# Granting Access Privileges Using OpenID Connect in Permissioned Distributed Ledgers

Shohei Kakei[1(✉)], Yoshiaki Shiraishi[2], and Shoichi Saito[1]

[1] Nagoya Institute of Technology, Nagoya, Aichi, Japan
kakei.shohei@nitech.ac.jp
[2] Kobe University, Kobe, Hyogo, Japan

**Abstract.** Permissioned distributed ledger technology (DLT), in which only authenticated entities participate, assumes trust among the participants and implicit consent for data manipulation. In light of international regulations such as the GDPR, it is necessary to clarify the access privileges of user data, even for systems that assume the trust of the participants. In this paper, we propose an access privilege granting method for service providers that need to access user data in permissioned DLT systems. The proposed method separates the access privilege for user data in the distributed ledger from the execution privilege for smart contracts. By requesting a user to grant the access privilege, the participants can manipulate user data using smart contracts. The access privilege is represented by a token issued by OpenID Connect (OIDC). Smart contracts can directly verify the token without the participant's interference. In this way, all the participants in the DLT network can reach a consensus that data manipulation is based on the user's consent. We implemented the prototype system with Keycloak, an OIDC-compliant identity provider, and Hyperledger Fabric, a permissioned DLT, and then evaluated its performance. Finally, the overhead of access control is 0.21%, from which we conclude that the load on the system is very small.

**Keywords:** Distributed ledger technology · Smart contract · Access control · OpenID Connect · Hyperledger Fabric

## 1 Introduction

Distributed Ledger Technology (DLT) is known as a technology that can eliminate the cost of siloed business processes. In particular, a permissioned DLT system, which is built using multiple known nodes, fits the business use cases requiring the collaboration of multiple organizations because the permissioned DLT system can restrict the organizations that can join a DLT network [7]. In terms of the managed data handled by a DLT system, they can be classified into two main types: user data, such as healthcare [6,26], education [8] or energy [9], and non-user data, such as logistics [4] or the vehicular ad hoc networks [17].


F. Li et al. (Eds.): SecureComm 2022, LNICST 462, pp. 290–308, 2023.
https://doi.org/10.1007/978-3-031-25538-0_16

A smart contract (SC), which is used as an interface to manipulate data in a distributed ledger, enables secure and flexible data manipulation based on consensus among the organizations. In a permissioned DLT system, the execution of an SC is implicitly agreed upon because of the assumption of trust among participating organizations. In order to guarantee users the right to restrict access to user data and the right to data portability as required by regulations such as the General Data Protection Regulation (GDPR) [23], it is necessary to clarify access privileges to data in a distributed ledger.

Employing the classic method of sharing credentials such as passwords and passcodes for delegating access to resources often leads to unauthorized access and misuse of the provided credentials. Sudarsan *et al.* [21] classify the delegation-based authorization models into three types: (i) identity delegation at the authentication level, (ii) delegation by access control/authorization server, and (iii) power-of-attorney (PoA) based authorization, and discuss their strength and weakness. According to the authors, delegation by access control/authorization server and PoA-based authorization are similar in allowing authorization on the user's behalf. On the contrary, the difference is that delegation by access control/authorization server is performed via an authorization server, while PoA [20] is performed by the user's direct signature. In PoA-based authorization, since the user directly uses the private key, there is less dependence on third parties, but it is often hard to use private keys flexibly according to the environment. On the other hand, delegation by access control/authorization server requires trust in the authorization server but allows delegation from various devices through authentication via a web browser. Furthermore, security can be improved by multi-factor authentication [13].

In this paper, we propose an access privilege granting method for service providers to access user data in a permissioned DLT system. The proposed method separates the access privilege for user data in the distributed ledger from the execution privilege for SCs. A data owner gives consent for the service provider to access data, and the service provider manipulates the data under the consent. In other words, data owners need to eliminate unauthorized intervention by service providers and ensure that their consent is correctly transmitted to the DLT system. Our approach uses JSON Web Token (JWT) mechanism to transmit a claim as the user's consent. The JWT has a digital signature proving its claim and is often employed in distributed architectures such as microservices [5,10,19]. In our approach, the JWT provides the user's consent to the service provider, and the service provider uses the JWT to prove access privileges to the DLT system. By doing this, a data owner does not have to implicitly trust the service providers for data manipulation and can explicitly allow the service providers to manipulate data.

The main contributions of the paper are summarized as follows:

- We advocate separating data access privileges from SC execution privileges and propose an access control method based on user consent in permissioned DLTs.

- We designed the proposed method using JWT. In this method, JWT securely informs the SC about what a user is consenting to. For example, the user can consent to read or write data.
- We implement a prototype system for the proposed method with Hyperledger Fabric and evaluate the processing overhead of the proposed access control.
- We present limitations from the perspective of misuse of the privileges through security analysis and discuss the challenges in applying the proposed method to permissionless DLTs.

The rest of the paper is organized as follows. Section 2 presents background of the proposed method. Section 3 shows the access control model of the proposed method and provides the security requirements in access control using JWT. The proposed method is presented in Sect. 4, and is shown to satisfy the security requirements in Sect. 5. We present a prototype implementation and evaluate the performance in Sect. 6. Section 7 discusses the proposed method, and Sect. 8 concludes this paper.

## 2 Background and Related Works

### 2.1 OpenID Connect

OpenID Connect (OIDC) [18] is a mechanism that extends the OAuth 2.0 authorization protocol with the ability to issue ID tokens containing user information for user authentication. In OIDC, an ID Provider (IdP) issues an authorization code to an end-user (EU), and the EU passes the authorization code to a Relying Party (RP). The RP authenticates the EU by exchanging the authorization code for an ID token and verifying it. By porting RP's authentication function to the IdP, the EU does not need to authenticate individual RPs.

IdPs can issue an ID Token as a JSON Web Token (JWT) [14] consisting of a header part, a payload part, and a signature part. The header part typically consists of the type of the token and the signature algorithm being used. The payload part contains arbitrary attribute information called claims (e.g., issuer, subject, and expiration date). The signature part contains the signature of the IdP on the data in the header and payload parts.

