Hindawi
Security and Communication Networks
Volume 2023, Article ID 9559439, 20 pages
https://doi.org/10.1155/2023/9559439

Research Article

Distributed Public Key Certificate-Issuing Infrastructure for Consortium Certificate Authority Using Distributed Ledger Technology

Keita Kumagai,[1] Shohei Kakei ,[1] Yoshiaki Shiraishi ,[2] and Shoichi Saito [1]

[1]*Nagoya Institute of Technology, Gokiso-cho, Showa-ku, Nagoya, Aichi 466-8555, Japan*
[2]*Kobe University, Rokkodai-cho, Nada-ku, Kobe, Hyogo 657-8501, Japan*

Correspondence should be addressed to Shohei Kakei; kakei.shohei@nitech.ac.jp

Received 12 August 2022; Revised 28 February 2023; Accepted 3 May 2023; Published 7 June 2023

Academic Editor: Yujue Wang

With the development of cloud services and the Internet of Things, the integration of heterogeneous systems is becoming increasingly complex. Identity management is important in the coordination of various systems, and public key infrastructure (PKI) is widely known as an identity management methods. In PKI, a certificate authority (CA) acts as a trust point to guarantee the identity of entities such as users, devices, and services. However, traditional CAs that delegate the operations to a specific organization are not always suitable for heterogeneous services, and a new methodology is required to enable multiple stakeholders to securely and cooperatively operate a CA. In this study, we introduce the concept of a consortium CA and propose a distributed public key certificate-issuing infrastructure that realizes a consortium CA. The proposed infrastructure enables multiple organizations to cooperatively operate a CA suitable for services involving multiple stakeholders. We identify four requirements for the cooperative operation of a consortium CA and design the proposed infrastructure with distributed ledger technology. Furthermore, we present the implementation of smart contracts with Hyperledger Fabric and prove that the proposed infrastructure satisfies the four requirements. Finally, we confirm that certificate issuance and verification are stable at approximately 4 and 3 ms, respectively.

1. Introduction

Identity is critical to security mechanisms such as authentication and access control. Weak identity management mechanisms allow for impersonation attacks. Public key infrastructure (PKI) is widely known as a mechanism for confirming the identity of entities and linking the identity to a public key certificate. In PKI, a certificate authority (CA) acts as a trust point to guarantee the identity of entities, such as users, devices, and services and allows only authenticated entities to connect to the system. A CA issues a public key certificate that contains a public key and the identity of its owner. By verifying the public key certificate, the verifier can believe that the CA guarantees that the owner possesses the public key. A CA is responsible for the authenticity of the content of a public key certificate and is called a single trust point.

CAs can be classified into two types depending on operation forms: public and private. Public CAs are operated as services by socially trusted companies for Internet use. The main role of a public CA is to issue public key certificates to web servers. Public CAs and browser vendors release baseline requirements that all public CAs must follow[1]. Unlike a public CA, which is strictly enforced, private CAs are not required to be as strict as public CAs. A private CA has the same function as a public CA in issuing public key certificates, but a private CA is operated only within a specific domain by a private organization. Thus, a private CA can be used in a form suitable for domain-specific situations: when issuing numerous public key certificates at a high frequency, when issuing public key certificates dedicated to

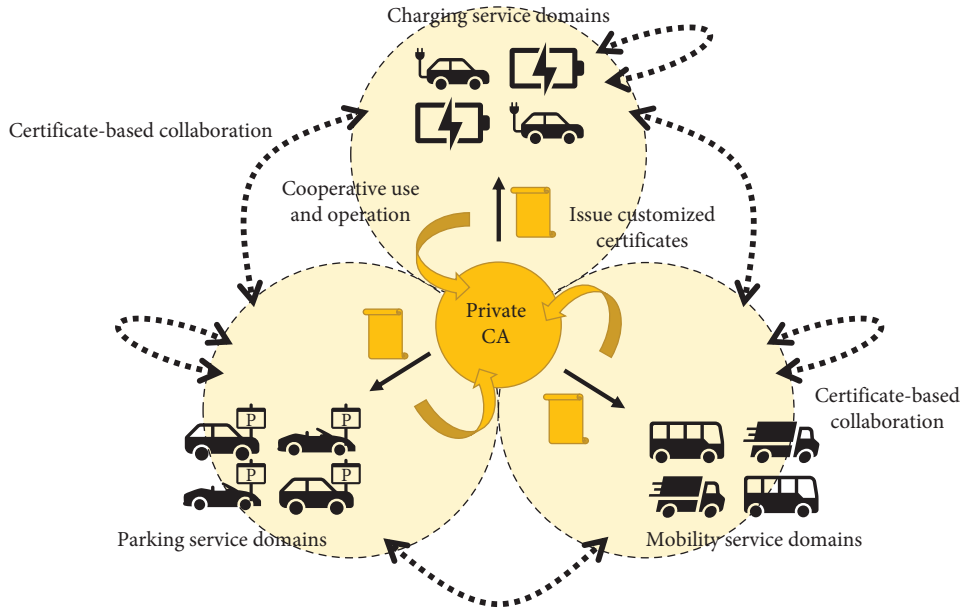

Figure 1: Example of cross-domain services based on [2].

a specific framework, or when policy prohibits the use of a public CA.

With the development of cloud services and wireless technology, various services and devices are being linked together. This is known as the Internet of Things, which involve connecting numerous devices to the Internet, not just people accessing the Internet with a browser. This trend has led to the creation of heterogeneous systems such as cross-domain services [2], cyber-physical systems [3], and digital twins [4]. Such a system involving various stakeholders requires flexible ID management according to individual situations, but it is difficult for public CAs and their additional mechanism, Certificate transparency (CT), to manage IDs flexibly. In CT, issued certificates are recorded in CT logs and made public. According to [5], client certificates published in CT logs can leak private enterprise information, such as business relationships, user growth measurements, and the existence of internal projects prior to their public announcements. The same is true for device certificates, which can reveal what kinds of devices and how many devices company own. In addition, there are financial cost issues with device certificates, and there are cases where public CAs are not suitable. The flexibility of private CAs may be suitable for this issue, but overcoming the stakes and operating a CA securely is challenging. Figure 1 is an example of a cross-domain service [2]. A consumer can use the charging services while using the parking services; this is an interdomain scenario. Moreover, mobility service domains, for instance, may provide route optimization services by integrating and combining different mobility services; this is a single-domain scenario. In this scenario, if a particular organization operates a private CA, the issuance of certificates would be managed by that organization. In such an operation, it is difficult to detect arbitrarily issued certificates. If each organization operates its own private CA, certificate issuance is not dependent on any one organization, but since each organization operates its own CA, the attack points increase. In addition, sufficient security measures must be taken by each organization to ensure that security is breached from the weakest point, that is, a methodology for securely operating a private CA under multiple stakeholders is required. In this study, we propose a distributed public key certificate-issuing infrastructure that allows multiple organizations to cooperatively operate CAs suitable for consortium-type services. In consortium-type services, because multiple organizations cooperate in providing services, the operation of the service should not be influenced by any particular organization. Therefore, to strictly guarantee the identity of service use in consortium-type services, the participating organizations in the consortium must be able to operate CAs in a cooperative manner. However, many security risks are associated with CA operations. The security risks increase if many organizations are involved in the operation of a CA. In this study, we focus on the risk of the unauthorized use of private keys. If a CA's private key is compromised by any organization in the consortium, the organization can issue arbitrary public key certificates using the compromised private key.

For security use of the CA's private key, we employ distributed ledger technology (DLT), also known as blockchain. The proposed infrastructure is built on a DLT system, and all organizations participate in the DLT system. Use of the private key is restricted to smart contracts only in the proposed infrastructure because the availability of the private key in any context leads to arbitrary public key certificates. Smart contracts have business logic agreed upon by all organizations. As long as a private key is used in a smart contract, its use can be considered as authorized by the participating organizations. However, smart contracts do not protect the confidentiality of the data they use; the private key could be compromised during the execution of smart contracts. In the proposed infrastructure, we use

Table 1: X.509 certificate format.

Presigned certificate	Version	
	Serial number	
	Signature algorithm	
	Issuer name	
	Validity period	Not before
		Not after
	Subject name	
	Subject public key info	Public key algorithm
		Subject public key
	Issuer unique ID	
	Subject unique ID	
	Extensions	
Signature algorithm		
Signature value		

Hyperledger Fabric Private Chaincode (FPC) [6] and protect the confidentiality of the private key during the execution of smart contracts.

This study contributes to the following:

(i) We introduce the concept of a consortium CA and identify the requirements for operating the consortium CA.

(ii) We design a distributed public key certificate-issuing infrastructure as a consortium CA using DLT.

(iii) We implement a prototype system for the proposed infrastructure with Hyperledger Fabric and evaluate the execution time of smart contracts that perform CA functions. The prototype system maintains the confidentiality of CA's private key using private chaincode, which is an extension of Hyperledger Fabric.

The remainder of this paper is organized as follows: Section 2 describes PKI, Intel Software Guard Extensions (Intel SGX), Hyperledger Fabric, and Hyperledger FPC, which are the technologies used in this proposal. Section 3 describes the concepts of consortium CA and the design policy of this proposal, and Section 4 discusses the data structures and transactions designed to implement the distributed public key certificate-issuing infrastructure. Section 5 describes the implemented smart contract. Section 6 provides a security analysis. Section 7 provides a qualitative and quantitative evaluation. Section 8 compares the proposed infrastructure with those in related studies, and Section 9 concludes the paper.

2. Background

2.1. PKI. PKI is a framework that links a public key and its owner's identity and provides a mechanism for verifying that the communicating party is actually the owner of the public key based on public key cryptography. A CA, which is a single trust point, is responsible for guaranteeing the identity of the owner and issues a public key certificate to the owner after confirming their identity. The CA signs the public key certificate with its own private key to guarantee its contents. The owner can prove its authenticity by presenting the public key certificate to other parties with whom it communicates.

A typical example of a public key certificate is X.509. The X.509 certificate format is standardized by the International Telecommunication Union (ITU). As shown in Table 1, an X.509 public key certificate consists of a presigned certificate, signature algorithm, and signature value. The presigned certificate contains basic information such as a subject name, a validity period, a public key, and extensions.

Public key certificates must be classified according to their purpose, and a major classification is whether they are public key certificates for CAs. Public key certificates for CAs are used to guarantee the ownership of public keys and verify other public key certificates issued by the CA. Whether a certificate is a CA certificate or not is expressed by a dedicated flag in the extensions field of the X.509 certificate.

