Design and Evaluation of a Self-Sovereign Identity System with a Custodial Wallet

Koshi Ikegawa and Tatsuya Sato

Hitachi, Ltd., Kokubunji, Tokyo, Japan
{koshi.ikegawa.mf,tatsuya.sato.so}@hitachi.com

Abstract. Self-Sovereign Identity (SSI) has gained attention as a paradigm that enables users to manage their digital identities and disclose only the necessary information. Although SSI traditionally focuses on managing digital identities on personal devices, deploying an SSI system aimed at large-scale service provision may require the integration of custodial wallets to accommodate users with varying levels of IT literacy and different age groups. In this paper, we present the design and performance evaluation of an SSI system with a custodial wallet. We also propose a performance modeling approach to estimate the number of agent instances required to meet system throughput requirements. The experimental results show that by increasing the number of Holders from 1 to 4, a throughput improvement of 2.47 times for VC issuance and 5.24 times for VP verification was achieved, reaching 32.4 TPS and 63.4 TPS, respectively. Furthermore, a performance model was created through regression analysis of the experimental results, enabling the estimation of the number of agent instances needed to meet system throughput requirements. This provides a guideline for future system design, allowing for the efficient allocation of resources and the realization of an effective system configuration.

1 Introduction

1.1 Web3 and Blockchain

Web3 is envisioned as the next generation of the internet, aiming to move away from centralized management of services and data. It enables users to assume a more proactive role in managing their digital assets. This concept emerged concurrently with the advent of distributed ledger technology (DLT) and blockchain, which originated with Bitcoin[9] and has become its flagship application. Blockchain enables transactions to be executed among many participants without relying on central authorities such as banks. A fundamental characteristic of DLT is that all transaction data and records are shared across every organization participating in the network and are stored in the ledgers maintained by each. Consequently, because altering any single record would require rewriting every copy of the ledger, the system achieves a high degree of tamper resistance.

Web3 and blockchain technologies emphasize decentralization, transparency, and user-centric governance mechanisms, which align with the nature of Open Source Software (OSS). As a result, many blockchain and DLT foundational technologies are developed as OSS. Ethereum is a representative example, widely used as a platform not only for managing cryptocurrencies but also for implementing decentralized applications (DApps) using smart contracts. Hyperledger Fabric [2] is part of the Linux Foundation Decentralized Trust (LFDT) project and is widely used as a consortium blockchain infrastructure for enterprises.

1.2 Self-Sovereign Identity (SSI)

The advent of Web3 and blockchain technologies has brought SSI into focus as a promising paradigm for digital identity management[1,14]. SSI enables individuals to control their own digital identifiers and to disclose only the attributes they choose, thereby enhancing usability while protecting privacy. The European Union's *General Data Protection Regulation* (GDPR, Regulation (EU) 2016/679)[5] reinforces these goals by mandating data minimization, purpose limitation, and explicit consent, while also granting data subjects powerful rights such as access, portability, and erasure. Any large-scale SSI deployment, especially one that integrates custodial wallets operated by regulated organizations, must therefore embed technical and organizational safeguards that demonstrably satisfy the GDPR. Christopher Allen first articulated the concept of SSI in "The Path to Self-Sovereign Identity," where he proposed ten principles centered on user autonomy. The Sovrin Foundation later expanded these into twelve principles that are now widely regarded as the conceptual foundation of SSI[15]. At the core of SSI is the premise that users can fully control their digital identities. By shifting authority away from centralized databases and back to the individual, SSI reduces reliance on third-party data stores and strengthens both privacy and security. In typical implementations, secure databases or blockchains hold cryptographically verifiable proofs of identity attributes, enabling relying parties to carry out trustworthy verification without requiring the disclosure of unnecessary information. Because of these properties, users can manage their identity data autonomously and share it only with trusted parties. A single digital identity can thus be reused across diverse services and transactions while preserving the confidentiality of personal information.

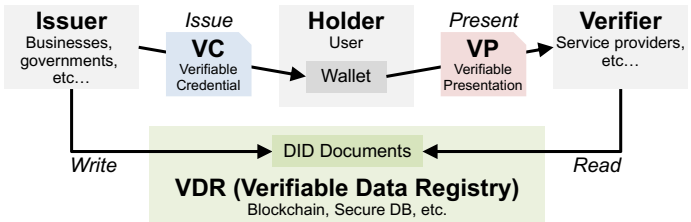

Fig. 1: Overview of Self-Sovereign Identity (SSI).