In this way, a JWT can carry user information whose authenticity is guaranteed by an IdP. JWTs are suitable for carrying user authentication in distributed architectures because they can be verified locally and directly through the public key [19]. Karim *et al.* [15] raised a concern regarding rich user authentication in resource-constrained IoT environments. They proposed an authentication model based on OIDC for the IoT manufacturer platform, which will enable users to maintain IoT devices. Their model offloads the user authentication process to an IoT gateway. The gateway performs rule-based user authentication with JWTs. Xu *et al.* [24] raised a concern regarding account management in edge computing. They proposed a microservice security agent platform that enables edge computing clients to access the edge computing service with JWT.

### 2.2 Distributed Ledger Technology

Distributed ledger technology (DLT) is a technology that synchronizes tamper-resistant distributed ledgers using a consensus algorithm [1,2]. The distributed ledger is managed by multiple nodes that construct the DLT network and is divided into two components: a data store (DS), which stores data, such as an user, application, and system data, and a smart contract (SC), which is an interface for manipulating data in a DS. Only agreed-upon data are stored in the DS through the mutual confirmation of the execution results of the SC by multiple nodes.

DLT systems can be classified into "permissionless" and "permissioned" systems based on the participation of nodes [16]. Permissionless DLT systems, consisting of a DLT network with an unspecified number of nodes, are fault-tolerant because they do not require a central administrator as a single point of trust. However, even in the case of faults in SCs, a permissionless DLT system cannot be temporarily suspended. Thus, it is hard to patch vulnerable SCs [11,27]. This allows damage to spread easily. On the other hand, permissioned DLT systems comprise a consortium consisting of only specific organizations. Thus, a permissioned DLT system can be suspended for maintenance and to prevent further damage at the consortium's discretion, but trust in the consortium is vital. Ethereum and Bitcoin are known as permissionless DLT systems, and Hyperledger Fabric and Corda are known as permissioned DLT systems.

Researchers indicate that DLT is improving the GDPR compliance for user data sharing. Antwi *et al.* [6] identified key requirements of healthcare applications and created testing scenarios using Hyperledger Fabric to investigate the potential for the GDPR compliance with healthcare applications. Delgado-von-Eitzen *et al.* [8] proposed a GDPR-compliant academic certification system based on Hyperledger Fabric. Systems oriented toward GDPR compliance tend to employ permissioned DLT because it can restrict participants from manipulating data depending on user attributes. In addition to user data, there are also studies that share information about organizational data, such as cyber attacks [12]. Truong *et al.* [22] proposed a design concept with technical mechanisms for a blockchain-based GDPR-compliant personal data management platform. The platform manages the data with a database and ports the mechanisms (e.g., authentication and authorization, access control, and logging) to a blockchain network.

## 3 Access Control and Security Requirements for Distributed Ledgers

### 3.1 Access Control for Distributed Ledgers

Models of DLT-based services can be broadly classified into two types: one in which the DLT system operates at the front end, as shown in Fig. 1-(a), and the other in which the DLT system operates at the back end, as shown in Fig. 1-(b). The former model type is suitable for a permissionless DLT system that

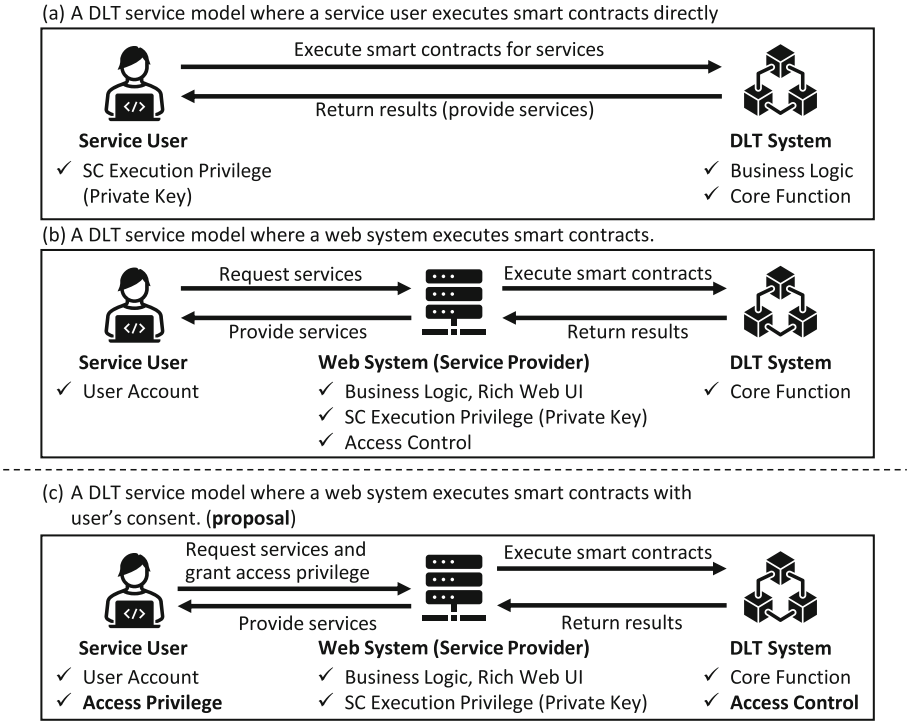


**Fig. 1.** Comparison of DLT service models and proposed DLT service model.

does not require a central administrator since the service user can execute SCs directly. There are two drawbacks of the permissionless DLT: one is the difficulty of implementing complex processing such as business logic or rich user interfaces due to its non-stoppable nature and the limitation of SCs. The other is that it forces all users to manage their private keys at their own risk. In contrast to the former model, in the latter model, core functions and complex processing can be separated into the DLT system and the web server, respectively. Thus, this model type is ideal for commercial services. The latter model is suitable for a permissioned DLT system that can restrict the entities executing SCs. The service user entrusts SC execution to the service provider (SP). The SP authenticates the service user, manipulates user data using SCs, and provides services using SC processing results. This model can achieve a rich DLT-based service and reduce the private key management cost for the service user, although the need for implicit trust in the SP arises.

When constructing a system using DLT, it is required to consider the characteristics of both. The systems oriented toward GDPR compliance described in Sect. 2.2 can be classified into the model shown in Fig. 1-(a). In this model, it is inevitable that usability will be sacrificed, although data manipulation is guaranteed to originate from the service user unless the private key is compromised.

