IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON EMERGING APPROACHES TO CYBER SECURITY

Received June 22, 2020, accepted July 12, 2020, date of publication July 22, 2020, date of current version August 4, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3011137

Cross-Certification Towards Distributed Authentication Infrastructure: A Case of Hyperledger Fabric

SHOHEI KAKEI[1], YOSHIAKI SHIRAISHI[2,3], (Member, IEEE), MASAMI MOHRI[4], (Member, IEEE), TORU NAKAMURA[3], MASAYUKI HASHIMOTO[3], AND SHOICHI SAITO[1]

[1]Nagoya Institute of Technology, Nagoya 466-8555, Japan
[2]Department of Electrical and Electronic Engineering, Kobe University, Kobe 657-8501, Japan
[3]Advanced Telecommunications Research Institute International, Kyoto 619-0237, Japan
[4]Department of Electrical, Electronic and Computer Engineering, Gifu University, Gifu 501-1112, Japan

Corresponding author: Shohei Kakei (kakei.shohei@nitech.ac.jp)

This work was supported by Grants-in-Aid for Scientific Research, Japan Society for the Promotion of Science, Grant Number JP19K24342, JP19K11963, and JP18K04133.

ABSTRACT In Internet of Things ecosystems, where various entities trade data and data analysis results, public key infrastructure plays an important role in establishing trust relationships between these entities to specify who trusts whose private keys. The owner of a private key is provided with a public key certificate issued by a certificate authority (CA) representing a trusted third party. Although this certificate ensures the reliability of the ecosystem by verifying the data source and preventing the denial of trading, it often causes an overconcentration of trust in a particular CA. Consequently, if that CA is infringed, all the related trust relationships become compromised. The paper proposes a distributed authentication infrastructure called Meta-PKI that decentralizes such overconcentration via a cross-certification procedure performed by multiple CAs. Although cross-certification is capable of establishing mutual trust relationships, it does not evaluate the trustworthiness of other CAs in a standardized manner. Therefore, this paper also proposes a new cross-certification method using a distributed ledger technology for building trust relationships based on unified criteria. It also describes the implementation of a Meta-PKI system for Hyperledger Fabric as a proof of concept. Once trust relationships have been established, it takes approximately 65.7 ms to validate them using the proposed system, which is secure against CA takeover and spoofing by outsider attackers.

INDEX TERMS Blockchain, distributed authentication infrastructure, distributed ledger technology, Hyperledger fabric, public key infrastructure.

I. INTRODUCTION

Data acquisition environments have evolved with the development of Internet of Things (IoT) technologies. In the IoT ecosystem [1], [2], data owners, data analysists, and data consumers aim at coexistence and shared prosperity, and when data are traded openly between various entities using a certain platform, the shared prosperity is expected to facilitate further data usage.

To ensure the reliability of data trading, it is necessary to verify who trades what data, certify the data sources, and prevent the denial of trading. Therefore, ensuring the identity of a trading partner is a core task in developing a reliable data trading platform.

Public Key Infrastructure (PKI) [3] is mainly used to guarantee identities. It is an infrastructure in which a certificate authority (CA), which is a trusted third party, guarantees the relationship between an entity and a public key. It also contains a mechanism for verifying communication partners by digital signatures. Each CA must be prepared for insider and outsider threats by developing proper security measures and training human resources. Nevertheless, some incidents involving CA security breaches have occurred [4]–[6].

PKI trust relationships are often defined in accordance with a system's architecture. A trust point essentially mediates the trust relationship between two entities, which is extended

The associate editor coordinating the review of this manuscript and approving it for publication was Luis Javier Garcia Villalba.

when different trust points trust each other in any architecture [7]–[9]. A trust relationship is established as follows: if entity E_A trusts a CA, and the CA identifies entity E_B, then E_A trusts the identity of E_B. Furthermore, if another entity E_C is identified by the CA, the trust relationship with E_A is extended to E_C. Meanwhile, if the CA builds a trust relationship with another CA that identifies entity E_D, the trust relationship with E_A is extended to E_D across the trust point. Using this method, the trust points of many entities are determined by the CA.

In Internet server authentications, E_A is a service user, and E_B, E_C, and E_D are service providers. The user trusts the identities of the providers and uses their services. The user can also trust other service providers without prior interactions by trusting the related CA as a single trust point. When a malicious service provider accesses a CA's private key illegally and creates a public key certificate, it is difficult for the user to distinguish the malicious service provider from valid service providers. As a result, all the trust relationships established through this CA become compromised.

This paper proposes a distributed authentication infrastructure that decentralizes CA trust points so they are distributed among multiple service providers and connects them via cross-certification. To illustrate the proposed approach, a cross-certification method using a smart contract on Hyperledger Fabric [10] is described. A smart contract is a function that can read/write to a distributed ledger by following the procedures defined in the source code. The proposed method allows the creation of a unique framework that regulates cross-certification in a unified manner.

The remainder of this paper is organized as follows. Section II provides an overview of PKI and Hyperledger Fabric. Section III discusses previous PKI-related work using distributed ledger technology (DLT). Section IV presents the principles and issues of CA trust point distribution and describes the proposed distributed authentication infrastructure (Meta-PKI). Section V describes the Meta-PKI transactions and management of the trust relationships. The implementation of Meta-PKI is demonstrated in Section VI. Section VII discusses the security of Meta-PKI, and the conclusions and directions of future work are provided in Section VIII. The abbreviations used in this paper are listed in Table 1.

TABLE 1. List of abbreviations.

Abbreviation	Description
CA	Certificate Authority
CT	Certificate Transparency
DA	Data Analyst
DC	Data Consumer
DLT	Distributed Ledger Technology
DN	Distinguished Name
DO	Data Owner
EE	End Entity
IoT	Internet of Things
KVS	Key Value Store
PGP	Pretty Good Privacy
PKI	Public Key Infrastructure
RW-set	Read/Write-set
Tx	Transaction

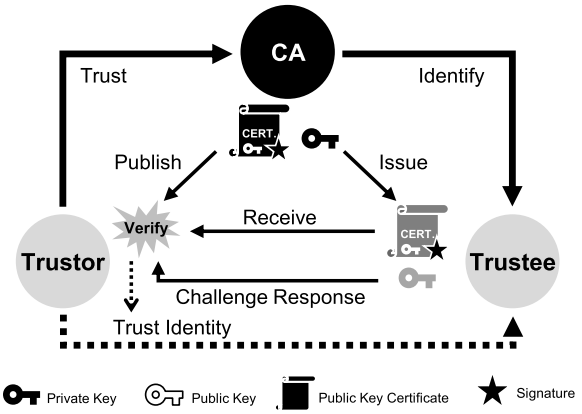

FIGURE 1. Trust relationships established by a CA.

II. BACKGROUND

A. PKI

1) OVERVIEW

Internet PKI [3] is an infrastructure in which a CA validates the relationship between an entity and a public key by issuing a public key certificate that contains the distinguished names (DNs) [11] of the entity. This certificate is mainly utilized to verify the identity of a server via HTTPS.

The trust relationships established by a CA are shown in Fig. 1. If the CA receives a certificate signing request [12], it confirms the identity of the trustee and creates a corresponding certificate. The CA guarantees the identity by signing the certificate with a CA private key. So that the trustor may validate the identity of the trustee, the trustor obtains the certificate and verifies its issuer. If the issuer is confirmed to be a trusted CA by the trustor and the trustee has a private key for that certificate, the trustor trusts the identity of the trustee. Challenge–response authentication protocols [13] are widely used methods for checking whether the trustee has a private key corresponding to a public key certificate.

Trust relationships are established by issuing certificates. In a large-scale PKI system, one CA builds trust relationships with another CA by issuing a certificate to its CA (this process is called *cross-certification*). Huang and Nicol [14] used first-order logic and presented the two main types of trust semantics: *trust in performance* and *trust in belief*. Inspired by Huang and Nicol [14], to represent the semantics of trust, this paper uses the logical operators defined in Table 2 in its formulas. Logical functors are also used and denoted as the corresponding logical operator with a dot above it, e.g., $\dot{\supset}$ is a function that mimics logical implication.

Trust in performance represents "trust in what the trustee performs," and can be denoted by $trust\_p(d, e, x, k)$. If information x is produced by trustee e, then trustor d believes x in

TABLE 2. List of logical operators for any formulas A, B.

Logical Operator	Description
$\supset$	**A** $\supset$ **B** denotes that **A** implies **B**.
$\wedge$	**A** $\wedge$ **B** denotes that it is true only if **A** is true and **B** is true.
$\neg$	$\neg$**A** denotes "not **A**."
$\equiv$	**A** $\equiv$ **B** denotes that **A** "is equivalent to" **B**.

the context of k. Formally,

$$trust\_p(d, e, x, k) \equiv madeBy(x, e, k) \supset believe(d, k \dot{\supset} x) \quad (1)$$

Trust in belief represents "trust in what the trustee believes," and can be denoted by $trust\_b(d, e, x, k)$. If information x is believed by trustee e, then trustor d believes x in the context of k. Formally,

$$trust\_b(d, e, x, k) \equiv believe\left(e, k \dot{\supset} x\right) \supset believe\left(d, k \dot{\supset} x\right). \quad (2)$$

In definitions (1) and (2), information x is a concrete (reified) proposition representing either an assertion made by e or a commitment made by e to perform (or not to perform) an action. Further, context k is a reified proposition representing the conjunction of a set of "propositions" that characterize a context. For our purposes, a reified proposition represents data, that is, x represents a public key certificate and k represents a set of issued and managed public key certificates. Specifically, we use these definitions in the context of issuing and managing public key certificates.