Figure 2 shows the flow of the application, issuance, and verification processes. In PKI, a certificate holder (CH) presents a public key certificate to a relying party (RP), and the RP identifies the CH with the certificate. Public key certificates are issued by authorities: a registration authority (RA) and a certificate authority (CA). In the application process, the CH generates a key pair and signs a certificate signing request (CSR) with the private key (Step 1-1). The CSR contains all information necessary for the CA to issue a certificate, such as a common name, organization name, and public key of the CH. Then, the CSR is sent to the RA (Steps 1-2). The RA verifies and confirms the identity of the CH (Steps 1–3). After confirming the identity, the application process is completed. In the issuance process, the RA sends the CSR to the CA and requests issuing a public key certificate (Steps 2-1). Upon receiving the request, the CA first creates a presigned certificate based on the CSR. Then, the CA signs the presigned certificate with the CA's private key to create a public key certificate (Step 2-2) and issues this public key certificate to the CH (Steps 2-3). The CA maintains a repository that publishes the issued public key certificates and certificate revocation lists (CRL) and updates the repository as necessary. In the verification process, the CH presents the public key certificate issued by the CA to the relying party to prove the identity of the CH (Steps 3–1). The relying party obtains the CRL and CA's public key certificate (Steps 3-2) and verifies the certificate presented by the CH (Step 3-3). The information retrieved from the repository can be cached; thus, so the relying party does not need to access the repository's every verification process. If the verification is successful, the relying party can trust the identity of the CH.

Figure 3 shows the processes of signing and verifying public key certificates. The signing process is performed by encrypting the hashed presigned certificate with the CA's private key, as shown on the left side of Figure 3, that is, the signature value of the certificate can only be created by the CA. The CA is required to sign the certificate with the agreement of its content, thereby guaranteeing the content of the certificates. Moreover, as shown on the right side of Figure 3, the verification is performed by checking whether

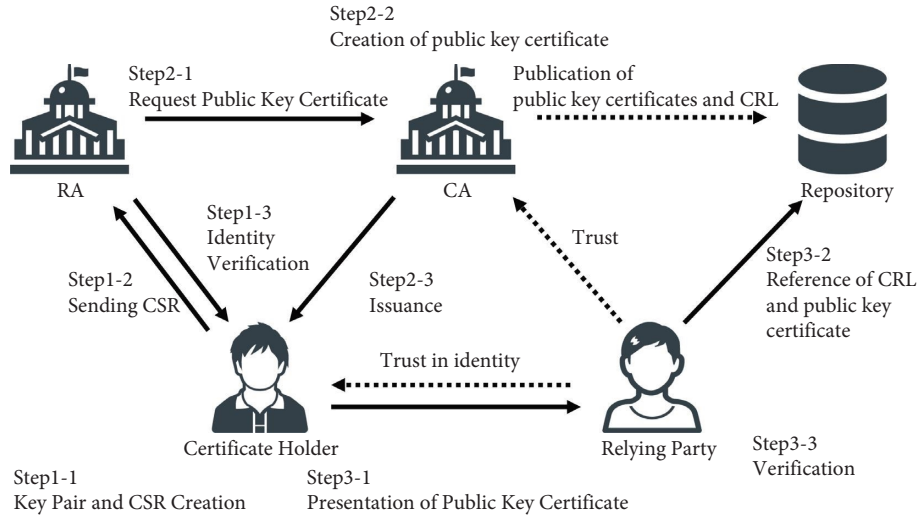

Figure 2: Flow of application, issuance, and verification processes of public key certificates in PKI.

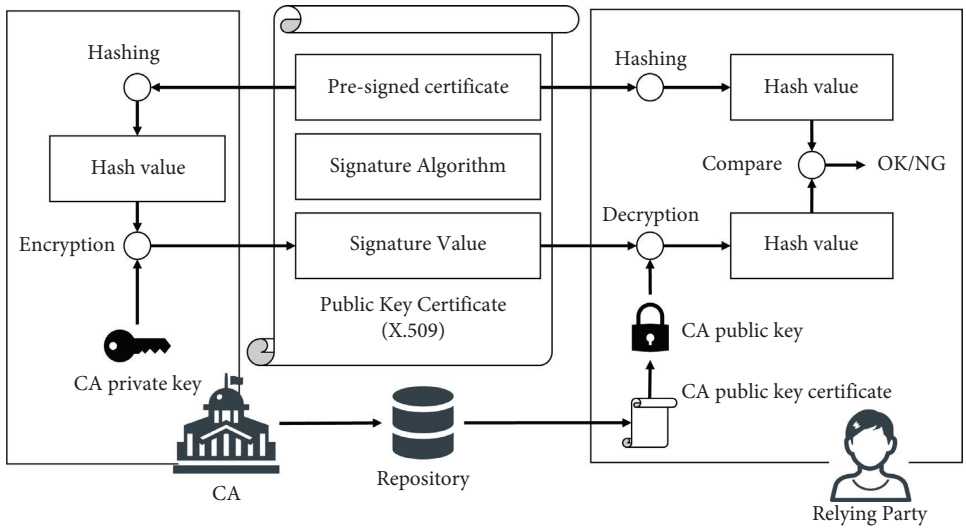

Figure 3: Signing and verification processes for public key certificates.

the hash value of the certificate matches the decrypted signature value with the CA's public key. The CA's public key certificate is publicly available; anyone can verify the certificate issued by the CA. Trust in the contents of a public key certificate is based on the CA properly managing its private key, that is, if a CA's private key is used fraudulently, the contents of a public key certificate can be falsified.

Certificate transparency (CT) [7] and the four cornered trust model [8, 9] are among the latest PKI technologies.

CT is an additional mechanism for increasing certificate transparency, in which the CA stores a record of every certificate it issues in third-party CT logs. By reviewing the CT log, the certificate holder can verify that no fraudulent certificates or certificates from nonpolicy CAs have been issued for his or her domain. This prevents the RP from trusting a fraudulently issued certificate. When a certificate is provided to the CT log, the CT logs return data called a signed certificate timestamp (SCT). The certificate owner presents the SCT with the certificate when it is presented. The RP can use the certificate and SCT to verify that the certificate's data are registered in the CT log.

The four cornered trust model adds an entity called a trust broker to the traditional PKI. The trust broker objectively evaluates the trustworthiness of the CA and its certificates; the RP does not trust the CA directly but trusts this trust broker. The trust broker provides three types of information to the RP:

(i) Quality of Certificate (QoCER): a score from 0 to 1 indicating the level of trust that can be placed in the certificate

(ii) Confidence level (CL): a score from 0 to 1 indicating the degree of confidence the trust broker has in the QoCER recommendation sent to the RP

(iii) Usage information about the recommended or allowed uses of the certificate

Based on the information provided by the trust broker, the RP will determine if the CA is trustworthy. In addition, the trust broker is responsible for the information provided to the RP and must respect and protect the privacy of the RP. The trust broker must be independent of the CA. Possible organizations that operate trust brokers include the following:

(i) A commercial organization whose business is to make recommendations regarding certificates

(ii) Governments wishing to promote electronic commerce in their countries

(iii) An international organization such as the United Nations to facilitate international trade

2.2. Intel Software Guard Extensions. Intel SGX [10, 11] is a trusted execution environment (TEE) that provides a secure execution environment in hardware. Intel SGX creates a cryptographically protected area called an Enclave and executes programs within the Enclave to protect data and execute programs securely. A function called "sealing" encrypts sensitive data within the Enclave, allowing it to be stored outside the Enclave in encrypted form. A function called "unsealing" can then decrypt the encrypted data within the Enclave. Therefore, sensitive data can be protected even outside the Enclave because it is encrypted rather than simply plain text when outside the Enclave.

There is a study [12] that uses Intel SGX to generate keys and certificates. This study seeks to improve the security of key generation and key distribution. The keys are securely generated and encrypted in the secure environment of Intel SGX. The encrypted key is then distributed as an optical signal for secure key distribution.

2.3. Hyperledger Fabric. Hyperledger Fabric [13], which is a DLT framework, is a Hyperledger project [14]. Hyperledger Fabric consists of two types of nodes: peer and orderer. A peer has a distributed ledger, communicates the execution of smart contracts, and updates data in the ledger to the orderer as transactions. The orderer sequences the transactions it receives from the peer and distributes the blocks generated from the transactions to all peers. The Fabric network consists only of authenticated nodes, which join the Fabric network using certificates issued by a dedicated private certification authority. Smart contract installation is based on the permission of each node participating in the Fabric network. Therefore, no particular administrator can arbitrarily decide which smart contracts to install, and only those smart contracts that are agreed upon by all nodes are installed.

Two types of commands are used to invoke smart contracts: invoke, which generates a transaction, and query, which does not generate a transaction. The invoke command can update data in the ledger by generating a transaction, but time is required for all nodes to synchronize their ledgers. Because the query command does not generate a transaction, it can only perform read operations that do not require updating the data in the ledger, thereby eliminating the need for synchronization time.

When a client invokes a smart contract via invoke, the following processes are performed. First, the client requests the creation of a transaction from multiple peers. Upon receiving the request, the peers execute the smart contract and return the result as a transaction to the client. After receiving transactions from all peers, the client sends the transaction to the orderer, which orders the transactions, generates the blocks, and sends the blocks to all peers. Peers verify each transaction in the block, and if no problems are identified, the transaction is reflected in the ledger.

In a DLT, each peer signs a transaction to prove that it has executed it. The key for signing is generated and managed by each peer. In Hyperledger Fabric, each organization can have a CA, and when a node joins the DLT, each CA generates a certificate for that node. When a node joins the DLT, each CA generates a certificate for that node, which is then used to verify the signatures in the blocks. If the keys used for signatures are compromised, there is a possibility of identity theft, so measures to prevent key compromise are considered. Some of the proposed key compromise countermeasures include the use of multisignature [15], the generation of secret keys from biometric information [16], and the use of hardware security module (HSM) [17].

2.4. Private Chaincode. Hyperledger FPC [6] is a mechanism to run Hyperledger Fabric on Intel SGX. The FPC allows and protects the execution of distributed ledgers and smart contracts, which are key-value type databases. The FPC is designed based on [18].

In FPC, the content of a peer can be protected by Intel SGX. Because the smart contract is executed within Enclave, no one can know the details of the smart contract execution and it can be executed securely. Data to be stored in the ledger are encrypted within Enclave and stored in the ledger in an encrypted state. When data are retrieved from the ledger, the data are retrieved in its encrypted state and decrypted in Enclave. At this time, the key for decryption exists only in Enclave. In addition, communication between nodes is protected by secure sockets layer (SSL) communication using a certificate issued at the time of registration.

In a blockchain such as Fabric, a peer executes the smart contract, and the peer can know the execution details of the smart contract. Therefore, utilizing highly private data is unsuitable in blockchains. However, the execution of smart contracts using Intel SGX, such as FPC, can be useful when leveraging highly private data.