For usability improvement, a new DLT service model is needed to resolve the problem with the model shown in Fig. 1-(b). In light of the above, this paper proposes the new DLT service model shown in Fig. 1-(c), which reduces the need for implicit trust in the SP.

Figure 1-(c) shows a new DLT service model that explicitly manages the access privilege for user data. In Fig. 1-(b), implicit trust arises because the SP controls data access as a single trust point. The proposed method moves the access control point from the SP to the DLT system, and the service user grants the access privilege to the SP. The SP has the SC execution privilege and executes SCs using the SC execution privilege and the access privilege.

In Fig. 1-(b), it is difficult to verify the legitimacy of manipulating data of users who are not in a service provision relationship because the SP is allowed to execute SCs that include the access privilege of user data. For example, when the service user provides user data with the DLT system, the service user expects that only the SP in use can access their data. However, it is difficult to guarantee such expectations in an environment where the access privilege is assigned to multiple organizations, including competitors. On the other hand, in the proposed model, the SP cannot access user data unless the service user grants the access privilege to the SP. When access to user data is required, the SP requests the service user to grant the access privilege. The access control point embedded in the SC determines whether data access is allowed or not based on the access privilege. The data manipulation is explicitly performed because multiple participants of the DLT network verify the access privilege.

### 3.2 Security Requirements for Access Control with JWT

In the model focusing on, the service user grants the SC the access privilege via the SP. For access privileges to be correctly conveyed from the user to the SC, the SC must be prevented from being interfered with and each node executing the SC must be able to correctly verify access privileges. In distributed architectures such as microservices, there are some mechanisms to directly authenticate user-related information asynchronously with the authentication server using the statelessness of JWT [5,10,19]. Therefore, the proposed model uses JWT, which can express a variety of claims, as access privileges.

The entity managing an access privilege can issue a JWT that includes the information necessary to verify access privileges as claims, and thereby the SC can know the access privilege of the service user simply by verifying the JWT. However, if the SP misuses the JWT, the access privileges will be violated. In order to mitigate the risk of misusing a JWT, the following requirements are required.

- **Req. 1:** The SC must only accept a JWT issued to legitimate a service user to use the legitimate service.
- **Req. 2:** The SC cannot use a JWT that has been used before.
- **Req. 3:** The SC cannot use a JWT that has been issued in the past.

Req. 1 is a requirement to prevent the use of access privileges of others or issued for other services. Req. 2 and Req. 3 are requirements to guarantee that the SP's use of access privileges originates from the SU's consent. If the SP could reuse a JWT, it would not be possible to guarantee that the execution of the SC originates from the SU. For the same reason, the act of retaining an unused JWT for later use by the SP must also be prevented.

## 4 Proposed Method

This paper proposes an access control method for user data in a distributed ledger. The proposed method guarantees that a service user accesses the user data based on user consent. An access token, which is a JWT in OIDC, represents a user's consent and is used as a temporary access privilege for the user data. A service user grants a service provider the access privilege by the access token. SC verifies the access token so that the user's consent is agreed upon among the participants of the DLT network.

### 4.1 Structure of the Proposed Method

The proposed method contains the following six components. The relationships with each component are shown in Fig. 2.

- **ID Provider (IdP):** The IdP is a component that issues ID tokens and access tokens in JWT format under the OIDC protocol. The IdP manages user accounts of SUs and issues authorization codes in response to SU requests. The authorization codes are exchanged by SPs to obtain ID tokens and access tokens. The IdP issues these tokens in the authorization code flow.
- **Service User (SU):** The SU is a service user of SPs and owns the rights to the SU's data in the DS. By granting the access privilege to the SP, the SU agrees to allow the SP to access the SU's data. In the proposed method, the access privilege is represented by the access token. The SU is the EU in OIDC.
- **Service Provider (SP):** The SP is a component that joins the DLT network according to the permission of the DLSP and provides services to the SU using SCs. Joining the DLT network gives the SP the privilege to execute the SCs. The SP trusts the IdP and authenticates the SU with the ID token issued from the IdP when providing services. If access to SU's data is required when providing services, the SP requests the access token from the SU and executes SCs with the access token.
- **Smart Contract (SC):** The SC is a part of a distributed ledger and is a set of code scripts that can process data in the DS. The SC can restrict the data manipulation by its code, and all SPs agree to the processing written in the SC. The processing of verifying access tokens is embedded in the SC, and the SC determines whether or not SU's data can be read or written according to access tokens.