Fig. 1 shows an overview of the basic components that realize the SSI system. The SSI system is mainly composed of three entities: Issuer, Holder, and Verifier, and a Verifiable Data Registry (VDR). Within this system, Verifiable Credentials (VCs) are issued, and Verifiable Presentations (VPs) are verified. The VDR is used to record DID information related to SSI and schema information for VCs in DID Document format. By using blockchain for the VDR, properties such as tamper-resistance, distribution, and persistence are ensured, enabling trustless and verifiable data management. This makes it highly appropriate as a foundation that technically supports the philosophy of SSI. A detailed description of each element of SSI is shown in Table 1.

1.3 Motivations and Contributions

Based on the principles of SSI, it is recommended that users use a non-custodial wallet that stores digital identities issued on machines they own. However, deploying a large-scale SSI system may require the integration of custodial wallets to accommodate users with varying levels of IT literacy and across different age

Table 1: The elements of a Self-Sovereign Identity (SSI).

Issuer: An Issuer creates and issues VCs. Typically, this role is handled by trusted bodies such as government agencies, educational institutions, or corporations. The Issuer verifies the subject's identity or attributes and signs the VC with its private key to ensure authenticity and integrity.
Holder: A Holder receives, stores, and manages VCs. They keep VCs in a digital wallet and selectively disclose them via VPs when necessary. This selective disclosure allows proof of specific information while preserving privacy.
Verifier: A Verifier requests and evaluates VPs from the Holder. Verification is performed by checking cryptographic signatures and consulting trusted registries, confirming the VC's validity. Based on the results, the Verifier decides on access or service provision.
VC: A Verifiable Credential (VC) is a cryptographically signed digital credential issued by an Issuer to a Holder. It contains claims (e.g., name, age, qualification) and adheres to standards such as those by the W3C. VCs are tamper-evident, machine-verifiable, and interoperable, serving as digital equivalents of paper credentials.
VP: A Verifiable Presentation (VP) is a package of one or more VCs presented by a Holder to a Verifier. It supports selective disclosure, enabling the Holder to reveal only necessary data. In many cases, zero-knowledge proofs (ZKPs) are used to prove facts without disclosing underlying details.
VDR: A Verifiable Data Registry (VDR) is a tamper-resistant storage system, often a blockchain, that hosts metadata (e.g., DIDs, schemas, and credential definitions). It records data in individual ledgers maintained by each organization, allowing Verifiers to confirm the legitimacy of credentials and Issuers.
DID: A Decentralized Identifier (DID) is a unique identifier created, owned, and controlled by the subject of the identity. DIDs enable verifiable SSI and are often used in conjunction with VCs and VPs. DID Documents contain public keys and service endpoints, allowing secure interactions between entities.

groups. Therefore, when providing a large-scale SSI system, it is necessary to have a configuration that allows users to select custodial wallets. On the other hand, discussions regarding the introduction of custodial wallets into SSI systems have not been widely conducted. This research aims to examine the configuration of custodial wallets for a large-scale SSI system and to propose a performance evaluation method. The contributions of our research are listed below.

1. Discussed the need for combining custodial and non-custodial wallets to provide large-scale SSI services and examined their configuration design.
2. Discussed the need for performance evaluation as a challenge of custodial wallets for realizing large-scale SSI services and proposed a performance evaluation method for custodial wallets in SSI systems.
3. Presented a method for estimating the required performance to meet customer requirements by constructing a performance model based on the results of the performance evaluation.

2 Related Work

Many implementations exist for building SSI systems[12], and some are being standardized and developed as Open Source Software (OSS). In particular, Hyperledger Indy[6], Aries[10], and AnonCreds[7], for which specifications and implementations are being promoted by the LFDT and the OpenWallet Foundation (OWF), are widely used as OSS for realizing SSI systems due to their community diversity and development maturity.