A possible use case for FPC is to train a model for detecting brain abnormalities as a convolutional neural network (CNN) [19]. To obtain a highly accurate model, data owned by a single entity are insufficient; more data are required. Therefore, collecting data from many entities is desirable, but regulations under the General Data Protection Regulation (GDPR) and the Health Insurance Portability and Accountability Act of 1996 (HIPAA) make sharing CT scans and MRI images of the brain difficult. In such cases,

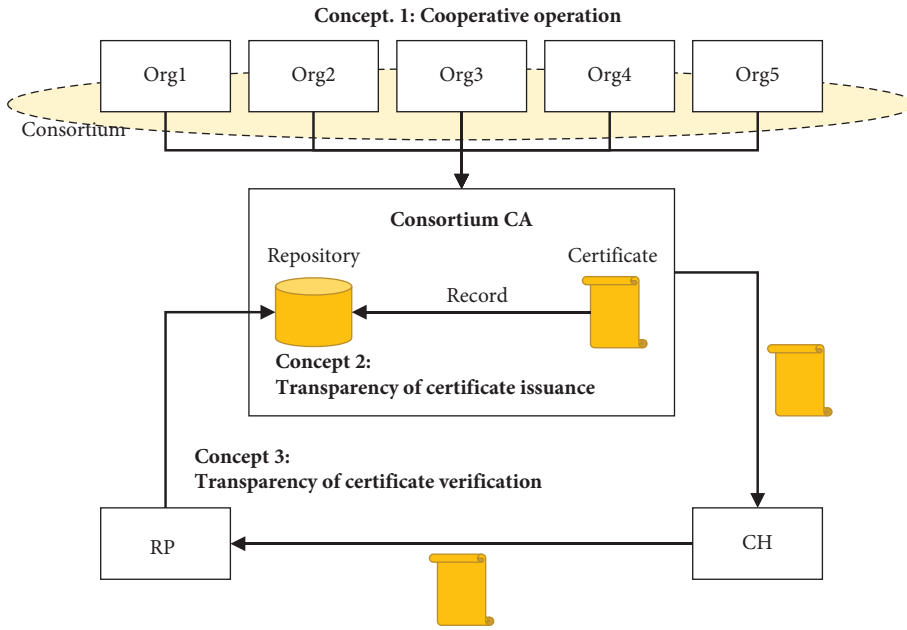

Figure 4: Concepts of consortium CA. This figure shows an example of a consortium consisting of five organizations.

FPC can be used to protect privacy while sharing information.

3. Concepts of Consortium CA and Design Policy

In this section, we introduce the concept of a consortium CA and define the design policies of a distributed public key certificate-issuing infrastructure for a consortium CA. Figure 4 shows the concepts of a consortium CA. Concept 1 is that the consortium CA should be operated by multiple organizations. Organizations participating in the consortium should have equal rights to use the consortium CA and not be restricted in their use by any particular organization. Concept 2 is that the transparency of certificate issuance should be guaranteed. To operate the consortium CA securely, all organizations should enable early detection of unauthorized certificates. Concept 3 is that the transparency of certificate validation should be guaranteed. No organization within the consortium should interfere with the RP's certificate validation.

A naive approach for cooperative operation among multiple organizations is to share the private key of the CA to an administrator of each organization, and issue public key certificates under the responsibility of each administrator. However, this approach is impractical because it requires trusting each administrator, and a malicious administrator can issue unauthorized public key certificates in secret using the private key illegally. For multiple organizations to cooperate in operating a distributed public key certificate-issuing infrastructure, public key certificates must be issued only when necessary at the administrators' discretion while preventing unauthorized issuance of certificates by administrators. Therefore, we define four requirements for the proposed infrastructure.

(1) Req. 1: public key certificates are issued without depending on a specific organization

(2) Req. 2: all public key certificates issued must be recorded

(3) Req. 3: the issuance of a public key certificate cannot be erased

(4) Req. 4: relying parties can confirm that a public key certificate has been issued

Req. 1 comes from Concept 1, Reqs. 2 and 3 come from Concept 2, and Req. 4 comes from Concept 3. Req. 1 prevents an administrator from arbitrarily issuing certificates. This requirement prohibits the delegation of the management of CA's private key to a specific administrator. Req. 2 guarantees the transparency of certificate issuance. This requirement provides accountability around certificate issuance and enables administrators to detect unauthorized certificates. Req. 3 also guarantees the transparency of certificate issuance. This requirement reinforces Req. 2 in terms of the permanence of the record. Req. 4 guarantees the transparency of certificate verification. This requirement allows RPs to check public key certificates without inhibition by any organizations.

We approach these requirements using smart contracts and distributed ledgers with DLT, that is, our approach is to design the logic of certificate issuance with smart contracts and the record of public key certificates with distributed ledgers. Because smart contracts allows transactions to be executed without a third party because the processing logic is predetermined, a public key certificate can be issued based on the logic agreed upon by all participants. This feature satisfies Req. 1. Although the logic may be shared, public key certificates may be issued independently of smart contracts if the private key can be used outside of smart contracts. In this case, such certificates are not recorded in distributed ledgers.

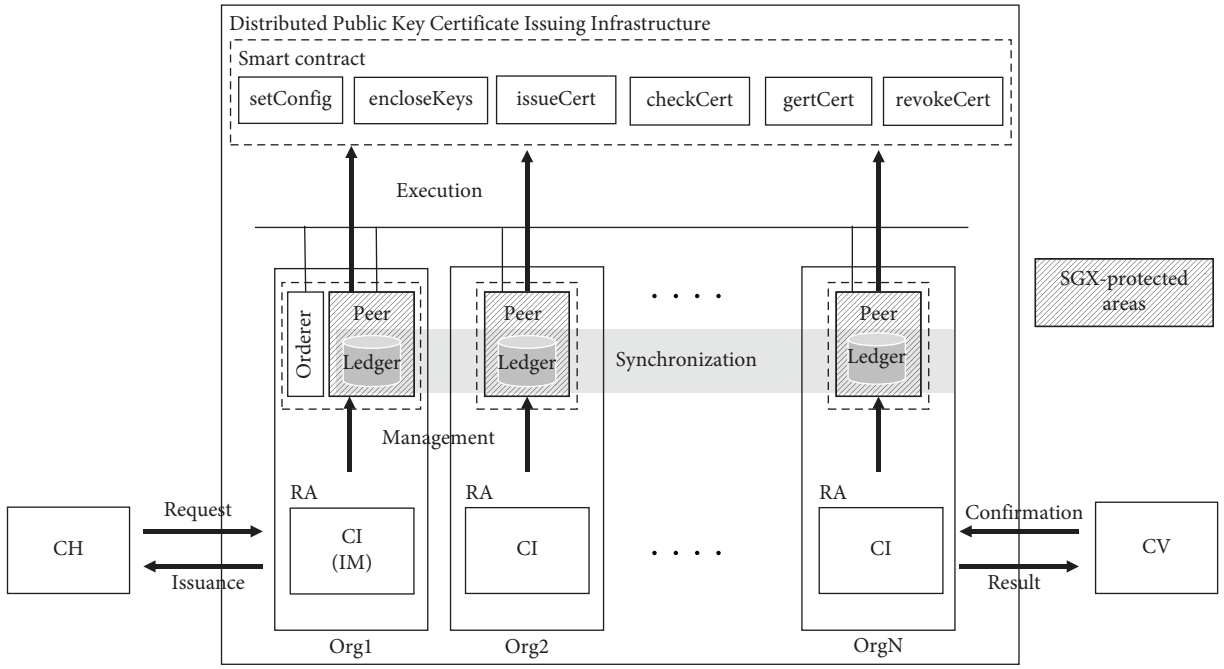

Figure 5: Structure of distributed public key certificate-issuing infrastructure.

Therefore, we limit the use of the private key within smart contracts. In this study, we refer making a specific private key used to sign public key certificates available only for a specific smart contract as "enclosing private keys." We approach Req. 2 with this idea. From Reqs. 1 and 2, distributed ledgers ensure that a record of issued public key certificates is maintained in a manner that cannot be tampered. Consequently, this is expected to conform with Req. 3.

Based on the design, public key certificates can be managed in distributed ledgers and placed under the control of the smart contract. The transparency of the issuance and management of public key certificates in the consortium CA is guaranteed. From this approach, we determine how to employ permissioned DLT as the proposed infrastructure. There are two reasons for this determination. First, designing an infrastructure that can be operated only among the identified organizations is possible. Because the CA functions are implemented using smart contracts, limiting entities that can use the functions is necessary. Permissionless DLT allows an unspecified number of entities to use smart contract. Second, permissionless DLT does not provide transaction finality. The lack of finality causes a potential risk of revoking an issued public key certificate. In permissioned DLT, transactions can be finalized by agreement among the participants. For these reasons, realizing the proposed infrastructure is desirable using permissioned DLT.

Finally, from the decision to use permissioned DLT, Req. 4 is derived. The decision to use permissioned DLT reduces the transparency of certificate verification. Because DLT system participants can only execute smart contracts, verification transparency is also required against a relying party, which is not a participant in the DLT system. This study incorporates the idea that the RP directly verifies the certificate verification results produced by the smart contract. This idea allows RPs to verify certificates without interfering with the organizations.

In this study, we design a distributed public key certificate-issuing infrastructure that satisfies the four requirements using Hyperledger Fabric, which is a permissioned DLT framework. Furthermore, we enclose private keys using FPC. The CA's private keys are securely used in smart contracts' Enclave.

4. Design of a Distributed Public Key Certificate-Issuing Infrastructure

4.1. Overview. Figure 5 shows the structure of the proposed infrastructure. The functionality of a CA is implemented by smart contracts on the FPC network and multiple organizations corresponding to RAs manage peers, which form the FPC network. Each RA can use CA functions by executing smart contracts through peers. In the FPC network, the ledgers held by each peer are synchronized, and the same information is written in all ledgers. Each organization runs a peer on an Intel SGX-enabled device. This ensures that the processing content and input/output of the smart contracts executed by peers, as well as the input/output to the ledgers, are maintained secretly.

The entities that comprise the distributed public key certificate-issuing infrastructure are as follows.

(1) Certificate holder (CH): A certificate holder is an entity that requests a certificate issuance to the proposed infrastructure.

(2) Certificate verifier (CV): A certificate verifier is an entity that verifies certificates.

Table 2: Data stored in the distributed ledger.

Key	Value
"CA." + hash (pk_{CA})	CA private key sk_{CA} Serial number of $cert_{CA}$ Private key sk_{Audit} Serial number of $cert_{Audit}$
"validKey"	"CA." + hash (pk_{CA})
"SerialNumber"	Serial number counter value
Serial number of a public key certificate	Public key certificate
"CRL"	List of a serial number of revoked public key certificate, revocation date, and reason code
"Config"	Issuer information Maximum expiry date

(3) Certificate issuer (CI): A certificate issuer is an entity that invokes a smart contract to issue a certificate.

(4) Infrastructure manager (IM): An infrastructure manager is the entity that sets up the proposed infrastructure. It is responsible for setting up the FPC network, deploying smart contracts, and creating and enclosing key pairs used to create public key certificates. After setup, it acts as a certificate issuer.