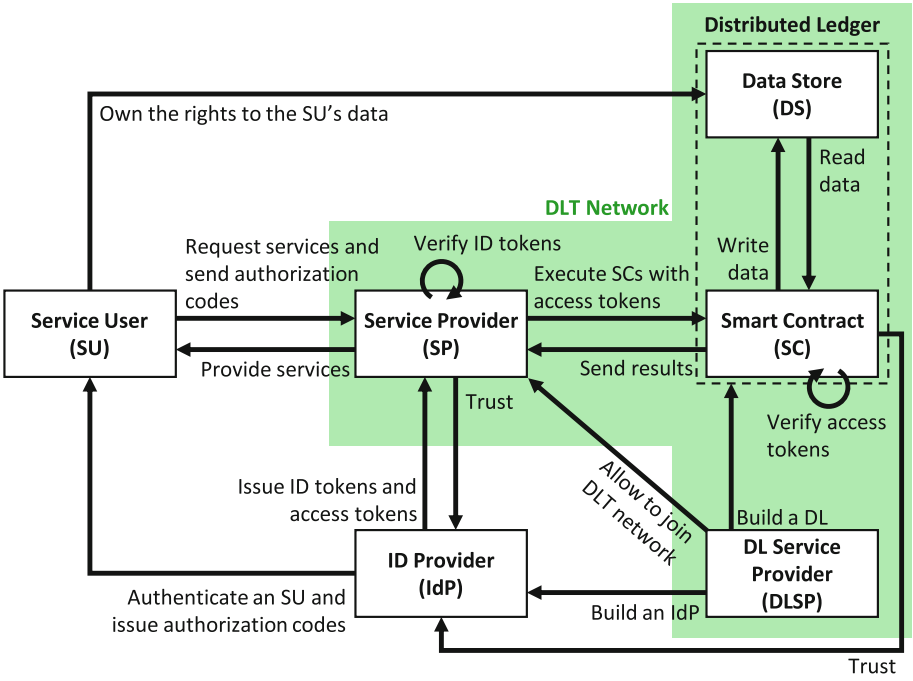


**Fig. 2.** Relationships with components of the proposed method.

- **Data Store (DS):** The DS is a part of a distributed ledger and can store any data. The DS is comprised of multiple instances of ledgers, which are maintained by each SP. In the proposed method, the DS stores application data (e.g., user data and configuration data) and system data (e.g., the identities of entities, the status of the access privileges, and public key certificates). The SU owns the rights to the SU's data, although the DS is managed under the DLT network.
- **Distributed Ledger Service Provider (DLSP):** The DLSP is a consortium of multiple organizations responsible for operating and managing a permissioned DLT network. Moreover, the DLSP is responsible for the DS's design, the SC's implementation, the management of participating SPs in the DLT network, and operates the IdP. The DLSP has to define access privileges according to a service design and implements the verification logic of a JWT in the SC.

### 4.2 Definition of Access Token and ID Token

The proposed method uses two types of JWTs signed by IdP: access tokens and ID tokens. The seven types of claims contained in a JWT that are used to construct the proposed method are listed below.

- **iss (Issuer):** The iss claim represents the issuing entity of the JWT. It is an identifier that represents the IdP in the proposed method. An IdP identifier is assigned to the IdP in advance by the DLSP.
- **aud (Audience):** The aud claim represents an RP to which the JWT is issued. It is an identifier that represents the SC in the proposed method. The SC identifier is configured to the IdP and SC in advance by the DLSP.
- **sub (Subject):** The sub claim represents the requesting entity of the JWT. It is an identifier that represents SUs in the proposed method. The IdP assigns a unique identifier to each SU when the IdP performs a user registration.
- **azp (Authorized Party):** The azp claim represents the target of JWT issuance. It is an identifier that represents SPs in the proposed method. It is determined when SPs are registered with the IdP.
- **iat (Issued At)** and **exp (Expiration Time):** These claims represent the validity period of the JWT. The iat claim is the date and time the JWT was issued, and the exp claim is the expiration date of the JWT. Both claims represent the validity period of access privileges in the proposed method.
- **scope:** The scope claim represents the range of access to data requested by RP. It represents the access privileges required by an SP in the proposed method. The available access privileges are configured in advance to the IdP by the DLSP.

### 4.3 Processing Flow

This section describes the four phases of the proposed method: setup phase, SP registration phase, SU registration phase, and service provision phase. In the setup phase, the DLSP launches the distributed ledger service by constructing the DS, the SC, and the IdP. In the SP Registration Phase, the DLSP qualifies an SP to participate in the DLT network and registers the SP as the RP with the IdP. In the SU registration phase, the DLSP registers an SU as a user of the distributed ledger service for the IdP. Finally, in the service provision phase, the SP provides services to the SU using SP's SC execution privilege and SU's DS access privilege. It is important to note that the DLSP has prepared the IdP, the SC, and the DS. The DLSP implements the SC to store these information types in the DS and stores the iss, aud claims, and IdP's public key $\mathtt{pk}_{\mathtt{IdP}}$ in the DS. Moreover, the DLSP has defined scopes and has set the scopes to the IdP.

**Setup Phase.** Figure 3 shows the processing flow of this phase. The DLSP generates the iss and aud claims and $\mathtt{pk}_{\mathtt{IdP}}$ and calls the SC to register these information types (Step 1-1). The SC writes these information types in the DS (Step 1-2). After this phase is performed for the first time, it is repeated each time the registration information is updated, such as following maintenance of the DLT network.

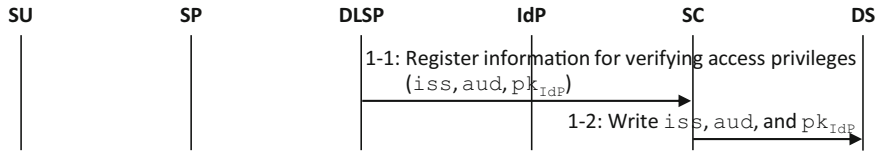


**Fig. 3.** Processing of the setup phase.

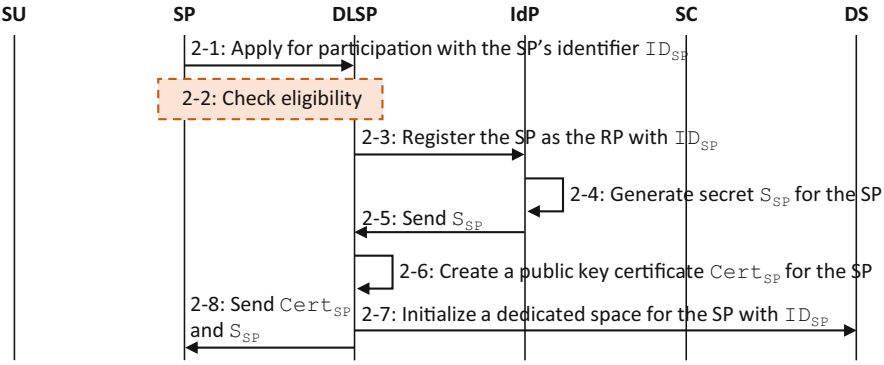


**Fig. 4.** SP registration phase.

**SP Registration Phase.** Figure 4 shows the processing flow of this phase. An SP applies to the DLSP for participation with its identifier $\mathtt{ID}_{\mathtt{SP}}$ (Step 2-1). When receiving the application, the DLSP checks whether the SP meets the criteria for participation in the DLT network (Step 2-2), and if it does, the DLSP registers the SP with the IdP as the RP with $\mathtt{ID}_{\mathtt{SP}}$ (Step 2-3). The IdP generates a secret $\mathtt{S}_{\mathtt{SP}}$ for the SP after confirming that $\mathtt{ID}_{\mathtt{SP}}$ is not already registered and issues $\mathtt{S}_{\mathtt{SP}}$ to the DLSP (Steps 2-4, 2-5). The DLSP creates a public key certificate $\mathtt{Cert}_{\mathtt{SP}}$, a qualification to participate in the DLT network, and initializes a dedicated space in the DS for the SP (Steps 2-6, 2-7). Finally, the DLSP issues $\mathtt{Cert}_{\mathtt{SP}}$ and $\mathtt{S}_{\mathtt{SP}}$ to the SP (Step 2-8).

