Electronic Preprint for Journal of Information Processing Vol.29

Regular Paper

IoT-oriented Secure Data Sharing Using Public Cloud

Kenta Yokogi[1,†1,a)] Naoya Kitagawa[1,†2,b)] Nariyoshi Yamai[1,c)]

Received: March 8, 2021, Accepted: September 9, 2021

Abstract: The number of IoT devices is continuously increasing. Secure data sharing governed by appropriate access control is required to safely utilize data generated by IoT devices. Storing data in a public cloud is suitable for deploying services with distributed data sharing on a large scale. However, this raises security concerns since even when the data are encrypted, an adverse third party may access them if a decryption key is stored within the same environment (key escrow problem). Conventional methods are not supposed to be used in the IoT environment or have issues with security, key distribution, and changing access authority. We propose a novel approach to securely share the data generated by IoT devices within a public cloud. Our method enables 1) addressing the key escrow problem; 2) providing forward secrecy; 3) ensuring indistinguishability under Adaptive Chosen Ciphertext Attack (safety equivalent to IND-CCA2); 4) changing access authority easily; and 5) saving computational resources of IoT devices. We implemented this method and evaluated its performance. The experimental results show that it has comparable or better performance compared with conventional methods. Furthermore, we confirm that resource consumption in our method is more practical even in the large-scale IoT environment.

Keywords: access control, cloud computing, internet of things, secure data sharing

1. Introduction

In recent years, the number of IoT devices has been growing and is currently expected to reach 40 billion worldwide by 2020 [1]. The areas of highest IoT growth are smart factories and smart cities. Since IoT devices continuously generate large volumes of data, managing the authority for accessing data generated by IoT is becoming ever more important. Sharing all of this data across multiple organizations, distributing it graphically and achieving the required access scale flexibility make utilizing public clouds provided by third parties unavoidable for a host of different reasons including convenience and cost [2], [3].

However, storing the data in a public cloud gives rise to security concerns. Even when the data are encrypted to restrict access targets, if the key required for decrypting the data is stored in the same public cloud, there is a concern that an unexpected third party may get access to the data due to a cloud system vulnerability, misconfiguration, or malicious employee. This problem is referred to as the key escrow problem.

Many secure data sharing methods using a public cloud have been proposed to mitigate the key escrow problem. However, these methods are not intended to be applied to the resource limited IoT environments and are associated with a number of problems such as key distribution, recipient addition, and limited computational resources for cryptographic operations.

In the present paper, while considering this background including the key escrow problem and limitations on existing secure data sharing solutions in the IoT environment, we propose a novel IoT-oriented secure data sharing scheme.

The main contributions of the present study can be summarized as follows:

(1) We propose a novel approach to securely share data generated by IoT devices with a public cloud. We discuss the issues of using the existing access control and data sharing schemes in the IoT environment, such as key management, distribution, and recipient addition methods and consider the key escrow problem. In the proposed approach, full trust in the public cloud is not needed and only a single semi-trusted entity is requested. We design this scheme to address the key escrow problem, to enable forward secrecy, and to facilitate easily changing the access authority. Moreover, the proposed scheme is assumed to be applied to the IoT environment and therefore, the required computational resources for users need to be minimized. In other words, a data owner does not perform any data encryption operations, and a data recipient performs the decryption operation only for the data encryption key (DEK).

(2) We confirm that the proposed scheme provides IND-CCA2 equivalent safety in the standard model by an attack-based formulation.

(3) We consider the public clouds such as Amazon S3 compatible, as a commonly used object storage, and therefore, the proposed method can be easily applied to existing cloud providers.

(4) We implement the proposed scheme to evaluate its perfor-

[1] Tokyo University of Agriculture and Technology, Koganei, Tokyo 184–8588, Japan
[†1] Presently with CRI Middleware Co., Ltd., Shibuya, Tokyo 150–0002, Japan
[†2] Presently with National Institute of Informatics, Chiyoda, Tokyo 101–8430, Japan
[a)] yokogi.kenta@cri-mw.co.jp
[b)] kitagawa@nii.ac.jp
[c)] nyamai@cc.tuat.ac.jp

mance. The experimental results show that it has equal or better performance compared with the existing data sharing schemes. In addition, as a result of estimations made under realistic conditions, we confirm that the resource consumption of the proposed method is sufficiently more practical even in the large-scale IoT environment.