(5) Smart contract (SC): A smart contract is a program that reads and writes data on a distributed ledger in DLT. In the distributed public key certificate-issuing infrastructure, it realizes the function of a CA.

(6) Data store (DS): A data store is a ledger for storing data. In a distributed public key certificate-issuing infrastructure, the ledger is protected by Intel SGX and stores the private keys used for issuance and public key certificates issued.

4.2. Data Structure. The distributed ledger stores the CA's private key, public key certificates issued by the distributed public key certificate-issuing infrastructure, and revoked public key certificate information.

Table 2 presents the structure of the data stored in the distributed ledger.

The distributed ledger holds two types of private keys sk_{CA} and sk_{Audit}. The former is a private key for signing public key certificates, and the latter is a private key for guaranteeing verification results. In addition, a public key certificate corresponding to each private key is also stored. These data are stored using pointers, which is the string concatenated string "CA" and hash values of a public key pk_{CA} corresponding to sk_{CA}, as a key.

Because the hash value of pk_{CA} is unique, a different key is generated for each key encapsulation, allowing multiple private keys to be stored. To identify which of the multiple private keys will be used to sign public key certificates, the pointer of the valid private key is stored with string "validKey" as a key.

Then, the counter value of the serial number assigned to the public key certificate is stored with string "SerialNumber" as a key, and the issued public key certificate is stored with its serial number as a key.

Revoked public key certificates are stored with the serial number, revocation date and time, and reason codes of the revoked public key certificate as the key "CRL." Finally, the issuer's information and maximum validity period of the public key certificate are stored with string "Config" as the key.

4.3. Transaction. The distributed public key certificate-issuing infrastructure has four transactions: setup, certificate issuance, certificate verification, and certificate revocation. The variables of the proposed infrastructure are listed in Table 3.

4.3.1. Setup. Figure 6 shows the sequence of the setup. The setup prepares for the issuance of public key certificates in the distributed public key certificate-issuing infrastructure after the FPC network is activated. First, the necessary information for certificate issuance, such as issuer information and maximum validity period, is set by executing setConfig (Step 1). In setConfig, the issuer information and maximum expiration date are stored (Steps 2-3). Then, two key pairs are generated. sk_{CA} is used for certificate issuance, and sk_{Audit} is used during the certificate verification process (Step 4). Then, the generated key pair is enclosed in a distributed ledger by executing encloseKeys (Steps 5–7). Finally, the IM deletes the generated keys (Step 8).

4.3.2. Certificate Issuance. Figure 7 shows the sequence of certificate issuance. In a certificate issuance transaction, a public key certificate is issued to a CH. First, a CH creates a CSR and sends it to a CI, which requests for certificate issuance (Steps 1-2). Then, the CI executes issueCert to issue the certificate and returns it to the CH (Steps 3–6).

4.3.3. Certificate Verification. Figure 8 shows the sequence of certificate verification. In a certificate verification transaction, a CV checks the validity of the CH's certificate $cert_{CH}'$. First, the CV sends $cert_{CH}'$ and a random number $r1$ to a CI to check whether the public key certificate $cert_{CH}'$ to be verified is registered in the distributed public key certificate-issuing infrastructure (Step 1). The CI executes checkCert with $cert_{CH}'$ and the random number $r1$ as

Table 3: List of variables used in this paper.

Variable name	Description
sn	Serial number that uniquely identifies a public key certificate
iss	Issuer name of the public key certificate, which represents the distributed public key certificate-issuing infrastructure
sub	Subject name of the public key certificate
med	The maximum expiration date of public key certificates that can be issued by the distributed public key certificate-issuing infrastructure
rd	The date when a public key certificate issued by the distributed public key certificate-issuing infrastructure is revoked
rc	Reason code that indicates the reason why a public key certificate issued by the distributed public key certificate-issuing infrastructure has been revoked
ed	The expiration date of the public key certificate
csr	This is the certificate signing request to issue the subject's public key certificate

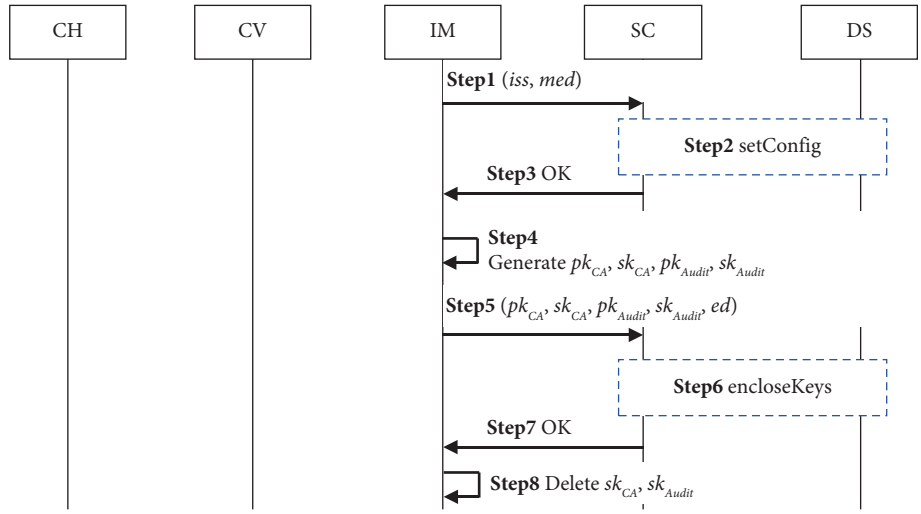

Figure 6: Setup sequence.

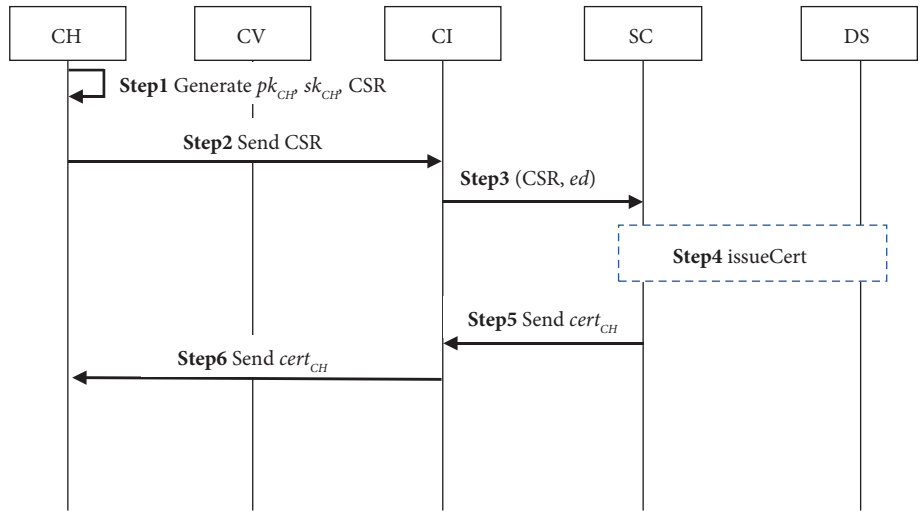

Figure 7: Certificate issuance sequence.

arguments and checks whether cert_{CH}' is a certificate stored in the ledger (Steps 2-3). If it can be confirmed, the SC signs $r1$ with sk_{Audit} and returns $\text{cert}_{\text{Audit}}$ and sig_{r1} (Steps 3-4), and the CI sends them to the CV (Step 5). If not stored in the ledger, cert_{CH}' is not a public key certificate issued by the distributed public key certificate-issuing infrastructure, and the CV is notified of this. Finally, the CV performs the verification (Step 6). In verification, after verifying $\text{cert}_{\text{Audit}}$

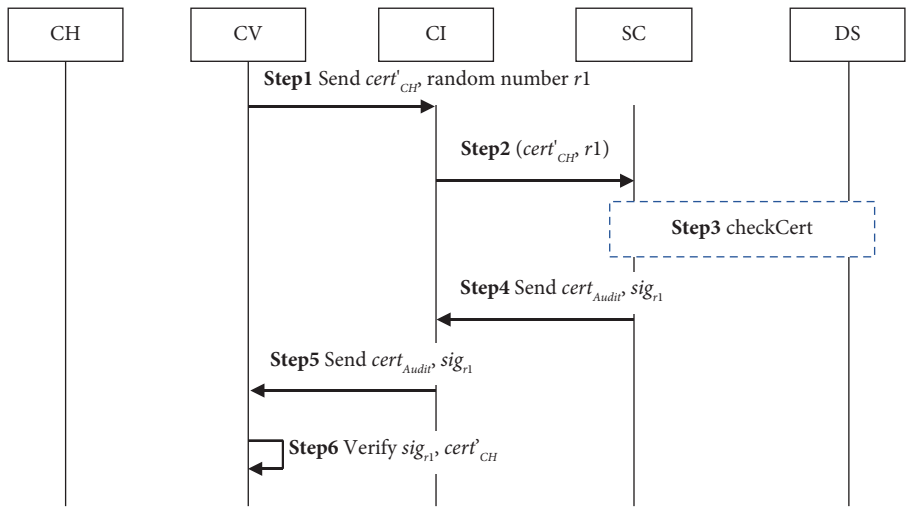

Figure 8: Certificate verification sequence.

using cert_{CA}, sig_{r1} can be verified using $\text{cert}_{\text{Audit}}$ to confirm that it has been processed by the SC execution. Then, cert_{CH}' is verified using cert_{CA}. It is noted that cert_{CA} is assumed to be known in advance.

4.3.4. Certificate Revocation. Figure 9 shows the sequence of certificate revocation. The certificate revocation transaction revokes a public key certificate. First, a CH makes a revocation request and sends the serial number of the public key certificate to be revoked to a CI (Step 1). The CI executes getCert and retrieves the public key certificate with its serial number as a key (Steps 2–4). Then, identification is required to confirm that the retrieved public key certificate belongs to the CH. To verify the identity, the CI requests the CH to sign a random number $r2$. The CI sends a random number $r2$ to the CH, and the CH generates a signature sig_{r2} for $r2$ using their private key (Steps 5–8). The CI verifies sig_{r2} with cert_{CH} (Step 9). If sig_{r2} is verified, the public key in cert_{CH} corresponds to CH's private key, which confirms that cert_{CH} belongs to the CH. Then, the public key certificate is revoked by executing revokeCert with the serial number of cert_{CH}, revocation date and time, and reason code as arguments (Steps 10–13).

5. Smart Contract Implementation

Six SCs are implemented, and the FPC's SCs can be developed using C++. The OpenSSL library is used to create and sign public key certificates. Table 4 provides the functions used in the proposed infrastructure.

5.1. setConfig. Algorithm 1 provides the pseudocode of setConfig. In setConfig, the putState function stores the issuer information and the maximum validity period with string "Config" as a key.