The criteria for participating in the DLT network are beyond the scope of this paper. However, the DLSP can set its criteria, such as verifying the legal existence of the organization. The public key certificate and the secret generated in this phase are used to execute an SC and request access privileges in the service provision phase.

**SU Registration Phase.** Figure 5 shows the processing flow of this phase. When receiving a user registration from an SU, the DLSP determines whether or not the SU meets the criteria for using the distributed ledger (Steps 3-1, 3-2). If it does, the DLSP creates an account for the SU in the IdP (Step 3-3), and the IdP issues an SU's identifier $\mathtt{ID}_{\mathtt{SU}}$ (Steps 3-4, 3-5). Next, the DLSP creates the SU's account on the DS using the SC (Step 3-6), and the SC initializes settings, including the status of the access privilege (Step 3-7). Finally, the DLSP

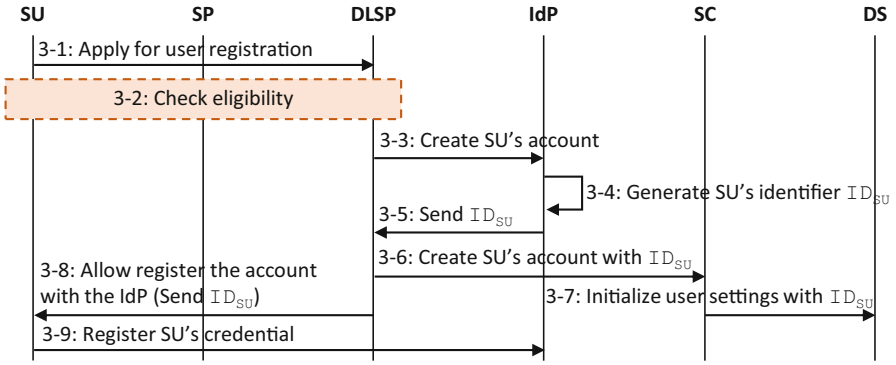


**Fig. 5.** SU registration phase.

allows the SU to register an account with the IdP (Step 3-7), and the SU sets authentication information in its IdP account (Step 3-8).

The criteria for using the distributed ledger are beyond the scope of this paper. However, the DLSP can set its criteria, such as confirming possession of a valid email address [3]. The details will be described at the service provision phase, but the date and time information of access tokens is stored to detect misuse. Then, in Step 3-7, the SC initializes that date and time information.

**Service Provision Phase.** Figure 6 shows the processing flow of this phase. When receiving a service request from an SU (Step 4-1), the SP decides on an SC to be executed (Step 4-2) and requests the necessary privileges to the SU using OIDC (Step 4-3). The SU, the SP, and the IdP execute the authorization code flow; the SU gives the SP an authorization code, and the SP obtains an ID token and an access token through the exchange of the authorization code. After the exchange of the tokens, the SP authenticates the SU using the sub claim of the ID token (Step 4-4) and executes the SC with arguments and the access token (Step 4-5).

The SC consists of the three processing: initiating the data manipulation (Step 4-6), executing the processing with DS read/write (Step 4-7), and finalizing the data manipulation (Step 4-8). In Step 4-6, the SC checks the validity of access privileges by checking the access token. The validation of the access privileges is performed as follows. Let $\mathtt{AT}_\mathtt{t}$ denote the access token at time $\mathtt{t}$, and $\mathtt{iat}_\mathtt{t}$ denote the iat claim contained in $\mathtt{AT}_\mathtt{t}$. First, the integrity of $\mathtt{AT}_\mathtt{t}$ is verified to confirm that a legitimate IdP signs $\mathtt{AT}_\mathtt{t}$.

**Step V1:** The SC verifies the signature part of the $\mathtt{AT}_\mathtt{t}$ using the $\mathtt{pk}_{\mathtt{IdP}}$ registered in the DS.

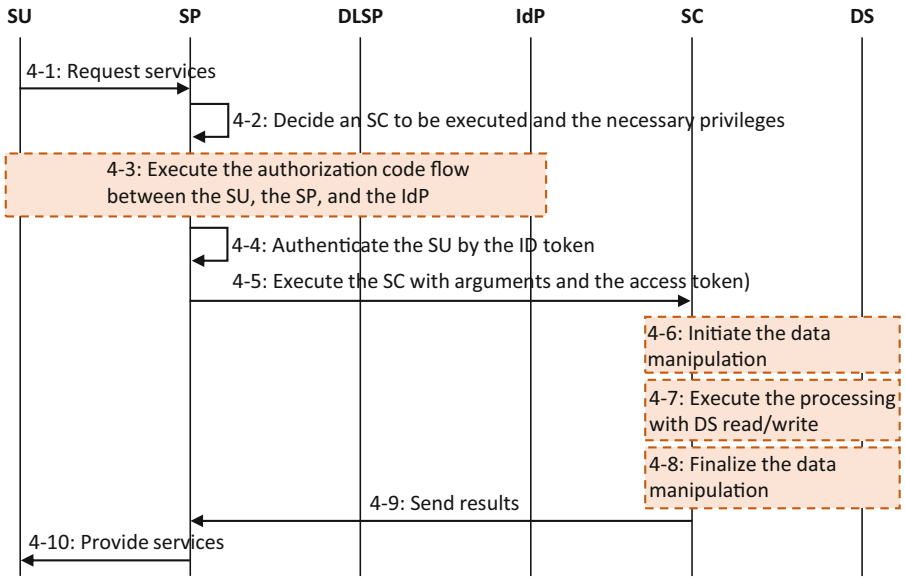


**Fig. 6.** Service provision phase.

Next, it is verified that the $\texttt{AT}_\texttt{t}$ is not reused.