2.1 OSS Implementation of SSI System

Indy[6] is a public-permissioned blockchain specifically designed to implement the VDR in SSI systems. Write access is restricted to authorized entities, while reads are public, capturing the "public-permissioned" model that balances public interest and governance. Indy exposes only public metadata needed for issuance and verification while keeping subject attributes off-chain. Specifically, the ledger stores: (i) DID/NYM records to publish identifiers and verification keys, (ii) schemas that define credential attributes, (iii) credential definitions that link issuer keys and cryptographic schemes to schemas, and (iv) revocation registries used to prove current revocation/non-revocation status. Write access is restricted to roles authorized by network governance, while reads are public.

Aries[10] is an open-source framework and specification for building interoperable agents that fulfill the roles of Issuer, Holder, and Verifier in SSI systems. Aries provides standardized protocols and APIs that enable agents to issue, store, and present Verifiable Credentials (VCs) while preserving privacy. At its core, Aries employs DIDComm-based secure asynchronous messaging: peer-to-peer connections are established through out-of-band invitations and DID exchange, and subsequent communication leverages threading, attachments, and delivery over HTTP or WebSocket. This persistent and encrypted channel supports not

only credential issuance and presentation but also messaging and revocation notifications between agents. The project is actively developed by the OWF community in multiple programming languages. Notably, the Python implementation, *A Cloud Agent Python (ACA-Py)*, is widely used in production and research settings and supports both custodial and non-custodial wallet configurations. In addition, the JavaScript implementation, *Credo-ts*, integrates with React Native, enabling cross-platform mobile wallet applications for iOS and Android.

AnonCreds[7] is a privacy-preserving credential specification that defines a VC format along with the associated proof and revocation mechanisms. It enables holders to prove claims about their attributes without disclosing the underlying data. For example, a holder can prove they meet an age requirement without revealing their exact date of birth, and can disclose only selected attributes during presentation. Specifically, AnonCreds employs the CL signature scheme and binds each credential to a *link secret* known only to the holder, preventing impersonation and limiting correlation across presentations. It supports selective disclosure and predicate proofs (e.g., $\geq$, $\leq$, $=$) over integer attributes. Revocation is realized via proofs based on cryptographic accumulators that demonstrate nonrevocation, using a published *tails file* and the current accumulator state. Schemas define attribute sets, while credential definitions publish the issuer' s public keys and revocation parameters.

Considering the aspects of OSS implementation, development maturity, healthy community management, and the feasibility of building custodial wallets, we adopted Hyperledger Indy and Aries for implementing our SSI system. Previous benchmarks have focused on individual agents or ledger components, whereas our research analyzes the impact of introducing custodial wallets on the overall system scale, providing a novel perspective.

2.2 Performance Evaluation of Indy and Aries

We present related work on the performance evaluation of Indy and Aries and clarify the positioning of our research. Baniata et al.[3] evaluated the latency of Indy when used as a VDR. They found that in blockchain configurations with 4-node and 8-node setups, the write response time ranged from 1 to 16 seconds and the read response time ranged from 0.01 to 5 seconds, depending on the transaction rate. Dunphy et al.[4] measured over 45,000 transactions on an Indy VDR and pointed out that the transaction processing capacity may decrease as the number of nodes increases. Bastos et al.[8] proposed a framework called "MinIndy" that automates the construction and management of Hyperledger Indy networks, achieving simplified operations while maintaining performance (throughput and latency) comparable to conventional methods. Their performance evaluation demonstrated that all transactions were processed successfully under a load of up to 250 clients, indicating practicality for development and research purposes. Pflanzner et al.[11] conducted latency analyses of SSI applications using Indy and Aries, performing detailed performance evaluations in both Google Cloud and local cloud environments. Their results showed that in the local environment, write latency was up to 85% faster and read latency was

up to 50% faster. Performance evaluations related to OSS implementations of Aries and Indy have been conducted in several studies and OSS implementations. Siqueira et al.[13] performed a multi-user performance evaluation of the Holder using ACA-Py, an implementation of Aries, and demonstrated stable operation with up to 80 users in a cloud environment, indicating that CPU usage is the main bottleneck and providing empirical insights into scalability. In contrast, our research designs an SSI system with a custodial wallet using ACA-Py and verifies the expected performance improvement by multiplexing the Holder in a real machine environment, while also presenting a performance improvement model based on multiplexing.

3 System Design

In this section, we describe the architecture and design considerations for implementing a Self-Sovereign Identity (SSI) system that supports both custodial and non-custodial wallets. We present the overall system structure, discuss the trade-offs between wallet types, and outline the approach for achieving scalability and flexibility in large-scale deployments.