Uncertainty in trust, or *distrust*, is considered the negative form of trust. According to the formal semantics described in [14], *distrust* means that trustor d believes that x is false. Formally, *distrust in performance* and *distrust in belief* are defined as follows:

$$distrust\_p(d, e, x, k) \equiv madeBy(x, e, k) \supset believe(d, k \dot{\supset} \dot{\neg} x) \quad (3)$$

$$distrust\_b(d, e, x, k) \equiv believe(e, k \dot{\supset} x) \supset believe(d, k \dot{\supset} \dot{\neg} x) \quad (4)$$

The trust relationships established among the end entities (EEs), EE_A and EE_B, and CA are described using formal semantics in Fig. 2. Here, CA issues a certificate for EE_B's public key. EE_A believes the certificate because EE_A trusts the performance of CA in the context of "issuing and maintaining certificates." In general, one PKI system consists of multiple CAs, which can be arranged via either a hierarchical model or a mesh model (Fig. 3) and build trust relationships with each other. Even if a trustor cannot trust the CA that is a trustee's trust anchor, it can trust the CA's performance by inferring indirect trust propagated through *trust in belief,* as follows:

$$trust\_b(a, b, x, k) \wedge trust\_p(b, c, x, k) \supset trust\_p(a, c, x, k) \quad (5)$$

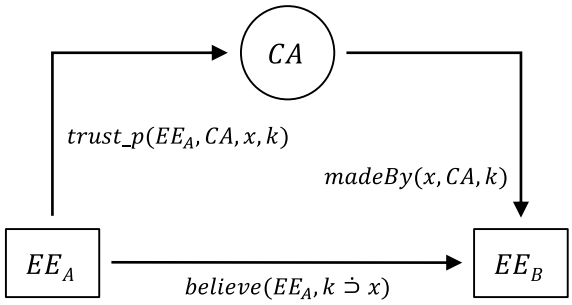

FIGURE 2. Simplified representation of the semantics of trust in PKI (EE_A believes EE_B's public key x issued by CA).

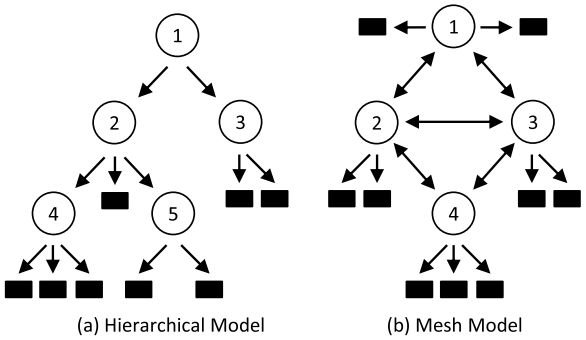

FIGURE 3. Schematics of the hierarchical and mesh models.

$$trust\_b(a, b, x, k) \wedge trust\_b(b, c, x, k) \supset trust\_b(a, c, x, k) \quad (6)$$

With the above trust reasoning, an EE can trust other CAs that its CA trusts in performance. *Trust in belief* is transitive, but *trust in performance* by itself is not transitive. For example, a patient trusts what his/her doctor believes about healthcare and the doctor trusts the efficacy of drugs; therefore, the patient indirectly trusts the efficacy of the drugs. Moreover, friends of the patient may indirectly trust the efficacy of the drugs.

2) COMPARISON BETWEEN THE HIERARCHICAL AND MESH MODELS

Table 3 lists the comparative parameters of the hierarchical and mesh models. First, the hierarchical model can expand trust relationships to all CAs simply by establishing trust relationships with the root CA because the root CA represents a single trust point. In contrast, trust relationships must be individually established in the mesh model because the mesh model does not contain a single trust point. Second, because the hierarchical model has a single trust point, its dependency on this point is stronger than that of the mesh model. Consequently, the hierarchical model can lower the reliability of the entire system to a greater extent than the mesh model.

On the Internet, most CAs are created using the hierarchical model because it enables trust relationships to be easily expanded. Owing to vendor support, some public key certificates of major root CAs are installed in primary Internet browsers, and a server keeps a public key certificate that is

TABLE 3. Parameters of the hierarchical and mesh models.

	Hierarchical Model	Mesh Model
Expandability of trust relationships	High	Low
Dependency on a trust point	High	Low

issued by a CA that comes under the major root CA that is at the top of the hierarchy. Consequently, a client can establish a trust relationship with the server.

B. HYPERLEDGER FABRIC

Hyperledger Fabric [10] is a platform for a distributed ledger framework. The ledger conducts data operations in a key–value format without a specific administrator and is managed by multiple peers; the peers have ledgers with the same content and form a distributed ledger network. Data are written to the ledger after an agreement that cannot be changed later is reached among all peers.

The data written to the ledger are managed by a set of units called a block. The block has a chain structure with a hash value computed from the previous block using a cryptographic hash function. For this reason, block tampering is detected as a disruption in the chain structure due to the mismatch of the subsequent hash value.

Fig. 4 shows the Hyperledger Fabric scheme. All peers in the distributed ledger network are connected to each other. Each peer has a distributed ledger with the same content and multiple chaincodes CC, which are also known as smart contracts. The distributed ledger consists of a blockchain and a world state. The blockchain has a journal of transactions that records read/write values (RW-set) to the ledger with a key, and the world state is a key–value store with all values versioned for each transaction.

The blockchain consists of multiple blocks arranged in a particular order. Block B_n contains the hash value H_{n-1} of the previous block B_{n-1} as well as that of block-data D_n and signature S_n created by a block generator. Data D_n include the transactions $T_n = \{T_n^1, T_n^2, \ldots\}$ that connect the RW-set to the world state. A read/write value in any given transaction cannot be arbitrarily determined.

The client requests a peer to generate T by specifying CC. The chaincode defines the operations performed on the distributed ledger network and reads/writes to the ledger to prevent writing arbitrary data.

Fig. 5 shows a flowchart of the Hyperledger Fabric transaction flow. The steps are as follows.

Step 1. **The client requests the execution of a chaincode.** In the distributed ledger network, the peers responsible for the execution of a chaincode (called endorsers) are determined when the chaincode is registered in the network. During the execution request, the client sends arguments and the chaincode name to the endorsers.

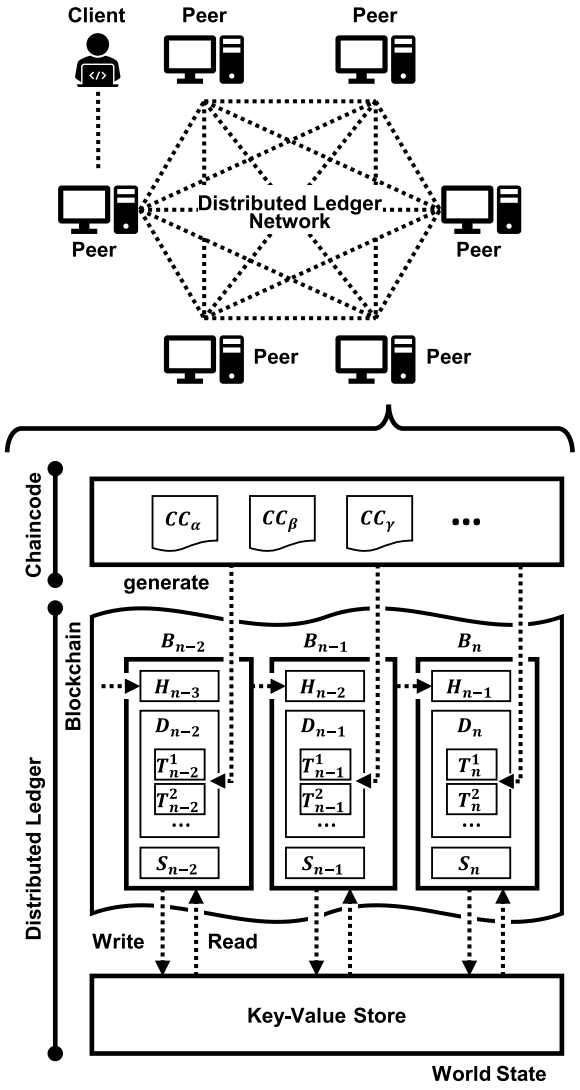

FIGURE 4. Description of Hyperledger Fabric.

Step 2. **The endorsers execute a chaincode.** Each endorser executes the chaincode specified by the client with the arguments and creates an RW-set signed with its private key. This signature serves as evidence that the endorser has executed the chaincode. Finally, the endorser sends the signature to the client.

Step 3. **The client collects the signatures.** To create a transaction, the client must collect signatures from a sufficient number of endorsers. The total number of signatures is specified at the time of the chaincode registration. For this reason, the client performs steps 1 and 2 for the endorsers. Finally, the client creates a transaction with the RW-set and the signatures.

Step 4. **The client sends the transaction to an orderer.** The client requests a peer called an orderer to reflect the transaction in the ledger. The orderer collects multiple transactions, arranges batches of the submitted transactions into a well-defined sequence,

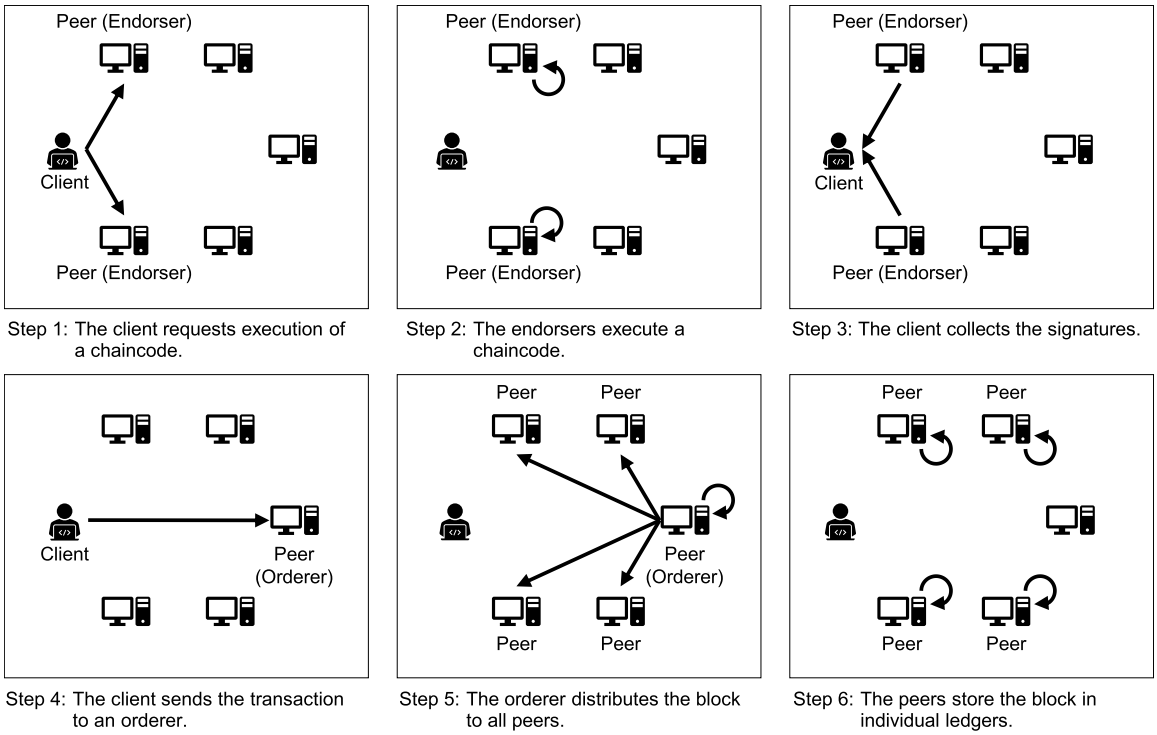

FIGURE 5. Transaction flow in Hyperledger Fabric.

and packages them into a block. The number of transactions in a block depends on either the block size or block generation interval.