**Step V2:** The SC compares $\texttt{iat}_\texttt{t}$ with $\texttt{iat}_{\texttt{t-1}}$ stored in the DS and verifies whether $\texttt{iat}_\texttt{t}$ is greater than $\texttt{iat}_{\texttt{t-1}}$, which means that $\texttt{AT}_\texttt{t}$ is the newer access token than the last used access token.

Next, it is verified that the legitimate IdP issues $\texttt{AT}_\texttt{t}$ to the legitimate SC, and the legitimate SP uses $\texttt{AT}_\texttt{t}$.

**Step V3:** The SC checks the executor of the SC with $\texttt{Cert}_{\texttt{SP}}$ and confirms that it is the same entity listed in the azp claim of $\texttt{AT}_\texttt{t}$.

**Step V4:** The SC compares the aud claim registered in the DS with the aud claim in $\texttt{AT}_\texttt{t}$ to confirm that the recipient of $\texttt{AT}_\texttt{t}$ is the SC itself.

**Step V5:** The SC compares the iss claim registered in the DS with the iss claim in $\texttt{AT}_\texttt{t}$ to confirm that the issuer of $\texttt{AT}_\texttt{t}$ is the legitimate IdP.

Finally, it is checked that $\texttt{AT}_\texttt{t}$ contains the necessary privileges for data manipulation.

**Step V6:** The SC verifies that the sub claim and the owner of the data to be manipulated match.

**Step V7:** Using the scope claims, the SC verifies that the access privileges meet the privileges required by the processing.

If any verification fails, the processing is aborted, and an error is returned to the SP. Next, in Step 4-7, the SC executes the processing for the DLT service and

reads/writes the SU's data. After the processing, the SC overwrites $\texttt{iat}_{\texttt{t-1}}$ with $\texttt{iat}_{\texttt{t}}$ (Step 4-8). Finally, the SC returns the results of the processing (Step 4-9), and the SP provides services using the results (Step 4-10).

## 5 Evaluating the Security Requirements of the Proposed Method

We evaluated that the processing flow designed in Sect. 4.3 satisfies the three security requirements defined in Sect. 3.2.

**Evaluation of Req. 1.** Req. 1 is the requirement to ensure that an SP and an SU are not impersonated. There are two possible methods of impersonation: rewriting the information in the azp claim and the sub claim or theft of a JWT containing the information to be impersonated.

Rewriting the claim information can be detected by the integrity verification of a JWT in Step V1. Since the IdP guarantees the claim information, it is secure as long as the signing key used by the IdP to sign the claim is not compromised.

There are two cases of theft of a JWT: theft from another service domain and theft within the same service domain. The former can be detected by comparing the azp claim with an executor of the SC in Step V3 since the executor is identified by $\texttt{Cert}_{\texttt{SP}}$. Such attacks can be prevented if the signing key corresponding to the $\texttt{Cert}_{\texttt{SP}}$ is not compromised. In the latter case, the subject of $\texttt{Cert}_{\texttt{SP}}$ and the azp claim are equal; thus, there is a risk that an SP impersonates an SU. However, the risk can be reduced by Req. 2 and Req. 3, which restrict misuse of an access token.

**Evaluation of Req. 2.** Req. 2 is required to prevent an SP from reusing a previously used access token. During the service provision phase, the date and time information of the iat claim stored in a DS is updated at each time of SC execution. As a result, the reuse of an access token can be detected by Step V2.

One way to prevent the reuse of an access token can be using the jti claim that uniquely identifies a JWT. Since the IdP sets a different string in the jti claim for each JWT, an SP can detect the reuse of the access token if the SP saves the jti claim used once in the distributed ledger. However, the jti claims of the used tokens must be stored in the distributed ledger until the access tokens expire. Hence, the proposed method uses iat claims, which are ordered information.

**Evaluation of Req. 3.** Req. 3 is required to prevent an SP from retaining an unused access token for later use. In the proposed method, the date and time information of the iat claim saved in a DS is updated at each time of SC execution. As a result, even if the SP tries to use the unused access token later, it can be detected by Step V2 since the token retained will be invalid when an SU uses the new access token. To reduce the risk of unauthorized retention by SPs, the validity period of the access token should be kept to the minimum necessary

that does not affect service provision. Specifically, the access token must be valid from Step 4-3 to Step 4-6.

## 6 Performance Evaluation

In the proposed method, the setup phase and the two registration phases are executed in advance, but the service provision phase is executed each time an SU requests a service. Therefore, latency for the service provision phase is a significant part of the overhead of the proposed method. In addition, the proposed method requires additional storage to manage the status of the access privilege. For this reason, we evaluate the processing time overhead and data size overhead during the service provision phase.

### 6.1 Experimental Setup

For this experiment, we implemented a data storage service with a permissioned DLT system as an experimental system and embedded the proposed access control mechanism into the experimental system. The experimental system has one scope `Change` and an SU can access its data with an access token that contains an SU's identity and the LIST scope. The scope regarding data registration is not defined because the SU's data are registered in the DS in advance for the experiment.

Our experiments were conducted on a machine running Linux kernel 4.15.0 on an Intel Core i9-9940X with 32 GB of main memory, Hyperledger Fabric v2.3.3 as a permissioned DLT system, and Keycloak v15.0.2 as an IdP system.

### 6.2 Experimental Result

All entities are deployed on a single experimental machine. The SU and the SP are implemented as the test script, while the two Docker containers in Hyperledger Fabric run the SC and the DS. We executed the test script for 1000 rounds with the processes from Step 4-1 to Step 4-10 as one round. The authorization code flow is a process that is generally executed in services that use OIDC; thus, we measured two kinds of processing times: Step 4-5 to Step 4-9 and Step 4-6 to Step 4-8, excluding Step 4-7. The former represents the service latency caused by Hyperledger Fabric, while the latter represents the pure overhead of the proposed method. In addition, because Hyperledger Fabric keeps records of DLT operations as a series of blocks, we measured the size of that block at the end of each round. We retrieved blocks using the "peer channel fetch" command.