3.1 Custodial vs. Non-custodial Wallets

In designing an internet-scale SSI platform, it is essential to respect the principle of self-sovereignty by providing non-custodial wallets that allow users to manage their digital identities on their own devices. However, the user base of large-scale SSI services, such as those digitizing government-issued passports, driver's licenses, and national identity cards, includes elderly individuals who may not be able to manage their private keys and digital identities on smartphones, as well as minors who require parental consent. It is also expected that some users may simply prefer not to manage their digital identities on personal devices. Therefore, it is necessary to provide custodial wallets managed by service providers on behalf of users, allowing users to choose between custodial and non-custodial options. However, there exists a trade-off between usability and security in this context. Custodial wallets simplify user experience and facilitate fraud detection and regulatory compliance but can become performance bottlenecks due to the aggregation of vast amounts of key information and the need to handle

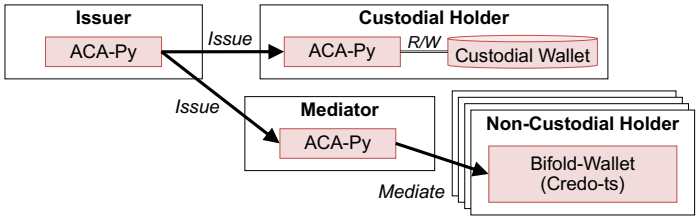

Fig. 2: Aries-based SSI system with selectable wallet.

cryptographic operations for millions of users. Non-custodial wallets eliminate centralized risks but place the burden of key backup, device maintenance, and multi-factor authentication on the users themselves. In this research, we recommend the implementation of a *Selectable Wallet* that provides both custodial and non-custodial options during the construction of an SSI system, allowing users to freely choose their preferred wallet type. Additionally, we propose a method for estimating the required performance to meet customer requirements by conducting performance evaluations of custodial wallets and constructing a performance model for the entire system.

3.2 Selectable Wallet Design

In this research, we designed an SSI system that allows users to choose between custodial and non-custodial wallets using ACA-Py and Bifold-Wallet (Credo-ts), which are OSS implementations of Aries agents. ACA-Py is a server-side agent implementation well-suited for custodial wallet scenarios. Here, the service provider centrally manages users' DIDs and VCs in a secure server-side database (e.g., PostgreSQL), exposing user operations via APIs. This approach enables SSI services for users who may lack IT literacy or have difficulty managing cryptographic keys. However, it requires the service provider to implement strict access controls, auditing, and compliance with regulations such as GDPR, since key management responsibility is centralized. Bifold-Wallet is implemented as a mobile application based on React Native, using the Credo-ts Aries JavaScript implementation. It serves as a non-custodial wallet, allowing users to manage their private keys and VCs locally on their smartphones. This maximizes self-sovereignty and privacy protection, as users retain control over their keys and can perform VC presentation and management entirely within the app. However, it necessitates robust backup and recovery mechanisms in case of device loss, as well as the introduction of multi-factor authentication. As shown in Fig. 2, the service provider offers both wallet types and provides a user interface that allows users to select their preferred wallet during registration or onboarding. For example, users with smartphones can be guided to install the Bifold-Wallet app, while those unable to manage devices can use a web portal backed by ACA-Py as a custodial wallet. This Selectable Wallet design enables flexible and secure SSI services for a wide range of user groups.

3.3 Performance and Scalability Challenges

If the majority of users choose non-custodial wallets, the principles of self-sovereignty and decentralization will be strengthened. Responsibility for key management shifts from service providers to users, reducing security risks. Furthermore, since users manage their keys on their own devices, the computational load on the service provider side is reduced, improving the scalability of the entire system. However, if a large number of users choose custodial wallets, the key management and cryptographic processing concentrated at the service provider could become a bottleneck, potentially degrading overall system performance. If

numerous custodial users simultaneously request VC issuance or VP verification, server-side load surges, response times increase, and the risk of timeouts rises. Therefore, a critical challenge in building large-scale SSI systems is ensuring the performance and scalability of custodial wallets. To address this challenge, we aim to improve performance and scalability by multiplexing Aries agents. Specifically, multiple Aries agents are launched in parallel, enabling each agent to independently handle VC issuance and VP verification. This distributes the processing load on the service provider side, improves response times, and helps avoid timeouts. In this research, it is necessary to verify the effectiveness of performance improvement through multiplexing in a real machine environment. Furthermore, we aim to construct a performance improvement model based on multiplexing to make the overall system performance predictable.

