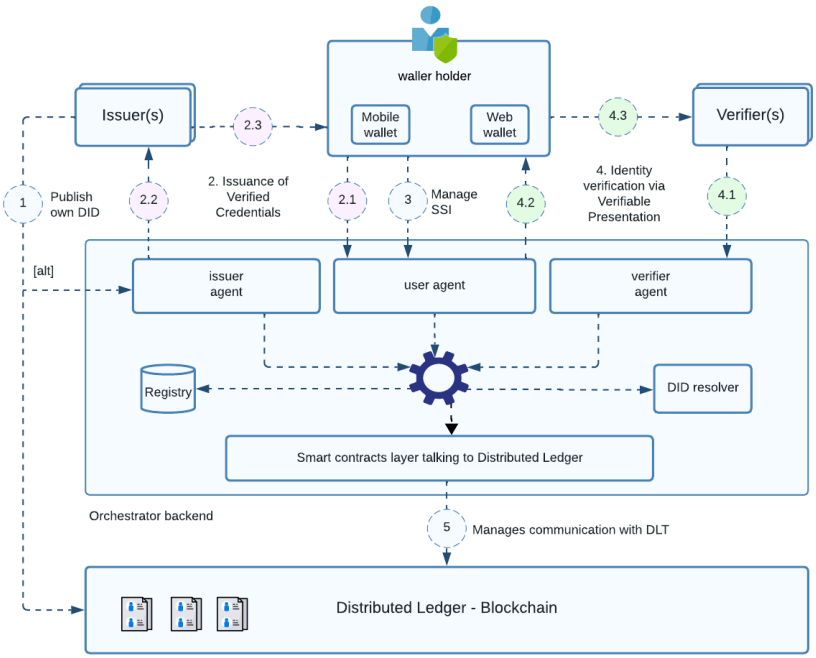

Fig. 1. Architectural components for a decentralized identity verification platform.

The backend orchestration server (*os*) plays several critical roles to support the overall system's functionality and security. Ideally in a fully decentralized system, such centralized backend servers should not be needed. But state of the art is far from reality. Therefore we need such backend servers that provide essential services to facilitate decentralized operations, enhance usability, and ensure compliance with standards. Our implementation of backend server consists of the following components.

- *The user agent (ua)* is probably the most important component as it is the interface for user interactions, especially when the user uses web wallet. It helps users initiate requests, helps with handling cryptographic operations, facilitating user data storage and sharing.
- *The issuer agent* plays a critical role in the issuance and management of VCs, ensuring that the credentials are correctly created, signed, and delivered to the user or their digital wallet. In our ecosystem this component is actually used as VC issuer, and uses users and credentials information retrieved after the authentication process.
- *The verifier agent* helps users especially assisting the authentication with the existing identity providers such as BankID, google and so on. This is required as most the the identity providers need, for security reasons, a backend component during identity verification or authentication process. This is even more relevant when user uses a web based wallet.
- *The registry* is vital when identities are used across multiple decentralised systems and services. This indexes public part of identities and related metadata, and also offers functionalities like registration, resolution, revocation of identities and related credentials.
- *The DID resolver* is a common component used by several other components and mainly offers functionality to resolve the DID into into the associated DID Document, which contains the necessary metadata and cryptographic material required to interact with the DID owner.
- *The DLT proxy* is also a common middleware component that facilitates communication between backend systems and a DLT network like blockchain. This component abstracts the complexity of interacting with the DLT, providing a simplified and secure interface for operations like registering, updating, and querying DIDs, as well as managing Verifiable Credentials (VCs).

The DLT plays a central role offering functionalities such as immutable storage for identity and related metadata, and therefore providing transparency and trust through it. It also acts a verifiable data registry. These functionalities are achieved via self-executing smart contracts deployed into the DLT.

4.3 Creation of DID and DID document

DID the unique ID of an entity and is represented as the encoded hash of the its public key,

$$DID_u = Encode(H(PK_u)) \tag{5}$$

DID document DID_D, contains additional information about an associated DID and contains set of associated public keys PK_u, authentication methods A and services S.