Step 5. **The orderer distributes the block to all peers.** Transactions may conflict with each other because the orderer receives requests from multiple clients in parallel. The orderer checks all the RW-sets in the transactions and stores the block to its ledger while determining invalid transactions. The invalid transactions are marked as invalid. After the block is finalized, the orderer distributes the block to all peers.

Step 6. **The peers store the block to the individual ledger.** To prevent unauthorized writing to the ledger, the peers check whether the transactions have been signed by the endorsers and the block has been signed by the orderer. Invalid transactions are ignored and valid transactions are stored in individual ledgers whose contents are identical for all peers.

III. RELATED WORK

In the current PKI model, if the trusted third party fails, the entire system will be affected. Hence, several DLT-based PKI systems have been proposed to distribute this risk of failure. These types of systems are divided into two main groups: PKI operated by DLT and PKI enhanced by DLT.

A. PKI OPERATED BY DLT

In this type of PKI system, the public key lifecycle is managed by DLT. The public key lifecycle management process (including registration, revocation, and validation stages) is generally performed by either a trusted third party (the CA model) or a public key owner (the pretty good privacy (PGP) model) [15].

1) CA MODEL

In the CA model, a CA is the trusted third party that performs the lifecycle management process. As described in Section II, a CA issues a public key certificate, which confirms the relationship between the public key owner's domain and the owner's public key. In essence, the CA's private key is controlled by the CA. Under this condition, a certificate user believes the certificate owner if the corresponding public key certificate is signed by the CA. Thus, certificate users can believe all certificate owners just by trusting the corresponding CAs. Meanwhile, attackers can deceive all certificate users by obtaining a CA private key, which is the main reason why a CA is a single point of failure.

CA models that use DLT have been proposed. For instance, Al-Bassam [16] proposed a smart contract-based PKI that is a decentralized PKI and identity management system. In that system, the owner is represented by an Ethereum address with three data fields: *Attribute*, *Signature,* and *Revocation*. *Attribute* stores the owner's PGP key, and *Signature* has a signature of *Attribute* signed by the owner. When revoking a signature, the owner creates a *Revocation* field that contains its identification.

In addition, Qin *et al.* [17] developed a distributed blockchain-based PKI named Cecoin that is based on

Bitcoin. Here, distributed nodes in a P2P network manage public key certificates with domain names using a decentralized certificate library based on the Merkle Patricia trie.

2) PGP MODEL

In the PGP model, a public key owner performs the lifecycle management process. This model does not contain a trusted third party such as CA, and each user in the system manages key pairs by himself (herself). When building trust relationships, a user takes the public key of another user and verifies whether it is owned by a legitimate user because a trusted third party does not certify the owner of a public key. Validation methods for checking the owners of public keys include *reliable transportation measures* and the *Web-of-Trust*. Reliable transportation measures for receiving public keys could include face-to-face contact, e-mail exchange, or cloud storage. In contrast, if a user does not have reliable transportation measures, he or she can obtain a valid public key with the help of acquaintances (this method is called the Web-of-Trust). As an illustration, we consider a situation in which Alice is not acquainted with Carol but wants to verify Carol's public key. At the same time, Alice is acquainted with Bob, who is acquainted with Carol. Hence, Bob signs Carol's public key with his private key. After Alice has successfully verified the authenticity of Carol's public key, she recognizes it. The PGP model is a decentralized authentication model because each user checks the validity of public keys without depending on a trusted third party. Therefore, the costs of key distribution in this model are higher than those in the CA model. A PGP keyserver can be used to distribute public keys and signatures; however, it may also become a single point of failure.

Yakubov *et al.* [18] reported that a public key cannot be obtained when the keyserver has failed, and a fake public key can be returned instead during a man-in-the-middle attack. The authors proposed a blockchain-based framework (BlockPGP) to provide reliable management for OpenPGP certificates and the keyserver infrastructure.

Hammi *et al.* [19] proposed a decentralized blockchain-based authentication system for IoT. The key concept of their system is a *bubble*, which consists of one Master node and several Follower nodes. Each of these nodes corresponds to an IoT device, which authenticates and communicates with other devices in different bubbles.

B. PKI ENHANCED BY DLT

As described in Section II, the hierarchical PKI model is widely used on the Internet. To take advantage of the existing PKI ecosystem, various methods have been proposed to enhance the CA functionality using DLT instead of replacing the CAs with DLT.

Conventionally, a CA only issues a public key certificate according to the applicant's request. Therefore, only the entities involved know that this certificate exists. This approach can enable a fraudulent certificate to be issued for the following reasons:

(1) A malicious applicant can request a certificate that is similar to an actual certificate because the certificate is identified by a DN. For example, a server (an applicant) can apply for a public key certificate that has a confusing Common Name in the DN (e.g., examp1e.com where the letter "l" is replaced by the digit "1"). This strategy can be utilized in a homograph attack [20].

(2) A compromised CA can issue a certificate without undergoing regular procedures. When an attacker obtains a CA's private key, he or she can create a certificate that contains an impersonating DN.

If a fraudulent certificate is issued legitimately, it is difficult to identify it as a false certificate. For this reason, certificate transparency (CT) [21] was proposed as a method for increasing the transparency of issuing public key certificates. In CT, a log server guarantees the transparency of issued certificates by creating publicly auditable logs. Public key certificate owners and CA administrators can verify that the certificates were not issued illegally by checking the logs. However, the log server in this scheme is a single point of failure. Hence, CT schemes based on DLT have been proposed.

Madala *et al.* [22] proposed a CT using blockchain that utilizes a private blockchain (Hyperledger Fabric). In their method, a CA, which is a member of the blockchain network, issues a public key certificate and stores it in a distributed ledger along with a revocation state (revoked/not revoked). The distributed ledger distributes the certificate to all peers and guarantees that it cannot be tampered with.

Kubilay *et al.* [23] proposed CertLedger, which is a PKI architecture with a CT based on a public blockchain. CT can improve the transparency of certificates, but an adversary can create a malicious log to provide evidence that unauthorized certificates are contained in it (the *split-world attack* [24]). Furthermore, CT only ensures the transparency of issuing certificates, not the transparency of revocation. To resolve these issues, CertLedger validates, stores, and revokes certificates using a public blockchain. During the issuance process, a trusted CA manages the CertLedger certificates. In the revocation process, the domain owner can revoke certificates instead of CAs.

Yao *et al.* [25] developed a privacy-preserving blockchain-based certificate status validation scheme (PBCert). It stores certificate operations in a blockchain to support public audits and efficient revocation checking, which uses the online certificate status protocol. A CA, which may be an honest-but-curious entity, can track the server accessed by the user by checking whether the server certificate has been revoked or not. PBCert preserves a client's privacy by impeding this operation using a shortened hash value and bloom filters.

C. SUMMARY OF RELATED WORK

The conventional PKI models require a trusted third party as a trust anchor to efficiently extend the trust relationships

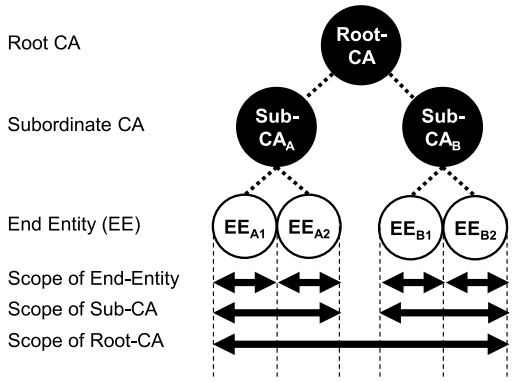

FIGURE 6. Scopes of various entities in the hierarchical PKI model.

established with others. In the PGP model, the user evaluates the trustworthiness of other entities without relying on a trusted third party, instead establishing trust relationships with them directly. Although a trusted third party is not required for building trust relationships, it is effectively utilized for public key distribution and thus represents a single point of failure. To eliminate such a point, decentralized PKI models including PKI operated by DLT and PKI enhanced by DLT have been proposed.

In contrast, this study proposes a new cross-certification PKI system enhanced by DLT. In this model, each CA evaluates the trustworthiness of other CAs to establish trust relationships with them. The proposed system uses smart contracts to write rules for trustworthiness evaluation and formulates unified trustworthiness criteria without a single point of trust.

IV. DISTRIBUTED AUTHENTICATION INFRASTRUCTURE: META-PKI

A. MAIN PRINCIPLE

In the hierarchical PKI model, shown in Fig. 6, the root CA issues certificates to subordinate CAs, and the subordinate CAs issue certificates to EEs. Because CAs are trust points responsible for issuing and managing certificates, the scope of the responsibility expands in a chain as the hierarchy deepens. Consequently, all trust points are concentrated at the root CA. We call this the *overconcentration of trust*.

Matsumoto *et al.* [26] proposed two important properties: *scoped authority* and *global authentication*. These properties enable EEs who trust each other to communicate even if they are located in separate domains. Every domain has a trust anchor that is trusted by all the EEs in that domain, and that trust anchor establishes a trust relationship with another trust anchor in another domain. Thus, the authority of the trust point can be limited within each domain while enabling communication between domains.

In [26], the domains can represent groups of various scales, such as companies, conglomerates, or countries, but overconcentration of trust will occur in groups where PKI systems are built using the hierarchical model. To prevent the overconcentration of trust, we build trust points at service levels such that the trust points do not span multiple services. In the proposed method, a root CA is operated by a service provider.