5.2. encloseKeys. Algorithm 2 provides the pseudocode of encloseKeys. encloseKeys takes key information and an expiration date as arguments. First, in line 16, the serial number and configuration information are obtained by the getState function to obtain the information necessary to generate a public key certificate. Two public key certificates are generated: cert_{CA} with pk_{CA} as the public key and $\text{cert}_{\text{Audit}}$ with pk_{Audit} as the public key. In line 3, a public key certificate is generated, and the issuer information, subject information, serial number, and expiration date are set as the certificate information. Then, the public key certificate is signed with sk_{CA}. Then, in line 22, a hash value of pk_{CA} is calculated, and the key information is stored in a ledger with string "CA.hash (pk_{CA})" as a key. String "CA.hash (pk_{CA})" is also stored with string "validKey" as a key, and the serial number is incremented.

5.3. issueCert. Algorithm 3 provides the pseudocode of issueCert. issueCert takes the CSR and expiration date as arguments. First, to obtain the information necessary to generate a public key certificate, the serial number, configuration information, and string "CA.hash (pk_{CA})" are obtained using the getState function, and key information is obtained using string "CA.hash (pk_{CA})" as a key. Then, from line 5, a public key certificate is generated. The certificate version is set to "3" to use the extended part of the public key certificate. In addition to setting the issuer information, subject information, serial number, and expiration date as certificate information, string "CA.hash (pk_{CA})" is inserted into the Authority Key Identifier area of the extended part of the public key certificate. This allows for the identification of which key signed the public key certificate during verification. In line 16, the public key certificate can then be signed with sk_{CA}, and a public key certificate can be generated. Finally, the generated public key certificate is stored in the ledger, and the serial number is incremented using the putState function.

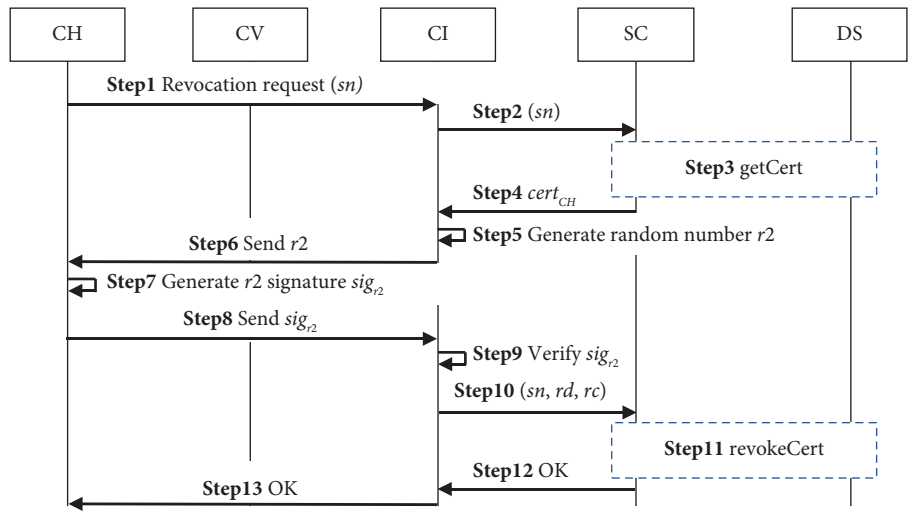

Figure 9: Certificate revocation sequence.

Table 4: List of functions used in this paper.

Function name	Description
putState (k, v)	Store in the ledger as key: k, value: v
$v \leftarrow$ getState (k)	Retrieve the value: v corresponding to key: k from the ledger
signCert (cert, sk)	Sign cert using sk

Input:*iss*, *med*
Output:*status*
(1) putState ("Config," (*iss*, *med*))
(2) **return** "OK"

Algorithm 1: setConfig.

5.4. checkCert. Algorithm 4provides the pseudocode of checkCert. checkCert first retrieves the serial number from the public key certificate cert_{CH}' using the getState function. From the serial number, it retrieves the public key certificate cert stored in the ledger and compares cert_{CH}' with the cert. This comparison determines whether cert_{CH}' is a public key certificate stored in the ledger. In line 6, the getState function obtains the CRL and checks the revocation status of cert_{CH}'. Then, the certificate is signed to guarantee that it has been processed by SC execution. In line 15, to identify the key that issued cert_{CH}', the extension from cert_{CH}' is extracted. The extension contains CA.hash (pk_{CA}), which can be used as a key to retrieve the key information. After extracting the key information, sign $r1$ with sk_{Audit} and generate sig_{r1}. Then, $\text{cert}_{\text{Audit}}$, which is necessary for verifying sig_{r1}, is extracted and output.

5.5. getCert. Algorithm 5 provides the pseudocode of getCert. In getCert, the public key certificate is retrieved from the serial number using the getState function.

5.6. revokeCert. Algorithm 6 provides the pseudocode of revokeCert. In revokeCert, the CRL stored in the ledger is retrieved. Then, *sn*, *rd*, and *rc* are added to the CRL, and the updated CRL is stored with string "CRL" as a key.

6. Security Analysis

In Section 6.1, we evaluate whether the proposed infrastructure satisfies the four requirements indicated in the design policy. In Section 6.2, we establish the threat model and we perform the security analysis based on the threat model in Section 6.3.

6.1. Evaluation of Meeting Requirements

6.1.1. Evaluation for Req. 1. Because a public key certificate is signed with a private key to guarantee the issuing entity, a CA is required to strictly manage the private key. For multiple organizations to cooperate in operating the proposed infrastructure, CIs must be able to use the CA's private key, but simply sharing the private key increases the risk of private key leaks and unauthorized use.

```
    Input: sk_CA, sk_Audit, pk_CA, pk_Audit, ed
    Output: status
(1) function genCert (sk, pk, conf, cnt, ed)
(2)   //Set issuer information, subject information, serial number and public key
(3)   cert.iss ⟵ conf.iss
(4)   cert.sub ⟵ conf.iss
(5)   cert.sn ⟵ cnt
(6)   cert.pk ⟵ pk
(7)   if ed does not exceed med then
(8)     cert.ed ⟵ ed//Set expiration date
(9)   else
(10)    return Error
(11)  end if
(12)  signCert (cert,sk)//Sign a certificate
(13)  return cert
(14) end function
(15) cnt ⟵ getState ("serialNumber")
(16) conf ⟵ getState ("Config")
(17) cert_CA ⟵ genCert (sk_CA, pk_CA, conf,cnt, ed)
(18) cert_Audit ⟵ genCert (sk_CA, pk_CA, conf,cnt + 1, ed)
(19) putState (cert_CA.sn, cert_CA)
(20) putState (cert_Audit.sn, cert_Audit)
(21) hash ⟵ Hash (pk_CA)
(22) putState ("CA." + hash, sk_CA, sk_Audit, cert_CA.sn, cert_Audit.sn)
(23) putState ("validKey," "CA." + hash)
(24) putState ("serialNumber," cnt + 1)
(25) return "OK"
```

Algorithm 2: encloseKeys.

```
    Input: csr, ed
    Output: cert
 (1) cnt ← getState ("serialNumber")
 (2) conf ← getState ("Config")
 (3) validkey ← getState ("validKey")
 (4) (sk_CA, sk_Audit, CA.sn, Audit.sn) ← getState (validkey)
 (5) cert.version ← 3
 (6) cert.iss ← conf.iss
 (7) cert.sub ← csr.sub
 (8) cert.sn ← cnt
 (9) cert.pk ← csr.pk
(10) cert.extension ← validkey
(11) if ed does not exceed med then
(12)   cert.ed ← ed
(13) else
(14)   return ← Error
(15) end if
(16) signCert (cert, sk_CA)
(17) putState (cert.sn, cert)
(18) putState ("serialNumber", cnt + 1)
(19) return cert
```

Algorithm 3: issueCert.

In the proposed infrastructure, a series of certificate creation processes, including the signing process using the private key, is implemented as an SC to avoid dependence on a specific CI, as shown in Algorithm 3. Since the private keys and the SC execution are protected by Intel SGX, all the CIs have fair access to the private keys and cannot interfere with other CIs. In this manner, the proposed infrastructure satisfies Req. 1.

```
    Input: cert'_CH, r1
    Output: sig, cert_Audit
(1) //Retrieve a certificate using cert'_CH serial number
(2) cert ← getState (cert'_CH.sn)
(3) ifcert and cert'_CH are not matched then
(4)     return Error ("Cert is invalid")
(5) end if
(6) crl ← getState ("crl")
(7) i ← 0
(8) //Confirm revocation status
(9) whilecert.sn ! = crl[i].sn do
(10)    i ← i + 1
(11) end while
(12) ifcert is revoked then
(13)    return Error ("Cert is revoked")
(14) end if
(15) extension ← cert'_CH.extension
(16) (sk_CA, sk_Audit, CA.sn, Audit.sn) ← getState (extension)
(17) sig ← sign(r1, sk_Audit)//Sign r1 using sk_Audit
(18) cert_Audit ← getState (Audit.sn)
(19) returnsig, cert_Audit
```

Algorithm 4: checkCert.

```
    Input: sn
    Output: cert
(1) cert ← getState (sn)
(2) return cert
```

Algorithm 5: getCert.

```
    Input: sn, rd, rc
    Output:status
(1) crl ← getState ("crl")//Retrieve crl
(2) crl ← add (sn, rd, rc)//Add revocation information to crl
(3) putState ("CRL," crl)
(4) return "OK"
```

Algorithm 6: revokeCert.

6.1.2. Evaluation for Req. 2. Each organization cooperates to maintain the proposed infrastructure; ans simultaneously plays a role in monitoring the other organizations to ensure that no fraudulent public key certificates are issued. Because the public key certificate creation process is defined by the SC, a certain level of security is guaranteed, such as prohibiting the use of weak cryptographic algorithms. However, being able to evaluate items that cannot be evaluated uniformly, such as the eligibility of the subject of issuance and validity of the expiration date, using public key certificates that have actually been issued is desirable.

In the proposed infrastructure, the SC must be performed to issue a public key certificate. The certificate issuance process is realized as a single transaction of issuance and recording, with public key certificates being stored in the distributed ledger at the same time they are created. This ensures that all public key certificates issued are recorded in the distributed ledgers. By managing the public key certificates in the distributed ledgers, the public key certificates issued are shared by all organizations. Meanwhile, the proposed infrastructure requires the infrastructure manager to enclose the private keys in the distributed ledgers. The IM must confirm that the enclosed private keys have been securely deleted, as shown in Figure 6. If the private keys are not deleted, the IM can create public key certificates in secret. However, public key certificates created without executing an SC are not recorded in the distributed ledgers; thus, only public key certificates that have been legitimately issued are recorded. The certificate verification sequence also checks whether a public key certificate is registered in the

distributed ledgers. Because public key certificates created in secret are not recorded in the distributed ledger, they can be detected by the certificate validation sequence. In this manner, the proposed infrastructure satisfies Req. 2.