$$DID_D \longrightarrow \{DID_D^U = DID_u, PK_u, A, S\} \tag{6}$$

Where A represents authentication methods that a subject or the entity identified in *DID* can prove the control over it. If an authentication method is *Digital signature based*,

$$A = \{type : Ed25519, publicKey : PK_u\} \tag{7}$$

If, in addition, a DID supports OAuth2.0 and BankID authentication methods, it can be represented as,

$$\begin{aligned} A = \{&\{type : Ed25519, publicKey : PK_u\}, \\ &\{type : OAuth2, AuthorizationEndpoint : A_e, \\ &TokenEndpoint : Te, ClientID : C_id, RedicectURI : R_u\}, \\ &\{type : BankID, AuthorizationEndpoint : A_e, TokenEndpoint : Te, \\ &ClientID : C_id, ClientSecret : C_s, RedicectURI : R_u\}\} \end{aligned} \tag{8}$$

Where S represents service(s) associated with DID providing contexts and capabilities. For example, it can contain identity verification service to verify identity of an DID, it can contain a payment server where an DID can be used for payment, it can list a messaging service to send messages to the given DID.

$$S = \{id, type, serviceEndpoint, description?\} \tag{9}$$

4.4 Representation of verifiable credentials

Verifiable credentials contain information about issuer I, the subject S, set of claims about the subject C and the proof itself P. The issuer information contains its identity id_I and public keys PK_I. The subject information contains identities (for example DIDs) id_S and set of attributes about the subject A. A claim C_i is basically a key-value pair $k_i \rightarrow v_i$. The proof contains the type of the proof t_P, generated timestamp c_P, methods for verification of the proof m_P, and the signature value $\sigma_P \rightarrow Sign_I(H(C))$.

$$VC = \{I, S, C, P\} \rightarrow \{(id_I, PK_I), (id_S, A), (K, V), (type_p, time_p, m_p, \sigma_p)\} \tag{10}$$

Note that verifiable credentials are not part of DID directly but DIDs are referred from them in the subject. VCs are stored in users wallet, side-by-side with DID documents! Verifiable credentials are usually valid for a limited time period.

Ideally, the issuer of the VCs is the service itself. In cases where it is not supported, the issuer could be could be a trusted third party as well!

4.5 Issuance of VC after OAuth2.0 authetication with Google and BankID

After successful authentication, we get *idToken* and *accessToken* from Google. The *idToken* contains *userInfo* among other things. Then VC can be issues from the *userInfo* among other information.

$$idToken, accessToken... = AuthWithGoogle(c_{id}, c_{secret}, redirectUri) \tag{11}$$

$$claims = ExtractClaims(idToken) \tag{12}$$

$$userInfo = \{email, name, authProvider, authTime\} \in claims \tag{13}$$

$$\begin{aligned} VC = \{&type["OAuth2, Google"], issuer = DID_I, \\ &subject = \{id = DID_S, email = userInfo.email, \\ &name = userInfo.name, authProvider = userInfo.authProvider, \\ &authTime = userInfo.authTime, accessToken = accessToken\}, \\ &proof = Sign(VC, privateKey_I)\} \end{aligned} \tag{14}$$

4.6 Representation of verifiable presentations

Verifiable Presentations (VP) are used to present Verifiable Credentials to verifiers in a secure manner. There are at least three ways of doing such. First, the verifiable credentials from the issuer VC_I can be presented directly. Second, selective claims from one or more issuers $VC_I^{selective}$ can be presented. Lastly, only the proof can be presented that demonstrate the possession of certain information without revelaing the actual information itself. Such proof is also called zero-knowledge-proof (ZKP) represented as $\pi = Prove(predicate, witness)$, where *predicate* is the statement about the claims that the prover wants to prove, and *witness* is the actual data that satisfies the predicate which is not revealed. So, VP can be represented as below.

$$VP = \{VC_I, VC_I^{selective}, \pi, proof_{_{VP}}\} \quad (15)$$

$$VC_I^{selective} = \{issuer : DID_I, subject = \{id = DID_S, selectiveClaims_I\} \quad (16)$$

$$proof = Sign(VC, privateKey_I)\} \quad (17)$$

4.7 Verifying a verifiable presentation

Verification of a VP is basically verifying digital signature of the given VP. That means verification of all the the VC , $VC_I^{selective}$ and ZKP (π).

$$\forall VC_i \wedge VC_i^{selective} \in VP, \quad Verify_{PK_I}(Proof, H(Claims)) = true \wedge Verify(\pi) = true \quad (18)$$

Verifying VC and $VC_I^{selective}$ is straightforward, as it is the verification of the signature using issuers public key PK_I. However, the verification of ZKP (π) is a bit tricky as it requires multiple steps.