B. PROBLEM

If each service provider becomes a trust point and establishes mutual trust relationships with other service providers by cross-certification, the concentration of trust points at a certain root CA can be prevented. In cross-certification, a CA evaluates the trustworthiness of other CAs and issues public key certificates with the private key of the issuer. A verifier can check the trust relationships established between various CAs by verifying the public key certificate. Because the trustworthiness of cross-certification is based on the evaluation process, the trustworthiness of all trust relationships cannot be guaranteed just by cross-certification according to the expert criteria of each service provider.

In this study, we construct a framework for the cross-certification procedure and propose a suitable cross-certification method without expert knowledge by automating its main steps.

C. OVERVIEW OF META-PKI

Fig. 7 shows the Meta-PKI scheme, which consists of three layers: an EE layer, a CA layer, and a Meta-CA (mCA) layer.

- The EE layer contains a group of EEs who request public key certificates from the CAs.
- The CA layer consists of a group of CAs generated by the Meta-CA. This layer guarantees the identities of the EEs and includes their trust point.
- The mCA layer is connected to the distributed ledger network and consists of a group of Meta-CAs that perform cross-certification using a chaincode. The Meta-CA establishes a trust relationship with other Meta-CAs, and this layer includes a trust point for CAs. The founder of the Meta-PKI system defines the operational policy framework and chaincodes so that the trust relationship is built at the policy level. Moreover, each Meta-CA sets an operational policy according to the framework. The policy level is represented by a decimal value in the range from 0.0 to 1.0. The proposed method calls this value a ''score,'' and the Meta-CA trusts another Meta-CA that has a higher score.

As shown in Fig. 7, the public key certificate owned by a Meta-CA and a policy specification are stored in the distributed ledger. A Meta-CA owns one self-signed public key certificate *mCaCert*, generates public key certificates *CaCert* for subordinate CAs, and issues public key certificates *CrossCert* to other Meta-CAs for cross-certification. The obtained certificates are stored in the world state of the distributed ledger with the identity ID_{mCA} of the mCA. Moreover, the world state manages the policy of Meta-PKI specifications using *policy_spec* as a key. The policy defines scores for various Meta-PKI parameters such as types of the cryptographic algorithm.

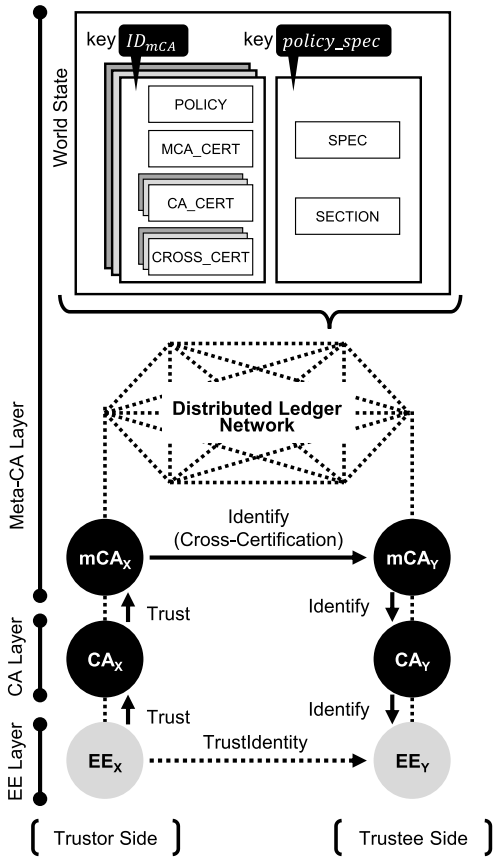

FIGURE 7. Meta-PKI scheme.

If a CA can issue a certificate without restrictions, certification paths may become more complex [27]. To automate the cross-certification procedure, the certification paths should be as simple as possible. In the proposed system, an mCA performs cross-certification and issues certificates only to the subordinate CAs that are located in the same domain, as shown in Fig. 7.

V. SCORE BASED CROSS-CERTIFICATION USING HYPERLEDGER FABRIC

A. META-PKI TRANSACTIONS

Meta-PKI defines eight transactions that primarily consist of cross-certifications and trust relationship verifications. Table 4 lists the data and keys stored in the ledger when a transaction is performed. In the proposed method, the specifications of the Meta-CA operational policy are stored using policy_spec as a key, and the Meta-CA information is stored under the identification name *ID* of the Meta-CA in a hierarchical format.

The entities in the following transaction descriptions use the notations shown in Fig. 7. Let E_X be an entity E identified by X, and let V_E be a variable V identified by E. In the entity representation, let mCA be a Meta-CA, and in the variable representation, let $Cert^{E_A}_{E_B}$ be a public key certificate specifying that E_A issues $Cert$ to E_B.

TABLE 4. Keys and values stored in the distributed ledger.

Tx	Key	Value
INI-Tx	policy_spec	SPEC: MCA_CERT: “CERTIFICATE” CA_CERT: “CERTIFICATE” SECTION: CERTIFICATE: SIGN_ALGO: MD5_WITH_RSA: 0 SHA1_WITH_RSA: 0.5 SHA256_WITH_RSA: 0.8 SHA512_WITH_RSA: 1.0 KEY_ALGO: RSA_512: 0 RSA_1024: 0.25 RSA_2048: 0.8 RSA_4096: 1.0 VALIDITY_PERIOD: 1440: 0.2 720: 0.4 360: 0.6 180: 0.8 90: 1.0
PAR-Tx	ID_{mCA}	POLICY: MCA_CERT: SIGN_ALGO: “SHA512_WITH_RSA” KEY_ALGO: “RSA_2048” VALIDITI_PERIOD: 90 CA_CERT: SIGN_ALGO: “SHA512_WITH_RSA” KEY_ALGO: “RSA_2048” VALIDITI_PERIOD: 90 MCA_CERT: $Cert^{mCA}_{mCA}$
UPD-Tx	ID_{mCA}	POLICY: MCA_CERT: SIGN_ALGO: “SHA512_WITH_RSA” KEY_ALGO: “RSA_4096” VALIDITI_PERIOD: 90 CA_CERT: SIGN_ALGO: “SHA512_WITH_RSA” KEY_ALGO: “RSA_4096” VALIDITI_PERIOD: 90 MCA_CERT: <...>
CON-Tx	ID_{mCA_X}	CROSS_CERT: ID_{mCA_Y}: CERT: $Cert^{mCA_X}_{mCA_Y}$
DES-Tx	ID_{mCA_X}	CROSS_CERT: ID_{mCA}: CERT: <...> DESTRUCTED: <date>
DEP-Tx	ID_{mCA}	CA_CERT: ID_{CA}: CERT: $Cert^{mCA}_{CA}$
ELI-Tx	ID_{mCA}	CA_CERT: ID_{CA}: CERT: <...> ELIMINATED: <date>
VAL-Tx	-	-

1) INITIALIZATION TRANSACTION (INI-TX)

A Meta-CA initializes the Meta-PKI system with INI-Tx. In this study, this Meta-CA is called the ‘‘initiator’’ to distinguish it from other Meta-CAs.

The founder defines the operational policy of all Meta-CAs used in the Meta-PKI system, and the initiator stores the policy in the ledger using policy_spec as a key. In Table 4, the policy utilized for a public key certificate is defined in the CERTIFICATE section. The public key certificate policies

of the Meta-CA and CA are stored in the **MCA_CERT** and **CA_CERT** subsections of the **POLICY** section, respectively, using the ID_{mCA} key.

The **CERTIFICATE** section consists of the signature algorithm (**SIGN_ALGO**), key algorithm (**KEY_ALGO**), and validity period (**VALIDITY_PERIOD**) of the public key certificate. The initiator assigns a score value to each policy in the range from 0.0 to 1.0. The **SPEC** and **SECTION** sections are defined depending on the Meta-PKI application.

2) PARTICIPATION TRANSACTION (PAR-TX)

The Meta-CA participates in the Meta-PKI system via PAR-Tx. It sets $Policy_{mCA}$ and generates *mCaCert* represented by $Cert_{mCA}^{mCA}$. $Policy_{mCA}$ and $Cert_{mCA}^{mCA}$ are stored in the ledger using ID_{mCA} as a key (Table 4). If the same *ID* is used, the transaction will fail.

3) DEPLOYMENT TRANSACTION (DEP-TX)

The Meta-CA deploys a CA via DEP-Tx. It generates a key pair of the CA and *CaCert* represented by $Cert_{CA}^{mCA}$, and adds ID_{CA} and $Cert_{CA}^{mCA}$ in the **CA_CERT** section with the key ID_{mCA} to the ledger, as shown in Table 4. The CA deployed by this transaction is a general CA that issues a public key certificate to the EEs via the protocols defined in [3].

4) ELIMINATION TRANSACTION (ELI-TX)

The Meta-CA eliminates a specified CA from the Meta-CA. ELI-Tx does not delete *CaCert* and adds the date of the CA elimination in the ID_{CA} subsection of the **CA_CERT** section with the key ID_{mCA} to the ledger, as shown in Table 4.

5) CONSTRUCTION TRANSACTION (CON-TX)

The Meta-CA on the trustor side (mCA_X) builds a trust relationship with the Meta-CA on the trustee side (mCA_Y) via CON-Tx. First, mCA_X computes scores of both Meta-CAs ($Score_{mCA_X}, Score_{mCA_Y}$). If $Score_{mCA_Y}$ is greater than or equal to $Score_{mCA_X}$, mCA_X generates a cross-certificate *CrossCert* represented by $Cert_{mCA_Y}^{mCA_X}$ for mCA_Y. Subsequently, mCA_X executes CON-Tx to add $Cert_{mCA_Y}^{mCA_X}$ to the ledger. In CON-Tx, both scores are recomputed, and ID_{mCA_Y} and $Cert_{mCA_Y}^{mCA_X}$ are added to the **CROSS_CERT** section with the ID_{mCA_X} key (see Table 4) after confirming that $Score_{mCA_Y}$ is greater than or equal to $Score_{mCA_X}$ in CON-Tx.

Here, *Score* is computed recursively from the weighted average in the POLICY section using the ID_{mCA} key. It is a decimal value in the range from 0.0 to 1.0.