6.1.3. Evaluation for Req. 3. If the issuance of a public key certificate can be erased, inconvenient issuance facts can be erased later. Because the issuance of a public key certificate is recorded as the storage of the public key certificate in distributed ledgers, the erasure of the issuance of a public key certificate means the erasure of the public key certificate stored in the distributed ledgers.

In relation to Req. 2, even if the public key certificates issued are recorded in the distributed ledgers without omission, fraudulent public key certificates cannot be detected if the records are being tampered with or deleted. The public key certificates stored in a distributed ledger cannot be deleted based on the distributed ledger's tamper resistance, which is generally provided by DLT. In addition, the proposed infrastructure, no SC can delete public key certificates recorded in the distributed ledgers, and the public key certificates cannot be deleted by abusing legitimate SCs. Even if SCs cannot delete public key certificates, issuance of public key certificates cannot be confirmed if they are no longer recognized by the proposed infrastructure. Therefore, public key certificates issued are stored with their hash value as a key to ensure that they are stored uniquely without being overwritten. In this manner, the proposed infrastructure satisfies Req. 3.

6.1.4. Evaluation for Req. 4. The verification of public key certificates involves checking that the public key certificate is signed with sk_{CA} and that the public key certificate is recorded in the distributed ledgers. As shown in Figure 5, the existence of this record is verified by a CI who has the authority to execute the SC. There is a risk concerning the CI returning incorrect results because the CI can replace the verification results.

The proposed infrastructure uses sk_{Audit} to sign the verification results in addition to sk_{CA} to sign public key certificates. The SC that performs the verification process signs the verification results with sk_{Audit} enclosed in the distributed ledgers. Therefore, the verification results are guaranteed to originate from the SC. The CV can verify the existence of public key certificates without unauthorized intervention by the CI. In this manner, the proposed infrastructure satisfies Req. 4.

6.2. Threat Models. Since the proposed infrastructure allows CIs to issue certificates, we conduct threat modeling by assuming a CI to be a malicious entity. The attacker's goal is to create a fraudulent certificate that will pass authentication checks. From the modeling assumption, the attacker has the capability of a CI. Therefore, the attacker is able to propose deploying smart contracts, agree upon processing of smart contracts, and execute smart contracts. According to the attacker's capability, there can be two types of attacks: deploying a smart contract with incorrect process injected and intervening in the verification process. In the former type, the attacker proposes a smart contract including the function that is advantageous in attacking. In this paper, we analyze the possibility of a smart contract that can issue a CA certificate. In the latter type, the attacker illegally intervenes in the verification process to avoid detection of invalid certificates. If the attacker obtains sk_{CA}, the attacker can generate invalid certificates using it. In this type of attack, the attacker needs to spoof that the invalid certificates are recorded on the distributed ledger.

6.3. Security Analysis Based on Threat Models. For the attack of issuing a CA certificate, if the attacker can generate a CA certificate by issueCert, the attacker uses a private key of that CA certificate and can issue certificates with that CA certificate as their parent. In this case, the CA certificate issued by the attacker is an intermediate certificate for the root CA certificate stored in the distributed ledger. This attack is detectable from two points of view: installation of smart contracts at each CI and verification of certificates. For the first point, the attacker proposes a smart contract, which includes the function issuing a CA certificate to all CIs, and all CIs have to install that smart contract. Since CA certificates are also a type of public key certificate [20], they can essentially be generated in the proposed infrastructure. However, the difference can be determined by the usage of certificates expressed in the extension area of certificates (e.g., Basic Constraints and Key Usage). Each CI is required to understand these differences and decide whether or not to install smart contracts. Although the specific scheme is the future work of this study, it is necessary to establish a policy to determine whether or not to install smart contracts and a system that allows all CIs to check the policy mechanically. For the second point, although the intermediate certificate generated by the attacker is recorded in the distributed ledger, subordinate certificates of that intermediate certificate are not recorded in the distributed ledger. Therefore, invalid certificates can be detected by checkCert.

For the attack of spoofing certificate issuance records on the distributed ledger, the attacker attempts to spoof that certificates are recorded in the distributed ledger on the certificate verification sequence. Since this attack falsifies certificates created without using issueCert as legitimate certificates, it is expected to be used in conjunction with the attack of issuing an intermediate CA certificate. In this attack, the attacker attempts to manipulate the checkCert results either by altering the checkCert results or by falsifying the records of the distributed ledger. The checkCert results are guaranteed by Req. 4, as shown in Section 6.1.4. In addition, it is not possible to spoof certificate issuance records in the distributed ledger since the attacker cannot inject invalid records into the distributed ledger by Req. 2 and Req. 3. If the attacker can obtain sk_{Audit}, the attacker can generate the proof without checkCert. However, the attackable period is very short since sk_{Audit} is deleted by the IM at Step 8 in the setup sequence.

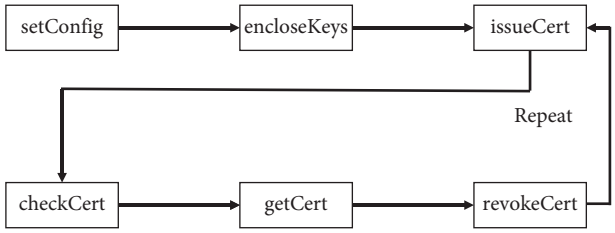

Figure 10: Execution environment.

7. Experimental Evaluation

7.1. Experiment's Summary. We evaluated the basic performance of the proposed infrastructure. Our evaluation aims to analyze the proposed infrastructure from the following three perspectives:

(i) The trend in the time required for the certificate issuance

(ii) The trend in the time required for the certificate verification

(iii) The trend in the time required for the certificate revocation

In the experiments, we executed the proposed SCs in the order shown in Figure 10. After setting up the proposed infrastructure with setConfig and encloseKeys, we repeatedly issued, verified, and revoked certificates 100 times with issueCert, checkCert, getCert, and revokeCert. Within each trial, we measured the execution time for each SC.

To conduct the experimental evaluation, we implemented the system shown in Figure 11. Two peers and one orderer were created as containers, and an FPC network was constructed with them. Our proposed SCs were executed on the FPC network, and its processing time was measured. The measurement was performed with SGX in simulation mode.

7.2. Evaluation Results. The execution times are shown in Table 5, and time to update the status of all ledgers is shown in Table 6. IssueCert and revokeCert write data to the ledger. Therefore, an operation to update the status of all ledgers is necessary to synchronize the ledgers. Looking at the first execution time, setConfig, getCert, and revokeCert required less than 1 ms, whereas encloseKeys, issueCert, and checkCert required several ms. setConfig and getCert have short execution time due to their small amount of processing.

First, we describe the evaluation results of the certificate issuance where issueCert is used. From Figure 12, issueCert has the longest execution time, and it is within a range of approximately 3.9 to 4.2 ms. As shown in Algorithm 3, since issueCert only uses the input data to create a certificate, the execution time is considered to be constant and consistent with the evaluation results. From Table 6, the time to update the status of all ledgers for issueCert is also in the range of 0.91 to 0.98 ms. From the implementation of the algorithm and the experimental results, we confirmed that the proposed infrastructure tends to be able to issue certificates stably.

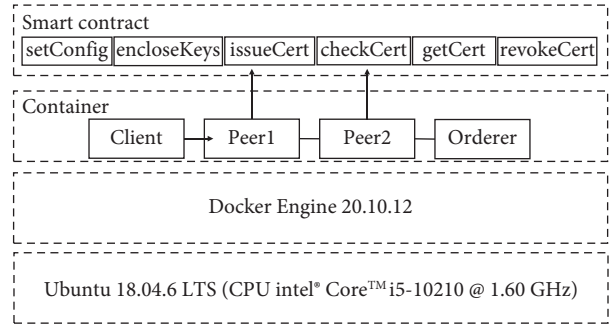

Figure 11: Execution procedure.

Then, we describe the evaluation results of the certificate verification where checkCert is used. From Figure 12, the execution time for checkCert is within a range of approximately 2.9 to 3.4 ms. As shown in Algorithm 4, since checkCert performs a linear search of the CRL to check that the certificate has not been revoked, the execution time is considered to increase linearly. However, our evaluation results showed no clear increase in execution time for as few as 100 CRLs. Once a revoked certificate is listed on CRLs, it is removed from the CRLs when the certificate expires. In other words, the number of CRLs does not continue to increase monotonically, and there is basically an upper limit to the number of CRLs. The upper limit depends on the application to which the consortium CA is applied, but we confirmed that about 100 CRLs have no significant effect on the performance change.

Finally, we describe the evaluation results of the certificate revocation where getCert and revokeCert are used. From Figure 12, the execution time for getCert shows little change, ranging from about 0.5 to 0.6 ms. As shown in Table 2, certificates are stored with its serial numbers as keys, and certificates can be retrieved from the ledger by simply inputting the key as shown in Algorithm 5. As a result, the processing cost of getCert is constant and small. On the other hand, the execution time for revokeCert increased from 0.7 to 2.2 ms. In revokeCert, the inputting certificate is added to the CRLs; hence, it takes the execution time to read and write the CRLs. The number of CRLs increases monotonically in this experiment. Thus, the size of data to be read and written has increased, and the processing time seems to have increased accordingly. However, the execution time is smaller than that of checkCert, which will be executed more frequently, and the number of CRLs is expected to be capped, so performance is not expected to be significantly impacted. From Table 6, in addition, the time to update the status of all ledgers for revokeCert is in the range of 0.67 to 0.75 ms. Compared to the time to update the status of all ledgers for issueCert, the time for revokeCert is smaller. This suggests that the amount of data written to the ledger is related to the time to update the status of all ledgers. Therefore, the time for revokeCert did not show an obvious increase in time, although the amount of data to be written would increase as the number of executions increased.

Table 5: Execution time.

SC	Execution time (ms)					
	1 time	20 times	40 times	60 times	80 times	100 times
setConfig	0.495 ± 0.003	—	—	—	—	—
encloseKeys	4.734 ± 0.017	—	—	—	—	—
issueCert	3.972 ± 0.044	3.876 ± 0.040	4.139 ± 0.491	3.885 ± 0.069	4.206 ± 0.783	3.900 ± 0.057
checkCert	2.946 ± 0.004	2.954 ± 0.014	3.098 ± 0.108	3.353 ± 0.273	3.219 ± 0.246	3.112 ± 0.002
getCert	0.582 ± 0.060	0.566 ± 0.000	0.563 ± 0.000	0.607 ± 0.005	0.641 ± 0.008	0.562 ± 0.000
revokeCert	0.708 ± 0.051	0.774 ± 0.135	1.236 ± 0.011	1.870 ± 0.273	1.967 ± 0.176	2.194 ± 0.025

Table 6: Time to update the status of all ledgers.