- First, the prover sends a commitment (hash of the secret information or witness ω, for example date-of-birth in a age verification scenario) to the verifier, $commitment = H(\omega, nonce)$
- The verifier generates a challenge based on the commitment, $c = Challenge(commitment, nonce)$
- The prover then generates a response to the challenge. The response $R(x, c, r)$ is not a hash function but a function specific to the ZKP protocol that combines the secret ω, the challenge c, and the random value r.
- The verifier then verifies the response $Verify(commitment, c, response) = true$.

4.8 Operations with verifiable data registry

A Verifiable Data Registry (VDR) in the context of Self-Sovereign Identity (SSI) can be either centralized or decentralized, depending on the requirements, design and implementation decisions! Centralized VDR are more succeptible to attacks and faces issues like trust, scalabiity and single point of failure. Decentralized VDR are transparent and resilience to attacks, however they are performance issues like slower transaction speeds. The VDR, in any case, offers the functionalities like registration and resolution of DIDs and their public keys, keep track of issued and revoked credentials, verify the authenticity and integrity of credentials and so on.

$$tx_{id} = VDR.register^{tx}(DID_D) \quad (19)$$

$$DID_D = VDR.resolve(DID) \quad (20)$$

$$publicKey = VDR.getPublicKey(DID) \quad (21)$$

$$status = VDR.getCredentialStatus(VC_{id}) \quad (22)$$

$$isValid = verifySignature(data, signature, publicKey) \quad (23)$$

4.9 DID Update

DID is constant as long as it is active, however DID document is evolutionary. A DID document can change when keys, authentication methods, services, and metadata associated with a DID changes. If DID_D^{δ} is the changes in DID, then updated document is represented as $DID_D^{'}$. When the changes in DID document are to be propagated to the registry, one gets new transaction id. And, depending on the registry implementation, the updated document either will replace the existing one, or keep the history but the latest version should always be used (as in blockchain).

$$DID_D^{'} = DID_D \cup DID_D^{\delta} \tag{24}$$

$$tx_{id}^{'} = VDR.update^{tx}(DID, DID_D^{\delta}, Sign_{PK_U}(DID_D^{\delta})) \tag{25}$$

4.10 DID linking

In some cases, an user can have multiple DIDs, each representing ID for defferent domains or contexts. For example one for job and the one for private usage. In such case, one should be considered as primary DID^P and other successive one as secondary DID^S. Corresponding DID documents are represented as DID_D^P and DID_D^S and defined as following. Note also that when DIDs are linked, individual DIDs are updated. That means, registry has to be informed about it if they are previously registered.

$$LA = \{sub : DID^P, obj : DID^S, Sign_{PK_P}(LA)\} \tag{26}$$

$$DID_D^{P^{'}} \rightarrow DID_D^{'} = DID_D \cup LA \rightarrow \{DID_P, linkedDIDs : [LA]\} \tag{27}$$

$$tx_{id}^{'} = VDR.update^{tx}(DID, DID_D^{P^{'}}, Sign_{PK_P}(LA)) \tag{28}$$

4.11 Identity revocation

A DID or verifiable credential (VC) can be revoked for various grounds. In these cases, these should not be trusted! The revocation of DID usually involves makking or removing from the Verifiable Data Registry (VDR). While revoking a VC involves updating the status of the VC in a revocation registry. Note that revocation commands are critical, and should be checked if are coming for a trusted issuer.

$$revokeDID(DID) \rightarrow VDR.revoke^{tx}(DID, Sign(PK_I)) \tag{29}$$

$$revokeVC(VC_id) \rightarrow VDR.updateStatus(VC_{id}, revoked, Sign(PK_I)) \tag{30}$$

4.12 VC refresh

Refreshing or updating a VC involves reissuing a new VC with updated information! While doing so the old one may or may not be revoked. This process is crucial to maintain the accuracy and reliability of the credentials over time. The process of VC refresh involves re-issuance of new VC and linking it to the existing VC id and its subject. If the registry contains information about the VC, then it needs to be updated there too.

$$newVC = Issuer.issue(subjectDID, newAttributes) \quad (31)$$

$$revokeVC(VC_{id}) \rightarrow VDR.updateCredentialStatus(VC_{id}, "revoked") \quad (32)$$

5 Concluding remarks

This paper outlines key architectural guidelines and essential operations for creating a robust decentralized identity ecosystem. Central to this system is the identity wallet, which securely stores crucial user data, including identities, keys, and hashes. While a centralized backend is not mandatory, it can play a pivotal role in facilitating broader adoption and managing key security concerns.