6) DESTRUCTION TRANSACTION (DES-TX)

DES-Tx uses mCA_X to destroy the trust relationship established with a specified *mCA*. DES-Tx does not delete the *CrossCert* certificate and adds the date of the destruction of the trust relationship in the ID_{mCA} subsection of the **CROSS_CERT** section with the key ID_{mCA_X} to the ledger, as shown in Table 4.

7) UPDATE TRANSACTION (UPD-TX)

The administrator of Meta-CA executes UPD-Tx when updating $Policy_{mCA}$. UPD-Tx needs the new $Policy_{mCA}$ that will replace the old $Policy_{mCA}$ and its signature. To avoid unauthorized updates, the signature has to be signed with the private key corresponding to $Cert_{mCA}^{mCA}$.

8) VALIDATION TRANSACTION (VAL-TX)

VAL-Tx validates the trust relationship built between two *EE* s. We describe our validation procedure using Fig. 8 as an example. In other words, we explain how *believe* $(EE_X, k\dot{\supset}p)$ is satisfied, where k is the context of "issuing and maintaining certificates," and p is the performance metric characterizing the issuance and maintenance of EE_Y's certificate by its trust anchor. Here, CA_{Y2} is EE_Y's trust anchor, and mCA_Y is CA_{Y2}'s trust anchor. In this case, the certificate chain becomes mCA_X–mCA_Y–CA_{Y2}.

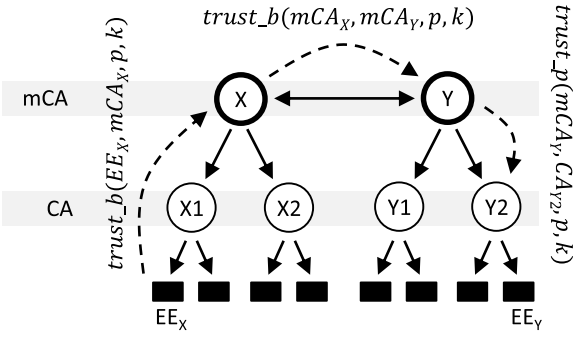

FIGURE 8. Trust relationships established between various entities from EE_X to EE_Y in Meta-PKI.

The trust relationship between EE_X and CA_{Y2} can be derived as follows:

$$\begin{aligned} &trust\_b(EE_X, mCA_X, p, k) \\ &\wedge\ trust\_b\,(mCA_X, mCA_Y, p, k) \\ &\supset\ trust\_b(EE_X, mCA_Y, p, k) \end{aligned} \tag{7}$$

and

$$\begin{aligned} &trust\_b(EE_X, mCA_Y, p, k) \\ &\wedge\ trust\_p\,(mCA_Y, CA_{Y2}, p, k) \\ &\supset\ trust\_p(EE_X, CA_{Y2}, p, k). \end{aligned} \tag{8}$$

As a result, the following trust relationship between EE_X and EE_Y is inferred.

$$\begin{aligned} madeBy\,(p, CA_{Y2}, k)\ \wedge\ trust\_p(EE_X, CA_{Y2}, p, k) \\ \supset\ believe(EE_X, k\dot{\supset}p) \end{aligned} \tag{9}$$

According to this reasoning, $Cert_{EE_Y}^{CA_{Y2}}$ is verified through the certification path from EE_X's root CA to EE_Y. This work performs certificate verification via the formula Verify_Cert(a, b), where a is the public key certificate to be verified, and b is the public key certificate used in the verification. It also verifies DEP-Tx via the function

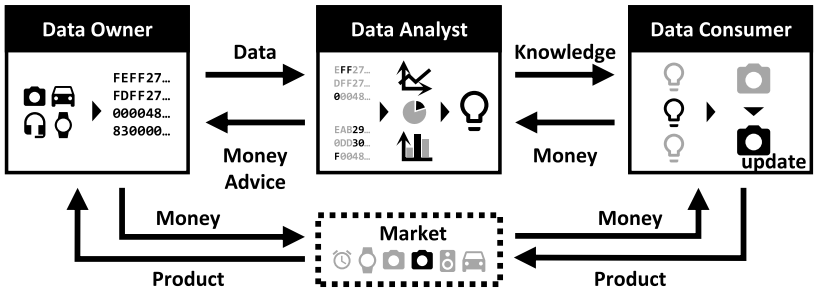

FIGURE 9. IoT ecosystem described in [1].

Verify_DepTx($Cert_B^A$), which checks that $Cert_B^A$ is stored in the ID_B subsection of the CA_CERT section with the key ID_A.

First, the issuance of $Cert_{EE_X}^{CA_{X1}}$ by CA_{X1} is verified as

$$\text{Verify\_Cert}\left(Cert_{EE_X}^{CA_{X1}}, Cert_{CA_{X1}}^{mCA_X}\right). \tag{10}$$

Second, the addition of $Cert_{CA_{X1}}^{mCA_X}$ to the ledger by mCA_X through DEP-Tx is checked using

$$\text{Verify\_DepTx}\left(Cert_{CA_{X1}}^{mCA_X}\right). \tag{11}$$

Third, the issuance of $Cert_{CA_{X1}}^{mCA_X}$ by mCA_X is verified as follows:

$$\text{Verify\_Cert}(Cert_{CA_{X1}}^{mCA_X}, Cert_{mCA_X}^{mCA_X}). \tag{12}$$

Fourth, the establishment of a trust relationship between mCA_X and mCA_Y through CON-Tx is verified as

$$\text{Verify\_Cert}\left(Cert_{mCA_Y}^{mCA_X}, Cert_{mCA_X}^{mCA_X}\right), \tag{13}$$

$$\text{Verify\_Cert}\left(Cert_{mCA_Y}^{mCA_Y}, Cert_{mCA_Y}^{mCA_X}\right). \tag{14}$$

Finally, the issuance of $Cert_{CA_{Y2}}^{mCA_Y}$ by mCA_Y is verified as follows:

$$\text{Verify\_Cert}\left(Cert_{CA_{Y2}}^{mCA_Y}, Cert_{mCA_Y}^{mCA_Y}\right) \tag{15}$$

and

$$\text{Verify\_DepTx}\left(Cert_{CA_{Y2}}^{mCA_Y}\right). \tag{16}$$

As a result, the trust relationship can be derived by verifying that $Cert_{EE_Y}^{CA_{Y2}}$ is issued by CA_{Y2} as follows:

$$\text{Verify\_Cert}\left(Cert_{EE_Y}^{CA_{Y2}}, Cert_{CA_{Y2}}^{mCA_Y}\right). \tag{17}$$

VAL-Tx returns "success" if the entire verification procedure is successful. No data are stored in the ledger via VAL-Tx.

B. SERVICE PROVIDERS AND TRUST RELATIONSHIPS

In the Meta-PKI system, a service provider that operates an mCA can establish the trust relationship with another service provider that has a higher score. If the score is lower, the trust relationship is not established. This score acts as a barrier that blocks trust relationships with unreliable service providers. However, because a service provider is responsible for the availability of a service, it is not possible to simply destroy trust relationships even if the service provider's score is no longer sufficient. Depending on the type of service, it is necessary to determine whether the trust relationships should be destroyed.

As an example of service provider collaboration, we consider IoT ecosystems for data trading. Some data trading ecosystems, such as those of [28], [29], are incorporating Blockchain and IoT. Fig. 9 shows an IoT ecosystem that is based on [1]. The owner of the IoT devices is a data owner (DO) who owns the data generated by its devices. The DO sends the data to a data analyst (DA). The DA analyses the provided data and returns the results. Moreover, the DA sells the new knowledge obtained from all the provided data to a data consumer (DC). The DC develops new products using this knowledge.

The DO, DA, and DC are considered domains and various competitors deploy services in each domain. In this scenario, the trust relationships are established from the DA to the DO and from the DC to the DA. For instance, let us suppose that the DA establishes trust relationships with many DOs to collect large amounts of data, and the DC establishes trust relationships with the selected DAs to obtain good quality knowledge. Further, suppose that the DA's operational policy is being updated for security. Because the DA has established trust relationships with many DOs, service availability is ensured even if the DA automatically destroys trust relationships with some DOs with lower scores. In contrast, it may be difficult for the DC to destroy trust relationships, even if the DA's score is lower than that of the DC. In this case, the administrator has to manually maintain the trust relationships.

In both these situations, both the DA and DC need a mechanism for detecting score updates so they can react quickly. In Hyperledger Fabric, such a mechanism, which is called an *Event* [30], is available.

In addition to operational policy update, it is possible to update the trust relationships by changing the service architecture, which is *split* or *merger*. If one service is split into multiple services, new trust relationships have to be established depending on the services to be used. If multiple services are merged into one service, unnecessary trust

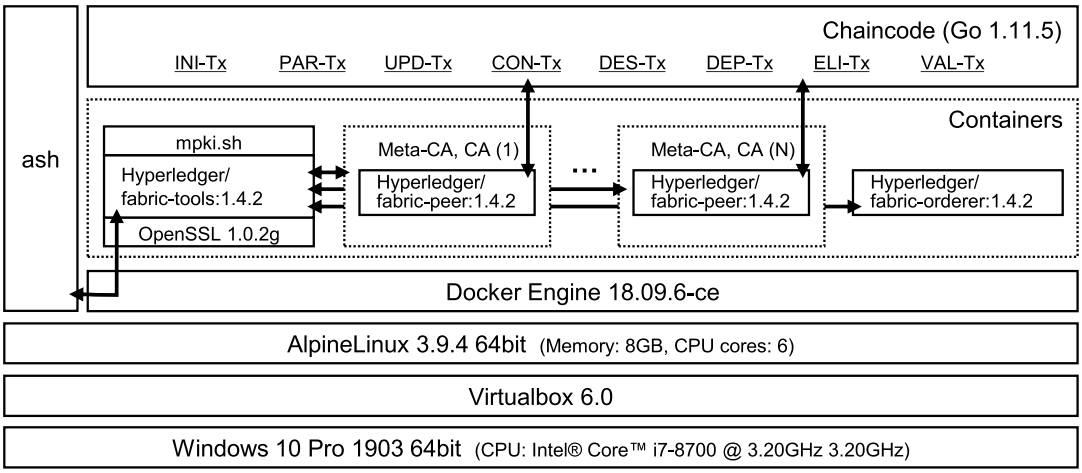

FIGURE 10. Meta-PKI system architecture.

relationships must be destroyed. These two cases should be handled manually rather than automatically; service administrators should control the trust relationships using the Meta-PKI transactions while taking relationships with other services into account.