4 Performance Evaluation Method

We present a performance measurement method using agent multiplexing in the SSI system with Aries/Indy. First, we describe the evaluation environment. Next, we outline the preparation steps. Then, we explain how to collect transaction-time logs between Issuer and Holder for VC issuance. Finally, we describe the log collection between Holder and Verifier for VP verification.

4.1 Evaluation Environment

In this study, we designed the system configuration of an SSI system with a custodial wallet to support large-scale service provision. Fig. 3 shows the design of our SSI system. Since this SSI system aims to provide large-scale services, we configured it by launching multiple virtual machines (VMs), each running agents with different roles. Each VM has 12 vCPU cores, 16 GB of memory, and 40 GB of storage capacity, with Ubuntu 24.04 LTS installed as the OS. Additionally, all software was containerized using Docker.

Next, we present the configuration and construction procedure of the SSI system using Aries/Indy. First, we set up a Controller to perform SSI system

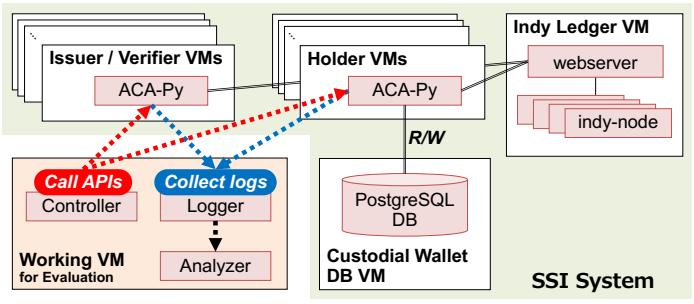

Fig. 3: Evaluation Environment of our SSI System.

construction operations and agent API calls, a Logger for log collection, and an Analyzer for log analysis on the Working VM. Next, we constructed the VDR on the Indy Ledger VM using von-network, an OSS that enables launching a simplified Indy blockchain network and web server. Then, we established the environment for ACA-Py, which is the Python implementation of Aries that operates as Issuer, Holder, and Verifier agents in the SSI system. The performance measurement in this study focuses on VC issuance and VP verification in the custodial wallet configuration of Holder. For VC issuance performance measurement, we measure communication between Issuer and Holder, while for VP verification performance measurement, we measure communication between Holder and Verifier; therefore, these measurements are independent. Thus, in our performance measurement, we launched Issuer and Verifier as the same agent. We refer to the agent that serves as both Issuer and Verifier in this paper as "Issuer/Verifier". We started one ACA-Py for one Issuer/Verifier VM and one Holder VM. Additionally, since it is necessary to multiplex Holder agents while using a single database (DB), we launched a PostgreSQL DB on the Custodial Wallet DB VM and configured all Holder VMs to connect to this DB.

4.2 Preparation

Preparation followed the flow in Fig. 4 and was orchestrated via controller - issued API calls. DIDs for each agent were registered on the Indy VDR, followed by registration of the Schema and credential definition (Cred_def) for the target VC. The Schema comprised ten attributes, each assigned an 8 - digit random integer. A user tenant was then provisioned on the Holder agent, and a connection between the Issuer/Verifier and Holder agents was established; subsequent VC issuance and VP verification referenced the resulting connection IDs. We executed 2,400 parallel operations and collected logs, varying the number of Issuer/Verifier agents from 1 to 4 and the number of Holder agents from 1 to 7.

4.3 Measurement for VC Issuance

We explain the method for collecting logs of transaction processing times between Issuer and Holder for VC issuance performance measurement. Fig. 5 shows the flowchart for performance measurement during VC issuance. First, the Logger that receives events via webhook is started, and log collection begins. Next, a request for VC issuance is made to the Issuer agent. The Issuer issues a VC to the Holder, who receives the VC and stores it in their wallet. This VC issuance and storage process is repeated in parallel, and logs are collected during the process. For this VC issuance, the "issue-credential-2.0/send" API of the Issuer agent is used. When this API is called, internal processes such as sending and receiving the Offer, sending and receiving the Request, sending and receiving the Credential, and completion notification are performed. The execution timestamps of each of these processes are sent to the Logger via event notifications through webhook for log collection.