Agreements between issuers and users are recorded on the blockchain, ensuring transparency and verifiability. However, the system must address potential security vulnerabilities, such as unauthorized parties adding counterfeit agreements. Implementing authorization mechanisms or using digital signatures from registered issuers could significantly enhance security and trust.

Despite the secure storage of IDs and agreements, risks such as tampering with public keys and agreements persist, potentially leading to identity forgery or malicious redirections. A centralized backend server is crucial for managing these risks and ensuring the integrity of the system. Additionally, improvements in protocols and endpoint management are necessary to address security issues related to the ID refresh feature and WebSocket-based VC transfers.

The implementation process revealed challenges with existing tools, such as performance issues with QR code generation and security risks associated with the Veramo library's use of JWTs [7]. Proposed solutions include signing JWTs with a DID's private key and addressing the limitations in cryptographic operations needed for features like Selective Disclosure.

Finally, expanding the system to allow websites and apps to request identification from the Digital ID Wallet presents a promising opportunity for wider adoption, akin to how payment methods like Vipps are used online. This would position this new paradigm as a versatile tool for digital identification across various platforms.

References

1. Sedlmeir, J., Smethurst, R., Rieger, A.: Digital Identities and Verifiable Credentials. Bus Inf Syst Eng 63, 603–613 (2021).

2. Avellaneda, O., Bachmann, A., Barbir, A., Brenan, J., Dingle, P., Duffy, K., Hamilton, M., Reed, D., Sporny, M.: Decentralized Identity: Where Did It Come From and Where Is It Going?. In IEEE Communications Standards Magazine (2019).
3. More, S., Patel, N., Parab, S., Maurya, S.: Blockchain based Tamper Proof Certificates. In the International Conference on Smart Data Intelligence (2021).
4. Verifiable Credentials, https://www.w3.org/TR/vc-data-model-2.0/, Accessed 2024
5. Shirodkar, S., Kulkarni, K., R., Khanjode, S., Kohle, P. D., Patil, P.: Layer 2 Solutions to Improve the Scalability of Blockchain. In: 5th International Conference on Advances in Science and Technology (ICAST 2022).
6. Abuhashim, A., Tan, C.: Smart Contract Designs on Blockchain Applications. In: IEEE Symposium on Computers and Communications (ISCC 2020).
7. Veramo Agent, https://veramo.io/docs, Accessed 2022/05
8. Gabriella L., Taija K., Mengcheng L., Markus H., Antti K., Pekka A.: Towards a trustful digital world: exploring self-sovereign identity ecosystems. ArXiv Labs. https://arxiv.org/abs/2105.15131. 2021
9. Bliudzius, R., Hagen, M.S., Kramer, D.: Decentralized Identity - a mobile wallet and verification platform. Bachelors Thesis. NTNU, Norway (2022)
10. Liu, Y., Lu, Q., Paik, H., Xu, X.: Design Patterns for Blockchain-based Self-Sovereign Identity. In Proceedings of the European Conference on Pattern Languages of Programs 2020, pp. 1–14.
11. Feulner, S., Sedlmeir, J., Schlatt, V. et al.: Exploring the use of self-sovereign identity for event ticketing systems. Electron Markets 32, 1759–1777 (2022).
12. Hustad, J., Fredrik, J.: Evaluation and implementation of Digital Identity Ledger for blockchain systems. Bachelors Thesis. NTNU, Norway (2021)
13. Bendvold, J.F., Allison, M.L., Cardona, P.: TokenTrivia: A multiplayer Web3-based Triviaspill game. Bachelors Project. NTNU, Norway (2023)
14. European Digital Identity, https://ec.europa.eu/digital-building-blocks/sites/display/EUDIGITALIDENTITYWALLET/EU+Digital+Identity+Wallet+Home, Accessed 2024
15. Kersic, V., Vidovic, U., Vrecko, A., Domajnko, M., Turkanovic, M.: Orchestrating Digital Wallets for On- and Off-Chain Decentralized Identity Management. In: IEEE Access, vol. 11, pp. 78135-78151, 2023.
16. Soltani, R., Trang Nguyen, U., An, A.: A New Approach to Client Onboarding Using Self-Sovereign Identity and Distributed Ledger. 2018 IEEE International Conference on Internet of Things (iThings 2018), pp. 1129-1136.
17. Nauta, J., Joosten, R.: Self-Sovereign Identity: A Comparison of IRMA and Sovrin. (2019)