VI. IMPLEMENTATION

A. OVERVIEW

Fig. 10 shows the Meta-PKI system architecture. In this study, it is implemented on a single machine running an Alpine Linux v3.9.4 operating system [31] using the Docker v18.09.6-ce platform [32]. This machine was installed on VirtualBox v6.0 [33], which ran on a Windows 10 machine.

This system uses the sample implementation of Hyperledger Fabric [34], which includes the ''fabric tools,'' ''fabric orderer,'' and ''fabric peer'' containers. The roles of these containers are as follows. ''Fabric tools'' is a utility that controls the distributed ledger network and performs the installation and execution of chaincodes. This container also has the shell script ''mpki.sh,'' which generates public key certificates with OpenSSL [35] for all Meta-CAs. ''Fabric orderer'' performs an ordering service, and ''fabric peer'' performs a peer service. The Meta-CAs and CAs were implemented using fabric peer containers.

The number of fabric peer containers can be changed by varying the system configuration. Such containers are usually deployed on multiple machines operated by different service providers. A user logs in directly to the guest operating system with the Almquist shell (ash) [36], which is the default shell of Alpine Linux and controls the fabric tools container.

The Meta-PKI chaincode was implemented using the Go programming language [37]. It was implemented as a single executable file, and its transactions are selected by arguments. Because data are stored in the ledger in a hierarchical format, this system uses the JSON format [38].

B. EVALUATION

1) CHAINCODE SECURITY

A chaincode can be written in a general-purpose programing language, such as Golang, NodeJS, or Java. Such languages contain a wide array of existing source code that allows developers to implement chaincodes more easily. However, these languages were not originally designed for writing chaincodes, which may lead to some challenges. As described in Section II, because each transaction originates from the multiple RW-sets generated by endorsers, a chaincode must be able to consistently produce the same RW-set. If different RW-sets are generated, the transaction will fail.

Yamashita *et al.* [39] identified 14 potential risks resulting from the nondeterminism of the Hyperledger Fabric chaincode written in Golang. These risks are caused by the language instructions, access outside the Blockchain, and other reasons. Table 5 lists the potential risks and their effects on the proposed system. They can be divided into four categories: 1) nondeterminism arising from the language instructions, 2) nondeterminism arising from the access outside the blockchain, 3) state database specifications, and 4) fabric specifications. The last two risks are caused by the specifications related to Hyperledger Fabric, whereas the first two risks result from Golang specifications. Thus, developers must be very careful when writing chaincodes in Golang. The proposed system uses a random number generator for issuing certificates in PAR-Tx, CON-Tx, and DEP-Tx. As described in Section V-A, because the processing operations related to certificate generation are executed outside the chaincode, risk (5) does not influence the proposed system. Because of the countermeasures outlined in Table 5, this implementation of the Meta-PKI system is secure from the described threats.

2) COMPARISON WITH OTHER WORK

Table 6 compares the method proposed in the current study with other work from the following perspectives:

TABLE 5. Potential risks associated with using the Hyperledger Fabric chaincode and their effects on the Meta-PKI system.

Category	Risk	Influence
Nondeterminism arising from language instructions	(1) Global Variable	**None**
	(2) KVS Structure Iteration	**None**: Meta-PKI uses key–value structures for managing policies, certificates, and other entities, but range iterations are not required.
	(3) Reified Object Addresses	**None**
	(4) Concurrency of Program	**None**
	(5) Random Number Generation	**None**: Certificates require random numbers, but they are generated by CAs instead of chaincodes.
	(6) System Timestamp	**None**: Certificates include generation times, but they are generated by CAs instead of chaincodes.
Nondeterminism arising from the access outside the blockchain	(7) Web Service	**None**
	(8) System Command Execution	**None**: OpenSSL command is used, but it is called before the transactions.
	(9) External Library Calling	**None**: Some external libraries are used (e.g., encoding/base64 and encoding/json), but all of them are deterministic.
	(10) External File Accessing	**None**
State Database Specification	(11) Range Query Risk	**None**
Fabric Specifications	(12) Field Declarations	**None**
	(13) Cross-Channel Chaincode Invocation	**None**
	(14) Read Your Write	**None**

TABLE 6. Comparison of the main parameters of the models developed in this and other work.

		CA	CT [21]	SCPKI [16]	Cecoin [17]	BlockPGP [18]	Bubbles of Trust [19]	CTB [22]	CertLedger [23]	PBCert [25]	Meta-PKI (this work)
Functions	Registration	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	Revocation	Y	Y	Y	Y	Y	N	Y	Y	Y	Y
	Identification	Y	Y	N	Y	N	N	Y	Y	Y	Y
Properties	Interoperability	N	N	N	N	Y	N	N	N	N	Y
	Transparency	N	Y	Y	Y	Y	Y	Y	Y	Y	PY
	Owner Privacy	N	N	N	N	N	N	N	N	N	Y
Prevention from	False Certificate Requests	Y	Y	N	Y	N	Y	Y	Y	Y	Y
	Misbehaving CAs	N	PY	N/A	Y	N/A	N/A	PY	PY	PY	PY

Abbreviations: Y: Yes, PY: Partially Yes, N: No, N/A: Not Applicable.

- Registration: the certificate owner registers a public key certificate with an identity in order to declare the certificate ownership.
- Revocation: the certificate owner revokes a public key certificate when it is no longer required.
- Identification: the certificate owner is identified and authenticated by a trusted third party, who selects suitable identification methods based on the assurance policy.
- Interoperability: As shown in Fig. 2, a trust relationship is established through one CA corresponding to the certificate issuer. It allows certification paths between different certificate issuers to be validated.
- Transparency: this property describes the transparency of issued certificates.
- Owner Privacy: The issued certificate contains information about a certificate owner (such as his/her e-mail address, name, and country). This information could be leaked if someone obtains the certificate.
- False Certificate Request: a malicious certificate owner requests a certificate registered by another certificate owner.
- Misbehaving CAs: a malicious or compromised CA issues fake certificates.

The proposed system contains basic functions because it simply extends the traditional CA. Compared with related methods, this system has two advantages: *Interoperability* and *Owner Privacy*. The former is ensured by adding the Meta-CA layer. Entities can trust other entities according to the trust relationships established between various Meta-CAs. BlockPGP also exhibits good interoperability properties, but its users must evaluate the trustworthiness of other users by themselves. In addition, the proposed system guarantees owner privacy by not storing the entity certificate in the distributed ledger, as shown in Table 4. As a result, the entity certificate is not disclosed, and its content is concealed from the public. However, fake certificates issued by the misbehaving CAs of the CA layer cannot be detected because of non-transparency. Meanwhile, the Meta-CAs meet the transparency criteria by storing certificates in the distributed ledger. Therefore, it is able to detect the issuance of fake certificates by misbehaving Meta-CAs.

3) PROCESSING TIME

Smart contracts take time to synchronize the ledger across all peers because they must reach a consensus regarding the results of these contracts. Therefore, the processing times of INI-Tx, PAR-Tx, CON-Tx, DEP-Tx, and VAL-Tx

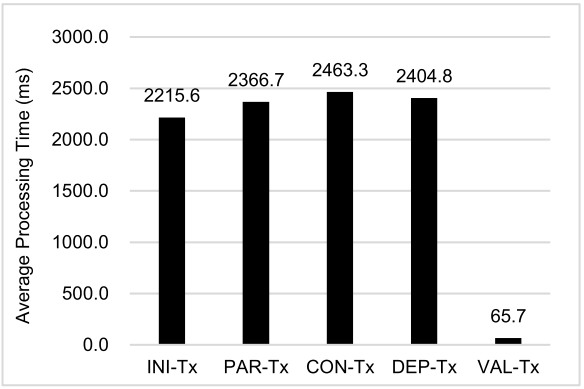

FIGURE 11. Average processing times required to execute various transactions. The average time is calculated from the results of 10 measurements.

estimated for the proposed architecture are as shown in Fig. 11. These transactions are selected to evaluate the establishment of trust relationships. The average times were calculated from the results of 10 measurements. The INI-Tx, PAR-Tx, CON-Tx, and DEP-Tx transactions take approximately 2,400 ms, which is much greater than the time required for executing VAL-Tx (65.7 ms) because the dataset described in Table 4 must be written to the ledger. Hence, when building a system using Meta-PKI, these processing times must be taken into account.

VII. SECURITY

A. ATTACKER

In this study, we consider a situation in which an attacker $CA_{attacker}$ tries to participate in the Meta-PKI system to establish a fake trust relationship with EE. Specifically, the attacker, for the purpose of gaining trust from EE_X, tries to deceive mCA_Y in (8) using $CA_{attacker}$. This would allow an attacker to then launch a man-in-the-middle attack. First, we examine a situation in which the attacker takes over a Meta-CA and makes it issue a public key certificate to $CA_{attacker}$. Subsequently, we study a situation in which the attacker prepares $mCA_{attacker}$ through participation in the Meta-PKI system to spoof a valid mCA. As a prerequisite, none of the entities participating in the Meta-PKI system perform dishonest actions or collude with the attacker.

In these scenarios, the attack is successful if $CA_{attacker}$ is not detected by the verifier via VAL-Tx because the purpose of the attacker is to make EE_X trust $CA_{attacker}$.

B. TAKEOVER OF A META-CA

This section considers an attack in which the attacker takes over a Meta-CA and adds $Cert^{mCA_Y}_{CA_{attacker}}$ to the CA_CERT section of the ledger.

In the first scenario, to keep the attack secret, the attacker makes the valid Meta-CA issue $Cert^{mCA_Y}_{CA_{attacker}}$ to $CA_{attacker}$ without DEP-Tx. Because the Meta-CA deploys a CA by storing $Cert^{mCA}_{CA}$ in the ledger via DEP-Tx, the attacker has to store $Cert^{mCA_Y}_{CA_{attacker}}$ without DEP-Tx. However, this strategy will ultimately fail because of the write restrictions in the distributed ledger. Therefore, VAL-Tx fails to verify whether $Cert^{mCA_Y}_{CA_{attacker}}$ is stored in the ledger using (16).