SC	Time to update the status of all ledgers (ms)					
	1 time	20 times	40 times	60 times	80 times	100 times
issueCert	0.972 ± 0.005	0.931 ± 0.002	0.953 ± 0.005	0.980 ± 0.016	0.920 ± 0.005	0.914 ± 0.012
revokeCert	0.683 ± 0.004	0.733 ± 0.003	0.702 ± 0.004	0.679 ± 0.002	0.714 ± 0.002	0.759 ± 0.014

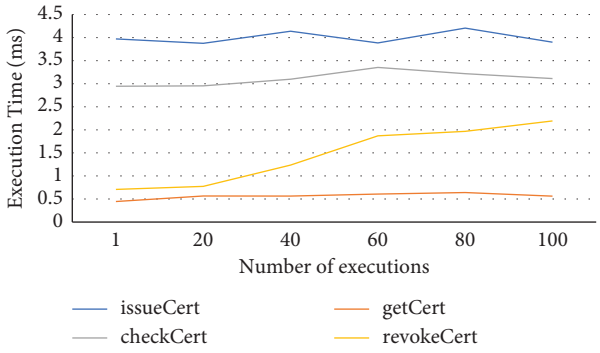

Figure 12: Change in execution time for issueCert, checkCert, getCert, and revokeCert.

8. Related Work

Table 7 shows a comparison of the proposed infrastructure and related studies combining CA-based PKI and blockchain. We reviewed the basic characteristics of them and analyzed them with regard to the fulfillment of the proposed requirements. The basic characteristics of Table 7 are listed below.

(i) Permission type: this indicates whether the study employs permissioned or permissionless blockchain

(ii) Blockchain type: this indicates a type of blockchain systems used in the study

(iii) Private key location: this indicates where the private key used for the certificate is stored

(iv) Certificate generation: this indicates which entity generates the certificate

(v) Certificate verification: this indicates whether the certificate verification process is performed locally or using the SC

(vi) Certificate revocation: this indicates how certificate revocation is performed

(vii) Registration confirmation: this indicates whether a process is used to confirm that the certificate is recorded on the blockchain

Our objective is to propose a consortium CA that can be cooperatively operated by multiple organizations. In contrast, most of the related studies proposing blockchain-based PKI have the objective of resolving the single point of trust of the CA and making the PKI more secure or improving the system for that purpose. Although the differences in the purpose make a complete comparison difficult, we analyzed whether the related studies meet the following four requirements defined in this paper and compared the differences among the related studies.

(i) Req. 1: this indicates whether "Public key certificates are issued without depending on a specific organization." is satisfied

(ii) Req. 2: this indicates whether "All public key certificates issued must be recorded." is satisfied

(iii) Req. 3: this indicates whether "The issuance of a public key certificate cannot be erased." is satisfied

(iv) Req. 4: this indicates whether "Relying parties can confirm that a public key certificate has been issued." is satisfied

The results of the analysis show that the related studies can be divided into five categories from the perspective of the requirements, as shown in Table 7. The first is a category that satisfies none of the requirements. This category includes those that apply blockchain technology for purposes other than certificate issuance (e.g., sharing revocation information or CA policies). Lei et al. [21] propose an efficient certificate revocation scheme in vehicle communication systems (VCS). Ahmed and Aura [22] propose a SC-assisted public key infrastructure (SCP) to manage certificate trust statuses. CAs and domain owners register and publish their certificate policies on the blockchain such that each participant can verify the trust status of certificates. IKP [23] incentivizes CAs to act ethically and report fraud, thereby discouraging abusive behavior. Elloh Adja et al. [24] propose

Table 7: Comparison with related studies of CA-based PKI using blockchain.

	Permission type	Blockchain type	Private key location	Certificate generation	Certificate verification	Certificate revocation	Registration confirmation	Requirements			
								Req. 1	Req. 2	Req. 3	Req. 4
Lei et al. [21]	Permissioned	Custom	Local storage	CA	Local	SC	●	●	●	●	●
Ahmed and Aura [22]	Permissionless	Ethereum	Local storage	CA	Local	SC	●	●	●	●	●
IKP [23]	Permissionless	Ethereum	Local storage	CA	Local	—	●	●	●	●	●
Elloh Adja et al. [24]	Permissionless	Ethereum	Local storage	CA	—	SC	●	●	●	●	●
Certchain [25]	Permissioned	Ethereum	Local storage	CA	Local	SC	○	●	●	○	●
Block CAM [26]	Permissioned	Ethereum	Local storage	CH	Local	SC	○	●	●	○	●
Boyen et al. [27]	Permissionless	Ethereum	Local storage	CA	Local	SC	○	●	●	○	●
CBPKI [28]	Permissionless	Ethereum	Cloud storage	CA	Local	SC	○	●	●	○	○
Wang et al. [29]	Permissionless	Custom	Local storage	CA	Local	SC	○	●	●	○	○
Hwang et al. [30]	Permissionless	Ethereum	Local storage	CA	—	SC	○	●	●	○	○
CertLedger [31]	Permissionless	Ethereum	Local storage	CA	SC	SC	●	●	●	○	○
Yakubov et al. [32]	Permissionless	Ethereum	Local storage	CA	SC	SC	●	●	◐	○	○
Rashid et al. [33]	Permissionless	Ethereum	Local storage	CA	Local	SC	○	●	◐	○	○
Proof chain [34]	Permissionless	Ethereum	Local storage	CA	SC	SC	○	●	○	○	○
Block PKI [35]	Permissionless	Ethereum	Local storage	SC, CA	Local	—	○	●	○	○	○
Li et al. [36]	Permissionless	Ethereum	Local storage	CA	Local	Local	●	●	○	○	○
Proposed scheme	Permissioned	Hyperledger Fabric	Ledger	SC	Local	SC	○	○	○	○	○

○: yes; ◐: partially yes; ●: no.

a new certificate revocation method and status verification scheme. It stores certificate revocation information in a public blockchain and provides a mechanism like a CRL distribution point.

The second is a category that satisfies only Req. 3. The studies in this category tend to utilize the blockchain's tamper-resistance capabilities to ensure the accountability of PKI for specific peers authorized to participate in the blockchain system. Certchain [25] involves an auditing scheme using blockchain for secure SSL communications. It records, publishes, and audits certificate operations such as certificate registrations, renewals, and revocation on the blockchain. BlockCAM [26] proposes a cross-domain authentication model using blockchain. The CA becomes a node on the blockchain, and the CA registers its issued certificates on the blockchain. Boyen et al. [27] propose DPKIT, which eliminates the CA as a single point of failure and ensures transparency in certificate issuance and revocation. Although DPKIT employs permissionless blockchain, auditing requires the cooperation of a dedicated node called DPKIT peer.

The third is a category that satisfies Reqs. 3 and 4. The studies in this category tend to utilize the blockchain technology for ensuring the accountability of PKI for even entities that do not participate in the blockchain system. However, certificates registered in the blockchain are still under the control of a CA, and there is a possibility of omission of certificate registration or registration of fraudulent certificates. CBPKI [28] involves a blockchain-based cloud-based PKI. It aims to leverage the security measures of cloud platforms by offloading certification authority to the cloud. Wang et al. [29] propose a certificate and revocation transparency system to prevent impersonation attacks using fraudulent certificates. Hwang et al. [30] solve the PKI problem using public blockchains that cannot handle numerous certificates using TP-Merkle trees. Kubilay et al. [31] propose a new PKI model with blockchain-based certificate transparency, CertLedger. CertLedger manages the states of all certificates and their revocation status and the set of the trusted CA certificates in blockchain.

The fourth is a category that satisfies Reqs. 3 and 4 and partially satisfies Req. 2. The studies in this category include an additional mechanism to prevent the registration of fraudulent certificates for the third category. Therefore, Req. 2 is partially satisfied. Yakubov et al. [32] propose a blockchain-based PKI management framework for managing certificates. Each CA has a dedicated SC, which allows it to register and revoke certificates. Rashid et al. [33] propose a blockchain-based mechanism for the issuance and management of transparent and secure digital certificates that can prevent CA abuse. The proposed system can solve attacks like Sybil attack, Spoofing attack, and MITM attack.

The fifth is a category that satisfies Reqs. 2, 3, and 4. The studies in this category have an additional mechanism to enforce that all of the certificates are recorded in blockchain for the fourth category. Specifically, the recording process is integrated into the issuing process. Saleem et al. [34] propose a decentralized PKI framework, ProofChain, to improve security. Blockchain miners act as CAs and issue certificates by storing them in the blockchain after each CA signs them. Dykcik et al. propose BlockPKI [35] to reduce the power of individual CAs and to make their actions publicly visible and accountable. A domain owner publishes a request on the blockchain for the issuance of a certificate along with the expected set of CAs. Then, each of the designated CAs performs domain validation and publishes a certificate with multisignature on the blockchain. Li et al. [36] propose a possible solution with new blockchain technology to solve problems like single-point attacks and man-in-the-middle attacks. There is no CA as a third party, and the verifier acts as a CA. When a user registers credential information with the blockchain, the verifier issues the user a certificate for its own server, which is stored in the blockchain. The user can then retrieve the certificate from the blockchain.

In addition to CA-based PKI, a blockchain-based Web of Trust PKI has also been proposed. WoT-based PKI guarantees an identity of a public key without relying on a single point of trust such as a CA. BCTrust [37] proposes a secure communication protocol in wireless sensor networks (WSN). Web of Trust does not use certificates, but authenticates messages by recording them on a blockchain. BlockPGP [38] proposes a blockchain-based framework that manages pretty good privacy (PGP) certificates and key-server infrastructure with high trust. Certificate holders can register and revoke PGP certificates on the blockchain and sign the certificates of others. Blockstack [39] is a blockchain-based naming and storage system . It associates public keys, data, and usernames in a similar manner as PGP using blockchain. DPKI [40] proposes a PKI solution to address attacks coming from a single point of failure in the Industrial Internet of Things (IIoT). DPKI uses a permissioned blockchain, where the participants are all devices in the IIoT network. SCPKI [41] is an alternative PKI system based on a decentralized and transparent design using the Web-of-Trust model and smart contracts on the Ethereum blockchain. Each entity stores its identity in the blockchain, and its authenticity is guaranteed by signatures of other entities. The blockchain stores identities and public keys, and each participant signs these data. Fromknecht et al. propose Certcoin [42], which ensures the association of public keys and identities with a public ledger. Then, Patsonakis et al. [43] improve Certcoin in terms of data size and implement their proposed system in [44]. Qin et al. propose Cecoin [45], which resolves a single point of failure of PKI by recording certificates as currencies to the Bitcoin system. Cecoin has the identity assignment to support delegation of certificate ownership.

The proposed infrastructure is classified into a new category, which satisfies all of the requirements. One major difference from existing research is the capability of storing private keys in a distributed ledger by applying Intel SGX. In most existing studies, a private key of a CA is stored in a CA's local storage. This capability allows the SC to use the private key for generating and confirming certificates.