4.4 Measurement for VP Verification

We explain the method for collecting logs of transaction processing times between Holder and Verifier for VP verification performance measurement. Fig. 6 shows the flowchart for performance measurement during VP verification. It is assumed that a series of VC issuance processes have been completed, and each Holder already has the VC issued by the Issuer stored in its user tenant. First, a webhook server for log collection is started, and log collection begins. Next, a request for VP submission for verification is made from the Verifier to the Holder. In this

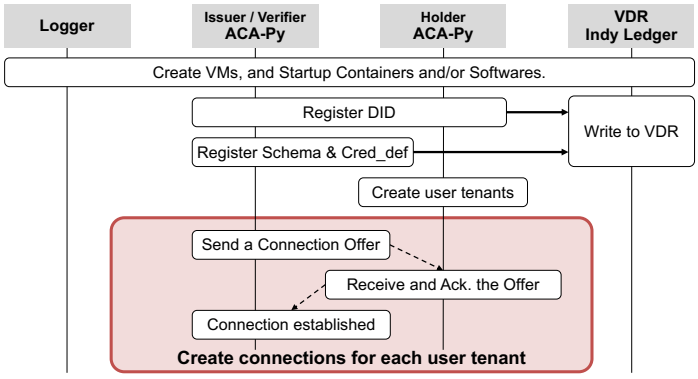

Fig. 4: Preparation for performance measurement.

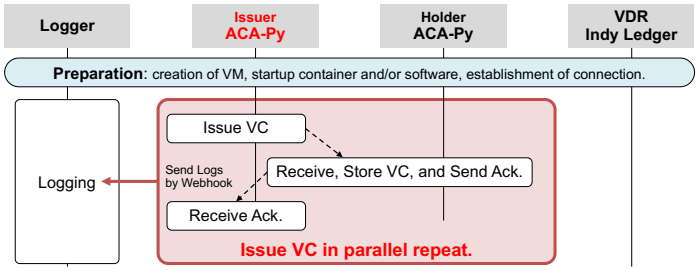

Fig. 5: Performance measurement for VC issuance.

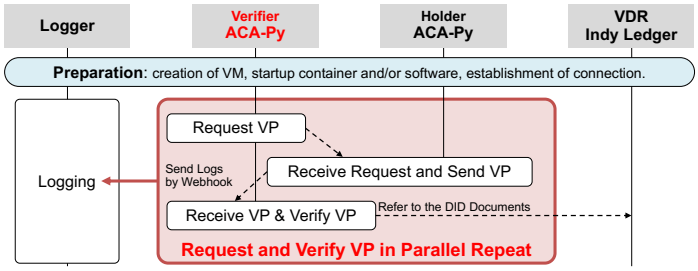

Fig. 6: Performance measurement for VP presentation and verification.

study, a VC with 10 items is used as the definition of Schema and Cred_def, and all 10 items are requested to be presented in the VP submission. At this time, the zero-knowledge proof function for VP presentation is not used; instead, the minimal data disclosure approach is requested. After the Holder submits the VP to the Verifier, the Verifier verifies the VP by referring to the Cred_def stored in the VDR. This VP verification and the notification of verification results are repeated in parallel, and logs are collected during the process. For this VP verification, the present-proof-2.0/send API of the Holder is used. When this API is called, internal processes such as sending and receiving the Request, sending and receiving the Presentation, and completion notification are performed. The execution timestamps of these processes are sent to the webhook server for log collection.

5 Result and Analysis

5.1 Performance Measurement Result

We present the performance measurement results for VC issuance and VP verification based on the collected logs. Additionally, since overheads occur at the beginning and end of the process, we computed the stable throughput by excluding 200 logs (400 logs in total) at the beginning and end.