Figure 7 shows trends in processing time in the experiment. The mean and variance of the processing times are $2282.7 \pm 6.9$ ms for requesting the chaincode and $4.8 \pm 0.7$ ms for the access control. The SC processing time is stable, with an overhead of only 0.21%. Figure 8 shows trends in data growth in the experiment. Both data sizes tend to increase monotonically. The mean and variance of data growth in each round are $7.884 \pm 0.002$ KB when the proposed method is applied,

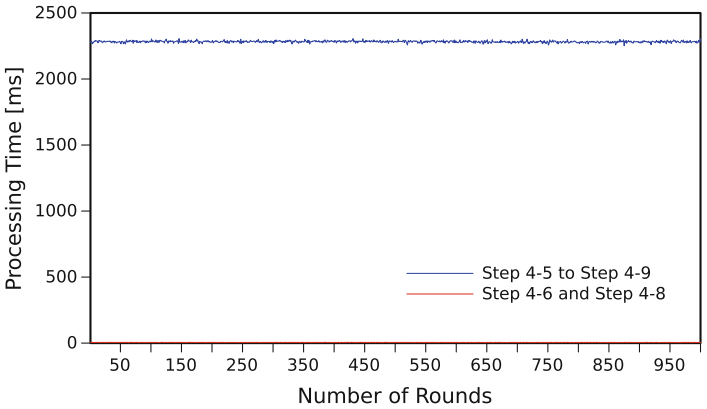


**Fig. 7.** Trends in processing time during the service provision phase. The blue line shows the small variation in the time that users are kept waiting by the smart contract. The red line indicates that the overhead is very small. (Color figure online)

and $6.118 \pm 0.002$ KB when the proposed method is not applied. The additional data size per processing in the proposed method is $1.766 \pm 0.002$ KB. Comparing the result without access control, the proposed method has an overhead 22.40%.

## 7 Discussion

### 7.1 Overhead of the Proposed Method

As shown in the experimental results, the overhead of the processing time is 0.21%, and the overhead of the data size is 22.40%. The result indicates that the proposed method has a negligible impact on the DLT service in terms of the overhead of the processing time. On the other hand, the result indicates that the proposed method requires a certain cost in data storage, but the cost can be estimated from the frequency of SC execution.

In contrast to the cost of the access control, the proposed method requires the cost to execute SCs in an invoke method. In Hyperledger Fabric, the invoke method executes a consensus algorithm, and it is known as a more time-consuming process than the query method that does not require executing the consensus algorithm. We found that the proposed method is well-suited for DLT services that change the state of user data as executed by the invoke method. In contrast, for DLT services that only read user data as executed by the query method, the proposed method bears the additional costs of executing the consensus algorithm.

### 7.2 Limitation of the Proposed Method

As shown in Step V2, the updated date and time of the iat claim causes all access tokens prior to that date and time to be determined as past. However,

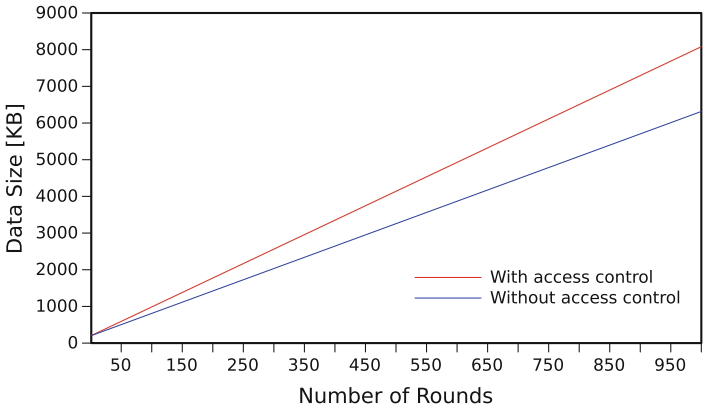


**Fig. 8.** Trends in data growth during the service provision phase. The graph shows that the data size increases monotonically with the number of SC executions with (red line) and without (blue line) the proposed method. (Color figure online)

the SP can obtain the latest unused access token by returning an error to the SU before the SP performs Step 4-5. Here, the SP can use the access token until it expires or the same SU requests the service. In order to detect such fraud, it is considered necessary to devise an auditing method that compares logs of IdPs, SCs, and SPs.

The work in [25] points out that processing with system clock should be done with caution because Hyperledger Fabric has non-deterministic risk. Our method uses system clock to verify the validity period of the access token. However, there is no problem except when executing a SC at the time of expiration since it is only used to compare access token expiration dates.

### 7.3 Access Privileges in Permissionless DLT and Importance of Separating Access Privileges in Permissioned DLT

When a permissionless DLT system is deployed as the front end, an SU operates the Web UI provided by an SP and executes an SC using its secret key. In other words, the SP does not execute the SC on behalf of the SU but supports the execution of the SC by the SU. Since the SC is executed mainly by the SU, access privileges are explicit, but the SU is responsible for managing its private key.

In Ethereum, a permissionless DLT system, private keys are associated with accounts, so a private key breach is equivalent to losing all assets associated with the account. On the other hand, in the proposed method, the SU does not need to manage private keys since it only needs to manage the credentials of IdP. In IdP authentication, not only password authentication but also FIDO authentication can be used, which is expected to improve user convenience. The IdP credentials are used to control granting access privileges to SPs, and credential

breach is equivalent to losing that control. However, since the DLSP manages the access privileges itself, the DLSP can regain control by reconfiguring the authentication information of the SU. By separating the access privileges to the data in the distributed ledger from the execution privileges of SC, the proposed method can reduce the burden of managing confidential information required by permissionless DLT systems while achieving a clear operation of the access privilege.

## 8 Conclusion

In this paper, we propose a method that separates an access privilege for user data in a distributed ledger from a privilege to execute smart contracts and controls access to the data with the user's consent. In the proposed method, the access privilege is represented by an access token in the form of JWT in OpenID Connect, and security requirements for its secure operation are defined. We implemented the experimental system of the proposed method using Hyperledger Fabric and Keycloak and evaluated the performance of the proposed method.

Future works include designing and developing the proposed method to support any smart contract application and devising a method to audit the proposed method to reduce the risk of fraud in a service provider.

**Acknowledgement.** This work was supported by JSPS KAKENHI Grant Number JP22K17881. This research results were partly obtained from the commissioned research under a contract of "Research and development on IoT malware removal/make it non-functional technologies for effective use of the radio spectrum" among "Research and Development for Expansion of Radio Wave Resources (JPJ000254)", which was supported by the Ministry of Internal Affairs and Communications, Japan.