According to our analysis, none of the existing studies satisfy Req. 1. In the consortium CA, it is important to be able to use a private key cooperatively among CAs participating in the consortium and to prevent fraud by

a specific CA. The proposed infrastructure satisfies Req. 1 by employing blockchain technology and Intel SGX. As a different approach from our proposal, there are some studies that utilize multisignature (e.g., [34, 35]). However, we conclude that they do not satisfy Req. 1 because a CA that creates a multisignature may be able to deny a request based on a CSR.

9. Conclusions

In this paper, we define four requirements of a consortium CA and propose a distributed public key certificate-issuing infrastructure that can be cooperatively operated by multiple organizations. To achieve cooperative operation, the proposed infrastructure encloses CA's private keys in a distributed ledger and enforces the usage of them. We design the proposed infrastructure to meet four requirements and evaluate the fulfillment of those requirements.

In addition, we measured the basic performance of the proposed infrastructure: issuing, verifying, and revoking public key certificates. Through the experimental evaluation, we confirm that the proposed infrastructure can work stably. The proposed infrastructure can issue public key certificates with a processing time of approximately 4 ms and can check that public key certificates are issued by the proposed infrastructure with a processing time of approximately 3 ms.

Data Availability

The data used to support the findings of this study are included within the manuscript.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by JSPS KAKENHI grant no. JP22K17881.

References

[1] CA/Browser Forum, "About the baseline requirements," 2015, https://cabforum.org/about-the-baseline-requirements/.

[2] A. Rech, C. Steger, and M. Pistauer, "A decentralized service-platform towards cross-domain entitlement handling," in *Proceedings of the 2019 IEEE International Conference on Blockchain (Blockchain)*, pp. 455–462, Seoul, South Korea, May, 2019.

[3] K. M. Alam, A. Sopena, and A. El Saddik, "Design and development of a cloud based cyber-physical architecture for the internet-of-things," in *Proceedings of the 2015 IEEE International Symposium on Multimedia (ISM)*, pp. 459–464, Miami, FL, USA, December, 2015.

[4] K. M. Alam and A. El Saddik, "C2PS: a digital twin architecture reference model for the cloud-based cyber-physical systems," *IEEE Access*, vol. 5, pp. 2050–2062, 2017.

[5] R. Roberts and D. Levin, "When certificate transparency is too transparent: analyzing information leakage in HTTPS domain names," in *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, pp. 87–92, London, UK, November, 2019.

[6] Hyperledger-labs, "Hyperledger fabric private chaincode," 2020, https://github.com/hyperledger/fabric-private-chaincode/tree/v1.0-rc1.

[7] B. Laurie, E. Messeri, R. Stradling, E. Messeri, and R. Stradling, "Certificate transparency version 2.0," *RFC 9162*, 2021, https://datatracker.ietf.org/doc/rfc9162/bibtex/.

[8] ITU, "Information technology - open systems interconnection - the directory: public-key and attribute certificate frameworks," 2022, https://www.itu.int/rec/T-REC-X.509/.

[9] A. S. Wazan, R. Laborde, F. Barrère, A. Benzekri, and D. W. Chadwick, "PKI interoperability: still an issue? A solution in the X.509 realm," in *Proceedings of the World Conference on Information Security Education*, Lisbon, Portugal, June, 2009.

[10] Intel, "Intel software guard extensions," 2018, https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html.

[11] V. Costan and S. Devadas, "Intel SGX explained," in *Cryptology ePrint Archive*, 2016.

[12] M. Domb and G. Leshem, "Secured key generation and transmission, using intel-SGX and optical communications," in *Proceedings of the 2019 Third World Conference on Smart Trends in Systems Security and Sustainablity (WorldS4)*, pp. 357–362, London, UK, July, 2019.

[13] Hyperledger, "Hyperledger fabric," 2022, https://www.hyperledger.org/use/fabric.

[14] Hyperledger, "Hyperledger," 2021, https://www.hyperledger.org/.

[15] R. Skuratovskii and A. Kalenyk, "Multisignature with double threshold condition in the blockchain and its application to and strong keys generating," in *Cryptology ePrint Archive*, 2021.

[16] Y. Kaga, "A secure and practical signature scheme for blockchain based on biometrics," *Information Security Practice and Experience*, Springer, Berlin, Germany, pp. 877–891, 2017.

[17] O. Boireau, "Securing the blockchain against hackers," *Network Security*, vol. 2018, pp. 8–11, 2018.

[18] M. Brandenburger, C. Cachin, R. Kapitza, and A. Sorniotti, "Blockchain and trusted computing: problems, pitfalls, and a solution for hyperledger fabric," 2018, https://arxiv.org/abs/1805.08541.

[19] Fpc Team, "FPC without trusted ledger," 2017, https://docs.google.com/document/d/1jbiOY6Eq7OLpM_s3nb-4X4AJXROgfRHOrNLQDLxVnsc/edit#heading=h.sz7cg9d09f71.

[20] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet X. Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile," *RFC 5280*, 2008, https://datatracker.ietf.org/doc/rfc5280/bibtex/.

[21] A. Lei, Y. Cao, S. Bao et al., "A blockchain based certificate revocation scheme for vehicular communication systems," *Future Generation Computer Systems*, vol. 110, pp. 892–903, 2020.

[22] A. S. Ahmed and T. Aura, "Turning trust around: smart contract-assisted public key infrastructure," in *Proceedings of the 2018 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE)*, pp. 104–111, New York, NY, USA, August, 2018.

[23] S. Matsumoto and R. M. Reischuk, "IKP: turning a PKI around with decentralized automated incentives," in *Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP)*, pp. 410–426, San Jose, CA, USA, May, 2017.

[24] Y. C. Elloh Adja, B. Hammi, A. Serhrouchni, and S. Zeadally, "A blockchain-based certificate revocation management and status verification system," *Computers and Security*, vol. 104, Article ID 102209, 2021.

[25] J. Chen, S. Yao, Q. Yuan, K. He, S. Ji, and R. Du, "Certchain: public and efficient certificate audit based on blockchain for tls connections," in *Proceedings of the IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pp. 2060–2068, Honolulu, HI, USA, April, 2018.

[26] W. Wang, N. Hu, and X. Liu, "BlockCAM: a blockchain-based cross-domain authentication model," in *Proceedings of the 2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*, pp. 896–901, Guangdong, China, June, 2018.

[27] X. Boyen, U. Herath, M. McKague, and D. Stebila, "Associative blockchain for decentralized PKI transparency," *Cryptography*, vol. 5, no. 2, p. 14, 2021.

[28] B. Khieu and M. Moh, "CBPKI: cloud blockchain-based public key infrastructure," in *Proceedings of the 2019 ACM Southeast Conference*, pp. 58–63, Kennesaw, GA, USA, April, 2019.

[29] Z. Wang, J. Lin, Q. Cai, Q. Wang, D. Zha, and J. Jing, "Blockchain-based certificate transparency and revocation transparency," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 681–697, 2022.

[30] G.-H. Hwang, T.-K. Chang, and H.-W. Chiang, "A semi-decentralized PKI system based on public blockchains with automatic indemnification mechanism," *Security and Communication Networks*, vol. 2021, Article ID 7400466, 15 pages, 2021.

[31] M. Y. Kubilay, M. S. Kiraz, and H. A. Mantar, "CertLedger: a new PKI model with Certificate Transparency based on blockchain," *Computers and Security*, vol. 85, pp. 333–352, 2019.

[32] A. Yakubov, W. Shbair, A. Wallbom, D. Sanda, and R. State, "A blockchain-based PKI management framework," in *Proceedings of the First IEEE/IFIP International Workshop on Managing and Managed by Blockchain (Man2Block)*, Taipei, Taiwan, April, 2018.

[33] A. Rashid, A. Masood, H. Abbas, and Y. Zhang, "Blockchain-based public key infrastructure: a transparent digital certification mechanism for secure communication," *IEEE Network*, vol. 35, no. 5, pp. 220–225, 2021.

[34] T. Saleem, M. U. Janjua, M. Hassan et al., "ProofChain: an X.509-compatible blockchain-based PKI framework with decentralized trust," *Computer Networks*, vol. 213, Article ID 109069, 2022.

[35] L. Dykcik, L. Chuat, P. Szalachowski, and A. Perrig, "BlockPKI: an automated, resilient, and transparent public-key infrastructure," in *Proceedings of the 2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 105–114, Singapore, November. 2018.

[36] Y. Li, Y. Yu, C. Lou, N. Guizani, and L. Wang, "Decentralized public key infrastructures atop blockchain," *IEEE Network*, vol. 34, no. 6, pp. 133–139, 2020.

[37] M. T. Hammi, P. Bellot, and A. Serhrouchni, "BCTrust: a decentralized authentication blockchain-based mechanism," in *Proceedings of the 2018 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–6, Las Vegas, NV, USA, April, 2018.

[38] A. Yakubov, W. Shbair, and R. State, "BlockPGP: a blockchain-based framework for PGP key servers," in *Proceedings of the 2018 Sixth International Symposium on Computing and Networking Workshops (CANDARW)*, pp. 316–322, Takayama, Japan, November, 2018.

[39] M. Ali, J. Nelson, R. Shea, and M. J. Freedman, "Blockstack: a global naming and storage system secured by blockchains," in *Proceedings of the 2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pp. 181–194, Denver, CO, USA, June, 2016.

[40] A. Papageorgiou, A. Mygiakis, K. Loupos, and T. Krousarlis, "DPKI: a blockchain-based decentralized public key infrastructure system," in *Proceedings of the 2020 Global Internet of Things Summit (GIoTS)*, pp. 1–5, Dublin, Ireland, June, 2020.

[41] M. Al-Bassam, "SCPKI: a smart contract-based PKI and identity system," in *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, pp. 35–40, Incheon, Republic of Korea, June, 2017.

[42] C. Fromknecht, D. Velicanu, and S. Yakoubov, "Certcoin: a namecoin based decentralized authentication system," vol. 6Technical Report D, pp. 46–56, Massachusetts Inst. Technol, Cambridge, MA, USA, 2014.

[43] C. Patsonakis, K. Samari, M. Roussopoulos, and A. Kiayias, "Towards a smart contract-based, decentralized, public-key infrastructure," in *Cryptology and Network Security*, pp. 299–321, Springer, Berlin, Germany, 2018.

[44] C. Patsonakis, K. Samari, A. Kiayias, and M. Roussopoulos, "Implementing a smart contract PKI," *IEEE Transactions on Engineering Management*, vol. 67, no. 4, pp. 1425–1443, 2020.

[45] B. Qin, J. Huang, Q. Wang, X. Luo, B. Liang, and W. Shi, "Cecoin: a decentralized PKI mitigating MitM attacks," *Future Generation Computer Systems*, vol. 107, pp. 805–815, 2020.