Result of VC Issuance The performance measurement results during VC issuance are presented. Fig. 7 shows throughput on the vertical axis and the number of Holder agents on the horizontal axis, with four graphs plotted based on the number of Issuer agents. When there is only one Issuer, the throughput remains almost constant even as the number of Holders increases. This indicates that when there is only one Issuer, it has already reached its performance limit. Next, as the number of Issuers increases to 2 and 3, the point at which the

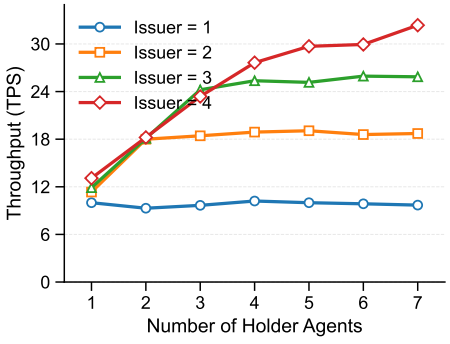

Fig. 7: Result of performance measurement for VC issuance.

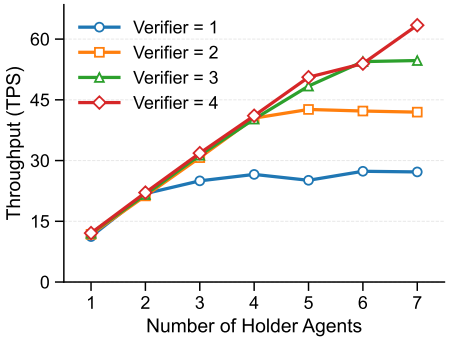

Fig. 8: Result of performance measurement for VP verification.

throughput levels off shifts to the right as the number of Holders increases. Additionally, when there are four Issuers, the throughput increases almost linearly with the number of Holders up to four. By increasing the number of Holders from 1 to 4, a throughput improvement of 2.47 times was achieved, reaching 32.4 TPS. These observations indicate that both Issuer and Holder have inherent performance limits in VC issuance processing; however, performance can be improved through multiplexing. Moreover, the graphs show that when there are four Issuers and four Holders, throughput continues to increase linearly without reaching a limit. Therefore, performance modeling was performed using data for up to four Issuer agents and four Holder agents.

Result of VP Verification The performance measurement results during VP verification are presented. Fig. 8 shows throughput on the vertical axis and the number of Holder agents on the horizontal axis, with four graphs plotted according to the number of Verifier agents. When there is only one Verifier, the throughput remains almost constant even as the number of Holders increases beyond three. This indicates that when there is only one Verifier, even if additional Holders are added beyond three, the Holder side has already reached its performance limit. Next, as the number of Verifiers increases to 2 and then 3, the point at which the throughput plateaus shifts to the right with increasing Holder numbers. Additionally, when there are four Verifiers, the throughput increases almost linearly with the number of Holders up to seven. By increasing the number of Holders from 1 to 4, a throughput improvement of 5.24 times was achieved, reaching 63.4 TPS. These observations indicate that in VP verification processing, both Holder and Verifier have inherent performance limits; however, multiplexing each can improve performance. Furthermore, the graphs show that when there are three Verifiers and seven Holders, throughput increases linearly without reaching performance limits. Therefore, performance modeling was performed using data for up to three Verifier agents and seven Holder agents.

5.2 Performance Modeling

Based on the performance measurement results, we present the performance models for VC issuance and VP verification. Let x be the number of multiplexed Issuer/Verifier agents, and let y be the throughput during VC issuance and VP verification, respectively. By substituting the required throughput into y in these equations and solving for x according to customer performance requirements, it becomes possible to estimate the necessary multiplexing count.

Regression analysis was performed on the performance measurement results during VC issuance to create linear and quadratic models. Let x be the number of multiplexed Issuer agents and y be the throughput. The performance models are expressed by the following equations. The regression equation for the linear model during VC issuance is given by Eq. (1), and for the quadratic model, it is given by Eq. (2).

$$y = 4.883x + 8.377 \tag{1}$$

$$y = -0.220x^2 + 5.980x + 7.279 \tag{2}$$

Similarly, regression analysis was performed on the performance measurement results during VP verification to create linear and quadratic models. Let x be the number of multiplexed Verifier agents and y be the throughput. The performance models are expressed by the following equations. The regression equation for the linear model during VP verification is given by Eq. (3), and for the quadratic model, it is given by Eq. (4).

$$y = 7.515x + 2.870 \tag{3}$$

$$y = -0.631x^2 + 10.669x - 0.284 \tag{4}$$

By using these performance models, it becomes possible to estimate the number of multiplexed agents required to meet the throughput requirements during VC issuance and VP verification when constructing the SSI system.