## References

1. Blockchain and distributed ledger technologies – Vocabulary. ISO 22739:2020 (2020)
2. Blockchain and distributed ledger technologies – Taxonomy and Ontology. ISO/TS 23258:2021 (2021)
3. How to prove and verify someone's identity (2022). https://www.gov.uk/government/publications/identity-proofing-and-verification-of-an-individual/how-to-prove-and-verify-someones-identity. Accessed 26 June 2022
4. IBM food trust: a new era in the world's food supply (2022). https://www.ibm.com/in-en/blockchain/solutions/food-trust. Accessed 26 June 2022
5. de Almeida, M.G., Canedo, E.D.: Authentication and authorization in microservices architecture: a systematic literature review. Appl. Sci. **12**(6) (2022). https://doi.org/10.3390/app12063023, https://www.mdpi.com/2076-3417/12/6/3023
6. Antwi, M., Adnane, A., Ahmad, F., Hussain, R., Habib UR Rehman, M., Kerrache, C.A.: The case of HyperLedger fabric as a blockchain solution for healthcare applications. Blockchain Res. Appl. **2**(1), 100012 (2021). https://doi.org/10.1016/j.bcra.2021.100012

7. Bedin, A.R.C., Capretz, M., Mir, S.: Blockchain for collaborative businesses. Mob. Netw. Appl. **26**(1), 277–284 (2020). https://doi.org/10.1007/s11036-020-01649-6
8. Delgado-von Eitzen, C., Anido-Rifón, L., Fernández-Iglesias, M.J.: Application of blockchain in education: GDPR-compliant and scalable certification and verification of academic information. Appl. Sci. **11**(10) (2021). https://doi.org/10.3390/app11104537
9. Guan, Z., Lu, X., Wang, N., Wu, J., Du, X., Guizani, M.: Towards secure and efficient energy trading in IIoT-enabled energy internet: a blockchain approach. Future Gener. Comput. Syst. **110**, 686–695 (2020). https://doi.org/10.1016/j.future.2019.09.027, https://www.sciencedirect.com/science/article/pii/S0167739X19315018
10. He, X., Yang, X.: Authentication and authorization of end user in microservice architecture. In: Journal of Physics: Conference Series, vol. 910, p. 012060. IOP Publishing (2017)
11. Huang, Y., Bian, Y., Li, R., Zhao, J.L., Shi, P.: Smart contract security: a software lifecycle perspective. IEEE Access **7**, 150184–150202 (2019). https://doi.org/10.1109/ACCESS.2019.2946988
12. Huff, P., Li, Q.: A distributed ledger for non-attributable cyber threat intelligence exchange. In: Garcia-Alfaro, J., Li, S., Poovendran, R., Debar, H., Yung, M. (eds.) SecureComm 2021. LNICST, vol. 398, pp. 164–184. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90019-9_9
13. Ibrokhimov, S., Hui, K.L., Abdulhakim Al-Absi, A., lee, H.J., Sain, M.: Multi-factor authentication in cyber physical system: a state of art survey. In: 2019 21st International Conference on Advanced Communication Technology (ICACT), pp. 279–284 (2019). https://doi.org/10.23919/ICACT.2019.8701960
14. Jones, M., Bradley, J., Sakimura, N.: JSON web token (JWT). RFC 7519 (2015). https://doi.org/10.17487/RFC7519
15. Karim, A., Adnan, M.A.: An OpenID based authentication service mechanisms for internet of things. In: 2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS), pp. 687–692 (2019). https://doi.org/10.1109/CCOMS.2019.8821761
16. Kuhn, R., Yaga, D., Voas, J.: Rethinking distributed ledger technology. Computer **52**(2), 68–72 (2019). https://doi.org/10.1109/MC.2019.2898162
17. Lu, Z., Liu, W., Wang, Q., Qu, G., Liu, Z.: A privacy-preserving trust model based on blockchain for VANETs. IEEE Access **6**, 45655–45664 (2018). https://doi.org/10.1109/ACCESS.2018.2864189
18. Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C.: OpenID Connect Core 1.0 (2014). https://openid.net/specs/openid-connect-core-1_0.html
19. ShuLin, Y., JiePing, H.: Research on unified authentication and authorization in microservice architecture. In: 2020 IEEE 20th International Conference on Communication Technology (ICCT), pp. 1169–1173 (2020). https://doi.org/10.1109/ICCT50939.2020.9295931
20. Sudarsan, S.V., Schelén, O., Bodin, U.: A model for signatories in cyber-physical systems. In: 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), vol. 1, pp. 15–21 (2020). https://doi.org/10.1109/ETFA46521.2020.9212081
21. Sudarsan, S.V., Schelén, O., Bodin, U.: Survey on delegated and self-contained authorization techniques in CPS and IoT. IEEE Access **9**, 98169–98184 (2021). https://doi.org/10.1109/ACCESS.2021.3093327
22. Truong, N.B., Sun, K., Lee, G.M., Guo, Y.: GDPR-compliant personal data management: a blockchain-based solution. IEEE Trans. Inf. Forensics Secur. **15**, 1746–1761 (2020). https://doi.org/10.1109/TIFS.2019.2948287

23. Voigt, P., Von dem Bussche, A.: The EU general data protection regulation. GDPR), A Practical Guide (2017)
24. Xu, R., Jin, W., Kim, D.: Microservice security agent based on API gateway in edge computing. Sensors **19**(22) (2019). https://doi.org/10.3390/s19224905
25. Yamashita, K., Nomura, Y., Zhou, E., Pi, B., Jun, S.: Potential risks of hyperledger fabric smart contracts. In: 2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE), pp. 1–10 (2019). https://doi.org/10.1109/IWBOSE.2019.8666486
26. Zhang, A., Lin, X.: Towards secure and privacy-preserving data sharing in e-health systems via consortium blockchain. J. Med. Syst. **42**(8), 1–18 (2018). https://doi.org/10.1007/s10916-018-0995-5
27. Zhou, H., Milani Fard, A., Makanju, A.: The state of Ethereum smart contracts security: vulnerabilities, countermeasures, and tool support. J. Cybersecur. Privacy **2**(2), 358–378 (2022). https://doi.org/10.3390/jcp2020019