In another scenario, the attacker, to perform VAL-Tx successfully, gains accesses to a Meta-CA and performs DEP-Tx to store $Cert^{mCA_Y}_{CA_{attacker}}$. In this case, the administrator of mCA_Y can detect $Cert^{mCA_Y}_{CA_{attacker}}$ by monitoring the ledger because the public key certificates stored in the ledger are shared by all peers.

EE_X does not trust $CA_{attacker}$ because mCA_Y does not believe the actions of $CA_{attacker}$ according to

$$\begin{aligned} &distrust\_p(mCA_Y, CA_{attacker}, p, k) \\ &\equiv madeBy\left(CA_{attacker}, k\dot{\supset}p\right) \\ &\supset believe\left(mCA_Y, k\dot{\supset}\dot{\neg}p\right) \end{aligned} \tag{18}$$

and

$$\begin{aligned} &trust\_b(EE_X, mCA_Y, p, k) \\ &\wedge distrust\_p\left(mCA_Y, CA_{attacker}, p, k\right) \\ &\supset distrust\_p(EE_X, CA_{attacker}, p, k). \end{aligned} \tag{19}$$

Hence, the proposed system is secure against an attacker who gains access to Meta-CAs.

C. SPOOFING OF META-CA

This section considers an attack during which $CA_{attacker}$ obtains $Cert^{mCA_Y}_{CA_{attacker}}$ illegally from mCA_Y by falsifying mCA'_Y. The feature of this attack is that the attacker runs mCA'_Y, which has ID_{mCA_Y}. The attacker is able to obtain the ID_{mCA_Y} of the spoofing target because ID_{mCA_Y} can be read publicly from the ledger.

In the first scenario, the attacker prepares mCA'_Y without PAR-Tx to attack secretly. In PAR-Tx, $Cert^{mCA}_{mCA}$ is stored in the ledger and used in (15) to verify the trust relationship between a CA and a Meta-CA. Because the attacker falsifies only ID_{mCA_Y}, mCA'_Y has the same ID as mCA_Y corresponding to ID_{mCA_Y}, but the certificate owned by mCA' ($Cert^{mCA'_Y}_{mCA'_Y}$) is not identical to the certificate owned by mCA ($Cert^{mCA_Y}_{mCA_Y}$) because the private key of mCA'_Y is different from that of mCA_Y. Consequently, because $CA_{attacker}$ possesses not $Cert^{mCA_Y}_{CA_{attacker}}$ but $Cert^{mCA'_Y}_{CA_{attacker}}$, VAL-Tx fails as follows:

$$\text{Verify\_Cert}(Cert^{mCA'_Y}_{CA_{attacker}}, Cert^{mCA_Y}_{mCA_Y}). \tag{20}$$

In another scenario, the attacker makes mCA'_Y participate in the Meta-PKI system by executing PAR-Tx. In this transaction, ID is stored as a key of $Cert^{mCA}_{mCA}$. Because PAR-Tx is implemented to prevent the double registration of ID, the attacker cannot make mCA'_Y participate in the Meta-PKI system with ID_{mCA_Y}.

EE_X does not trust mCA'_Y because mCA_X does not believe mCA'_Y according to

$$\begin{aligned} &distrust\_b(mCA_X, mCA'_Y, p, k) \\ &\equiv believe(mCA'_Y, k\dot{\supset}p) \\ &\supset believe\left(mCA_X, k\dot{\supset}\dot{\neg}p\right), \end{aligned} \tag{21}$$

$$trust\_b(EE_X, mCA_X, p, k) \wedge distrust\_b(mCA_X, mCA'_Y, p, k) \supset distrust\_b(EE_X, mCA'_Y, p, k), \quad (22)$$

and

$$distrust\_b(EE_X, mCA'_Y, p, k) \wedge trust\_p\left(mCA'_Y, CA_{attacker}, p, k\right) \supset distrust\_p(EE_X, CA_{attacker}, p, k). \quad (23)$$

Hence, the proposed system is secure against an attacker who spoofs Meta-CAs.

D. DISCUSSION

As described in Table 6, Meta-PKI prevents false certificate requests and CAs misbehaving against Meta-CAs. From the perspective of securing Meta-CAs against spoofing, because the double registration of *ID* is prevented by the data structure in which the IDs are used as keys, false certificate requests are prevented. To secure Meta-CAs against takeovers, because Meta-CAs can detect the issuing of certificates because of the transparency of the distributed ledger, Meta-CA administrators can detect misbehaving Meta-CAs. The ledger only provides the transparency, so the administrators must still check and revoke invalid certificates. If the ledger is checked more frequently, the administrators can handle misbehaving CAs quickly. Moreover, in Hyperledger Fabric, *Event* [30] can be used to detect ledger updates.

For a man-in-the-middle attack in the proposed system to be successful, the attacker must be trusted by its EEs. To achieve this, the attacker must compromise the Meta-CAs because the EEs only trust the CAs with certificates that were issued by the Meta-CAs and stored in the distributed ledger. However, as shown in Sections VII-B and VII-C, the attacker considered in this work cannot take over or spoof Meta-CAs. As a result, he or she cannot launch a man-in-the-middle attack. Meanwhile, if the CA layer is formed in the hierarchical model, the attacker may compromise one of the CAs in the hierarchy. This represents an advantage for the attacker, but we here focus on a simplified architecture in this study and will consider this aspect in future research work. Spoofing can be addressed, for instance, by changing the data structure in the distributed ledger described in Table 4 to store certificates with ID_{CA}.

The conventional drawbacks of issued certificates (such as the homograph attack discussed earlier) must also be considered. Because these certificates are identified by DNs in the proposed system, the homograph attack confuses the user regarding the honesty of the certificate owner. Similar to the CT, the proposed system records the certificates issued by Meta-CAs publicly in the distributed ledger. Thus, an honest certificate owner can check whether a certificate that has a DN similar to his/her DN is stored in the distributed ledger. Furthermore, because certificates are stored through a chaincode, it is possible to combine the chaincode with various detection mechanisms such as that described in [40].

VIII. CONCLUSION AND FUTURE WORK

To ensure the reliability of trading data, it is important to determine who trades what data while verifying data sources and preventing the denial of trading. Validating the identity of a trading partner can be an important fundamental part of supporting a trustworthy data trading platform. This research focuses on the overconcentration of trust in CAs and aims to decentralize trust points for each service provider by proposing Meta-PKI.

Trust relationships between distributed CAs can be established by performing cross-certification, in which a CA evaluates the reliability of other CAs and issues a public key certificate. However, simple cross-certification of various service providers is insufficient to build proper trust relationships because the cross-certification procedure is not standardized.

Hence, this study proposed a new cross-certification method based on a smart contract that guarantees the fulfillment of contracts without a central administrator. In this method, cross-certification is automated using the smart contract as a framework that computes a score based on the operational policy of a CA. As a proof of concept, it was demonstrated that establishing trust relationships and deploying a CA took approximately 2.4 s, whereas validating the trust relationships took approximately 65.7 ms. Thus, practical applications based on the proposed Meta-PKI system must consider these processing times.

After conducting a security evaluation, it was found that the Meta-PKI system was secure against attackers taking over and spoofing Meta-CAs assuming that the participating entities have not performed dishonest actions nor colluded with the attacker.

The discussion of the results assumes that Meta-CAs are very reliable. Although this assumption is very general considering that many incidents occur because of the incorrect operation of CAs, the Meta-PKI system should incorporate a mechanism that evaluates whether the Meta-CAs are operating according to the required policy. Furthermore, a method of computing the score for small devices that uses metrics other than encryption intensity indexes and the expiration date of a public key certificate should be developed to reduce power consumption.

REFERENCES

[1] C. Perera, "Sensing as a service (S2aaS): Buying and selling IoT data," 2017, *arXiv:1702.02380*. [Online]. Available: http://arxiv.org/abs/1702.02380

[2] S. Kubler, J. Robert, A. Hefnawy, K. Främling, C. Cherifi, and A. Bouras, "Open IoT ecosystem for sporting event management," *IEEE Access*, vol. 5, pp. 7064–7079, 2017, doi: 10.1109/ACCESS.2017.2692247.

[3] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, RFC document 5280, 2008. Accessed: Jan. 31, 2020. [Online]. Available: https://tools.ietf.org/html/rfc5280

[4] *Comodo Report of Incident–Comodo detected and thwarted an intrusion on 26-MAR-2011*. Accessed: Jan. 31, 2020. [Online]. Available: https://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html