6 Conclusions

This study presented an architecture that introduces a custodial wallet into an SSI system, together with a workload-oriented performance evaluation method and an analytical model for capacity planning. By multiplexing Issuer/Verifier and Holder agents, the proposed design achieved higher throughput and demonstrated its suitability for large-scale service delivery. The current evaluation used fewer than ten agents. Future work will extend the experiments to tens or hundreds of agents so that the regression model can be validated under heavier loads and refined where needed. In addition, our prototype allocates one agent per virtual machine. Placing multiple agents on a single machine and orchestrating them with Kubernetes or a similar platform would allow automatic scaling and better utilization of hardware resources. Production traffic is rarely constant; multiplexing is most valuable during peak demand. A controller that adjusts the number of active agents in real time, based on user requests, would reduce operational cost without sacrificing responsiveness. Detailed simulations or field trials in a multi-tenant environment will clarify how dynamic allocation affects latency, isolation, and fault tolerance. The custodial approach offers stronger key protection and simpler user experience, yet it raises questions about privacy and regulatory compliance. Future research should explore these trade-offs in depth, assess potential attack surfaces, and propose mitigation measures that satisfy standards such as GDPR. By lowering the entry barrier for end users, our custodial-enabled SSI architecture supports inclusive e-government and fintech services, and it provides a practical path toward mass adoption of Web3 technologies. We expect that further studies on large-scale agent deployment, adaptive resource management, and privacy safeguards will lead to even greater gains in reliability and performance.

References

1. Allen, C.: The Path to Self-Sovereign Identity (2016), https://www.lifewithalacrity.com/article/the-path-to-self-soverereign-identity, Last accessed 15 Sep 2025
2. Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukolić, M., Cocco, S.W., Yellick, J.: Hyperledger Fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the thirteenth EuroSys conference. Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3190508.3190538
3. Baniata, H., Pflanzner, T., Feher, Z., Kertesz, A.: Latency assessment of blockchain-based SSI applications utilizing Hyperledger Indy. In: Proceedings of the 12th International Conference on Cloud Computing and Services Science (CLOSER). pp. 264–271. INSTICC, SciTePress (2022). https://doi.org/10.5220/0011082300003200
4. Dunphy, P.: A note on the blockchain trilemma for decentralized identity: Learning from experiments with Hyperledger Indy (2022). https://doi.org/10.48550/arXiv.2204.05784
5. European Parliament, Council of the European Union: Regulation (EU) 2016/679 of the European Parliament and of the Council, https://eur-lex.europa.eu/eli/reg/2016/679, Last accessed 15 Sep 2025
6. Linux Foundation Decentralized Trust: Hyperledger Indy (2017), https://www.hyperledger.org/projects/hyperledger-indy, Last accessed 15 Sep 2025
7. Linux Foundation Decentralized Trust: AnonCreds Specification (2022), https://hyperledger.github.io/anoncreds-spec/, Last accessed 15 Sep 2025
8. M. Bastos et al.: MinIndy: Automating the deployment and management of Hyperledger Indy networks. In: 2024 IEEE Latin-American Conference on Communications (LATINCOM). pp. 1–6 (2024). https://doi.org/10.1109/LATINCOM62985.2024.10770675
9. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (March 2009), https://bitcoin.org/bitcoin.pdf
10. Open Wallet Foundation: A Cloud Agent Python (ACA-Py) (2019), https://aca-py.org/latest/, Last accessed 15 Sep 2025
11. Pflanzner, T., Baniata, H., Kertesz, A.: Latency analysis of blockchain-based SSI applications. Future Internet **14**(10) (2022). https://doi.org/10.3390/fi14100282
12. Satybaldy, A., Nowostawski, M., Ellingsen, J.: Self-Sovereign Identity Systems, pp. 447–461. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-42504-3_28
13. Siqueira, A., Da Conceição, A.F., Rocha, V.: Performance evaluation of self-sovereign identity use cases. In: 2023 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS). pp. 135–144 (2023). https://doi.org/10.1109/DAPPS57946.2023.00026
14. Soltani, R., Nguyen, U.T., An, A.: A survey of self-sovereign identity ecosystem. Security and Communication Networks **2021**(8873429), 1–26 (2021). https://doi.org/10.1155/2021/8873429
15. Sovrin Foundation: "Principles of SSI v3" (2022), https://sovrin.org/principles-of-ssi/, Last accessed 15 Sep 2025