[5] J. R. Prints. (2011). *DigiNotar Certificate Authority Breach-Operation Black Tulip*. Accessed: Jan. 31, 2020. [Online]. Available: https://tweakimg.net/files/upload/Operation+Black+Tulip+v1.0.pdf
[6] D. O'Brien, R. Sleevi, and A. Whalley. (2017). *Chrome's Plan to Distrust Symantec Certificates*. Google Security Blog. Jan. 31, 2020. [Online]. Available: https://security.googleblog.com/2017/09/chromes-plan-to-distrust-symantec.html
[7] R. Perlman, "An overview of PKI trust models," *IEEE Netw.*, vol. 13, no. 6, pp. 38–43, 1999, doi: 10.1109/65.806987.
[8] C. Liu, Y. Feng, M. Fan, and G. Wang, "PKI mesh trust model based on trusted computing," in *Proc. 9th Int. Conf. for Young Comput. Scientists*, Nov. 2008, pp. 1401–1405, doi: 10.1109/ICYCS.2008.384.
[9] A. S. Wazan, R. Laborde, F. Barrere, and A. Benzekri, "The X.509 trust model needs a technical and legal expert," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2012, pp. 6895–6900, doi: 10.1109/ICC.2012.6364860.
[10] Hyperledger. *A Blockchain Platform for the Enterprise*. Hyperledger Fabric. Accessed: Jan. 31, 2020. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.4/
[11] S. Kille, *A String Representation of Distinguished Names*, RFC document 1779, 1995. Available:. [Online]. Available: https://tools.ietf.org/html/rfc1779
[12] M. Nystrom and B. Kaliski, *PKCS #10: Certification Request Syntax Specification Version 1.7*, RFC document 2986, 2000. [Online]. Available: https://tools.ietf.org/html/rfc2986
[13] J. Katz, A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, FL, USA: CRC Press, 1996.
[14] J. Huang and D. Nicol, "A calculus of trust and its application to PKI and identity management," in *Proc. 8th Symp. Identity Trust Internet (IDtrust)*, 2009, pp. 23–37, doi: 10.1145/1527017.1527021.
[15] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer, *OpenPGP Message Format*, RFC document 4880, 2007. [Online]. Available: https://tools.ietf.org/html/rfc4880
[16] M. Al-Bassam, "SCPKI: A smart contract-based PKI and identity system," in *Proc. ACM Workshop Blockchain, Cryptocurrencies Contracts (BCC)*, 2017, pp. 35–40, doi: 10.1145/3055518.3055530.
[17] B. Qin, J. Huang, Q. Wang, X. Luo, B. Liang, and W. Shi, "Cecoin: A decentralized PKI mitigating MitM attacks," *Future Gener. Comput. Syst.*, vol. 107, pp. 805–815, Jun. 2020, doi: 10.1016/j.future.2017.08.025.
[18] A. Yakubov, W. Shbair, and R. State, "BlockPGP: A blockchain-based framework for PGP key servers," in *Proc. 6th Int. Symp. Comput. Netw. Workshops (CANDARW)*, Nov. 2018, doi: 10.1109/CANDARW.2018.00065.
[19] M. T. Hammi, B. Hammi, P. Bellot, and A. Serhrouchni, "Bubbles of trust: A decentralized blockchain-based authentication system for IoT," *Comput. Secur.*, vol. 78, pp. 126–142, Sep. 2018, doi: 10.1016/j.cose.2018.06.004.
[20] E. Gabrilovich and A. Gontmakher, "The homograph attack," *Commun. ACM*, vol. 45, no. 2, p. 128, Feb. 2002, doi: 10.1145/503124.503156.
[21] B. Laurie, A. Langley, and E. Kasper, *Certificate Transparency*, RFC document 6962, 2013. Available:. [Online]. Available: https://tools.ietf.org/html/rfc6962
[22] D. S. V. Madala, M. P. Jhanwar, and A. Chattopadhyay, "Certificate transparency using blockchain," in *Proc. IEEE Int. Conf. Data Mining Workshops (ICDMW)*, Nov. 2018, pp. 71–80, doi: 10.1109/ICDMW.2018.00018.
[23] M. Y. Kubilay, M. S. Kiraz, and H. A. Mantar, "CertLedger: A new PKI model with certificate transparency based on blockchain," *Comput. Secur.*, vol. 85, pp. 333–352, Aug. 2019, doi: 10.1016/j.cose.2019.05.013.
[24] L. Chuat, P. Szalachowski, A. Perrig, B. Laurie, and E. Messeri, "Efficient gossip protocols for verifying the consistency of certificate logs," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Sep. 2015, pp. 415–423, doi: 10.1109/CNS.2015.7346853.
[25] S. Yao, J. Chen, K. He, R. Du, T. Zhu, and X. Chen, "PBCert: privacy-preserving blockchain-based certificate status validation toward mass storage management," *IEEE Access*, vol. 7, pp. 6117–6128, 2019, doi: 10.1109/ACCESS.2018.2889898.
[26] S. Matsumoto, R. M. Reischuk, P. Szalachowski, T. H.-J. Kim, and A. Perrig, "Authentication challenges in a global environment," *ACM Trans. Privacy Secur.*, vol. 20, no. 1, pp. 1–34, Feb. 2017, doi: 10.1145/3007208.
[27] M. Cooper, Y. Dzambasow, P. Hesse, S. Joseph, and R. Nicholas, *Internet X.509 Public Key Infrastructure: Certification Path Building*, RFC doucumrent 4158, 2005. Accessed: Jan. 31, 2020. [Online]. Available: https://tools.ietf.org/html/rfc4158
[28] W. Dai, C. Dai, K.-K.-R. Choo, C. Cui, D. Zou, and H. Jin, "SDTE: A secure blockchain-based data trading ecosystem," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 725–737, 2020, doi: 10.1109/TIFS.2019.2928256.
[29] Z. Guan, X. Lu, N. Wang, J. Wu, X. Du, and M. Guizani, "Towards secure and efficient energy trading in IIoT-enabled energy Internet: A blockchain approach," *Future Gener. Comput. Syst.*, vol. 110, pp. 686–695, Sep. 2020, doi: 10.1016/j.future.2019.09.027.
[30] *Hyperledger Fabric SDK for Node.js Tutorial: Fabric-Client: How to use the Channel-Based Event Service*. Hyperledger. Accessed: Jun. 5, 2020. [Online]. Available: https://hyperledger.github.io/fabric-sdk-node/release-1.4/tutorial-channel-events.html
[31] *Small. Simple. Secure*. Alpine Linux. Accessed: Jan. 30, 2020. [Online]. Available: https://alpinelinux.org/
[32] *Debug Your App, Not Your Environment*. Docker. Accessed: Jan. 30, 2020. [Online]. Available: https://www.docker.com/
[33] *Welcome to VirtualBox.org*. VirtualBox. Accessed: Jan. 30, 2020. [Online]. Available: https://www.virtualbox.org/
[34] *Hyperledger Fabric Samples*. GitHub. Accessed: Jan. 30, 2020. [Online]. Available: https://github.com/hyperledger/fabric-samples
[35] *Welcome to OpenSSL*. OpenSSL. Accessed: Jan. 30, 2020. [Online]. Available: https://www.openssl.org/
[36] (2014). *Almquist Shell*. Rosetta Code. Accessed: Jan. 30, 2020. [Online]. Available: https://rosettacode.org/wiki/Almquist_Shell
[37] *The Go Programming Language*. Accessed: Jan. 30, 2020. [Online]. Available: https://golang.org/
[38] *Introducing JSON*. JSON. Accessed: Jan. 30, 2020. [Online]. Available: https://www.json.org/json-en.html
[39] K. Yamashita, Y. Nomura, E. Zhou, B. Pi, and S. Jun, "Potential risks of hyperledger fabric smart contracts," in *Proc. IEEE Int. Workshop Blockchain Oriented Softw. Eng. (IWBOSE)*, Feb. 2019, pp. 1–10, doi: 10.1109/IWBOSE.2019.8666486.
[40] H. Suzuki, D. Chiba, Y. Yoneya, T. Mori, and S. Goto, "ShamFinder: An automated framework for detecting IDN homographs," in *Proc. Internet Meas. Conf.*, Oct. 2019, pp. 449–462, doi: 10.1145/3355369.3355587.

SHOHEI KAKEI received the B.E. and M.E. degrees from Gifu University, Japan, in 2011 and 2013, respectively, and the Ph.D. degree from Kobe University, Japan, in 2019. He has been an Assistant Professor with the Department of Computer Science and Engineering, Nagoya Institute of Technology, Japan, since 2019. His current research interests include digital forensics, blockchain technology, and information security. He is a member of IEICE and IPSJ.

YOSHIAKI SHIRAISHI (Member, IEEE) received the B.E. and M.E. degrees from Ehime University, Japan, in 1995 and 1997, respectively, and the Ph.D. degree from the University of Tokushima, Japan, in 2000. From 2002 to 2006, he was a Lecturer with the Department of Informatics, Kindai University, Japan. From 2006 to 2013, he was an Associate Professor with the Department of Computer Science and Engineering, Nagoya Institute of Technology, Japan. Since 2013, he has been an Associate Professor with the Department of Electrical and Electronic Engineering, Kobe University, Japan. His current research interests include information security, cryptography, computer networks, and knowledge sharing and creation support. He is a member of ACM, and a Senior Member of IEICE and IPSJ. He received the SCIS 20th Anniversary Award and the SCIS Paper Award from ISEC Group of IEICE, in 2003 and 2006, respectively. He received the SIG-ITS Excellent Paper Award from SIG-ITS of IPSJ, in 2015.

MASAMI MOHRI (Member, IEEE) received the B.E. and M.E. degrees from Ehime University, Japan, in 1993 and 1995, respectively, and the Ph.D. degree in engineering from the University of Tokushima, Japan, in 2002. From 1995 to 1998, she was an Assistant Professor with the Department of Management and Information Science, Kagawa Junior College, Japan. From 1998 to 2002, she was a Research Associate with the Department of Information Science and Intelligent Systems, University of Tokushima. From 2003 to 2007, she was a Lecturer with the Department of Information Science and Intelligent Systems. From 2007 to 2017, she was an Associate Professor with the Information and Multimedia Center, Gifu University, Japan. Since 2017, she has been an Associate Professor with the Department of Electrical, Electronic and Computer Engineering, Gifu University. Her research interests are in coding theory, information security, and cryptography. She is a Senior Member of IEICE.

TORU NAKAMURA received the B.E., M.E., and Ph.D. degrees from Kyushu University, in 2006, 2008, and 2011, respectively. In 2011, he joined KDDI and in the same year, he moved KDDI Research and Development Laboratories, Inc. (currently renamed KDDI Research, Inc.). Since 2018, he has been a Researcher with the Advanced Telecommunications Research Institute International (ATR). His current research interests include security and privacy, especially privacy enhanced technology and analysis of privacy attitudes. He is a member of IEICE and IPSJ. He received CSS2016SPT Best Paper Award.

MASAYUKI HASHIMOTO received the B.E., M.E., and Ph.D. degrees in communication engineering from Osaka University, Osaka, Japan, in 1995, 1997, and 2007, respectively, and the M.B.A. degree from the Graduate School of Management, GLOBIS University, Tokyo, Japan, in 2017.

From 1997 to 2007, he was a Research Engineer with KDDI Research Inc. From 2007 to 2017, he was a Manager with KDDI Research Inc. From 2017 to 2019, he was a Manager with KDDI Corporation. Since 2019, he has been a Head of the Department of Advanced Security with the Adaptive Communication Research Laboratories, Advanced Telecommunications Research Institute International. His research interests include supply chain security, vulnerability-information extraction, privacy-policy description, group signature, and anonymization for privacy. He was a recipient of ITE Suzuki Encouragement.

SHOICHI SAITO received the B.S. and M.E. degrees in engineering from Ritsumeikan University, in 1993 and 1995, respectively, and the Dr.Eng. degree in 2000. He became a Research Associate at the Department of Computer and Communication Sciences, Wakayama University, in 1998. He was an Assistant Professor in 2003, and an Associate Professor in 2005. He was an Associate Professor with the Nagoya Institute of Technology, in 2006, where he has been a Professor since 2016. His research interests are operating systems, security, and the Internet. He is a member of ACM, IEEE CS, and IPSJ.