The rest of the paper is organized as follows. In Section 2, we provide an overview on the related work dedicated to access control, data encryption and secure data sharing. The details about the proposed IoT-oriented secure data sharing scheme are provided in Section 3. We then analyze the security and evaluate the performance in Sections 4 and 5. Finally, we conclude this paper in Section 6.

2. Related Work

2.1 Access Control

In this section we describe two types of popular and widely used access control methods: role-based access control (RBAC) [4] and attribute-based access control (ABAC) [5].

In the RBAC model, an authority is assigned to a role, and the role is granted to an access control entity as one attribute information. The authority and access control entity are connected with the many-to-many relationship, and the role can be regarded as the naming of that relationship. It is possible to group multiple authorities or reuse the pre-defined ones by using roles, rather than granting authorities directly to the access control entities.

In its turn, in the ABAC model, attribute information of the access control entity, such as age or job post, is evaluated and it is then determined whether to grant authority to that entity. A corresponding policy is defined as a set of conditions and results. Dynamic parameters such as date or time, are used to define policy conditions.

If the attribute information is correctly granted, the ABAC model can handle the increasing number of access control entities and changes in the attribute information. By performing the automated configuration of the IoT devices based on the ABAC model, it is possible to efficiently manage access control in large-scale IoT environments [6].

2.2 Data Encryption

There are two types of data encryption methods: symmetric key encryption and public (asymmetric) key encryption. Symmetric key encryption implies using the same common key to encrypt and decrypt operations. Public key encryption on the other hand implies defining different keys to perform encryption and decryption. Generally, symmetric key encryption requires less computational resources than the public key encryption.

As of 2019, the advanced encryption standard (AES) is one of the most commonly used symmetric key encryption methods. AES was specified by the National Institute of Standards and Technology of the United States as a part of Federal Information Processing Standards 197 [7] in 2001.

In its turn, the RSA method proposed by Rivest, Shamir, and Adleman (RSA) in 1977 [8] is one of the first practical public key ciphers and is still widely used at present. RSA is a cipher based on the difficulty corresponding to factorization of the product of two large prime numbers.

In addition, a cryptography scheme based on the difficulty corresponding to the discrete logarithm problem on elliptic curves (elliptic-curve cryptography (ECC)) has been drawing attention as a method to replace RSA. ECC is considered as capable of achieving equivalent security with a shorter key length and less processing time compared with RSA. One example of ECC is an elliptic-curve system called the ElGamal encryption system proposed by ElGamal in 1984 [9]. In public key encryption, the public key of the recipient is used. If the recipient's public key is replaced, decryption by an unintended third party becomes possible.

The public key infrastructure (PKI) is used to prevent these adverse decryptions. Moreover, there is a kind of public key encryption called ID-based encryption [10] in which encryption is performed using public information such as the recipient's e-mail address or username as a part of the public key. Since the public information is directly linked to the recipient, replacing these keys is difficult. Therefore, by using ID-based encryption, it is possible to construct a public key cryptosystem that prevents public key replacement at a lower computational and financial cost compared with PKI.

However, even if ID-based encryption is applied, it will not prove successful unless a private key can be generated freely based on the public information, and therefore, it is necessary to deploy a reliable key center to manage the public key and private key generation.

2.3 Secure Data Sharing

Xu et al. proposed a certificateless proxy re-encryption scheme for secure data sharing using a public cloud (CL-PRE) [11]. The proxy re-encryption is a method of data re-encryption that enables decrypting the data with another key and does not need to perform decryption during the re-encryption process. To re-encrypt the data encrypted by the data owner, the data recipient uses a re-encryption key generated from the data owner's private key and the data recipient's public key. The authors outlined that a reason for using re-encryption is that if the number of users becomes large in the data sharing scheme, the PKI-based method may have a problem with public key management, and ID-based encryption is associated with the key escrow problem. In CL-PRE, when the data owner uploads the data, it is encrypted by using the common key encryption with the generated DEK. Then, the data owner encrypts DEK with its own public key and generates a re-encryption key, which is able to re-encrypt the encrypted DEK for the data recipient and upload these data to the cloud. When the data recipient requests the data, the proxy re-encrypts the encrypted DEK using the re-encryption key and provides it to the data recipient along with the encrypted data. The data recipient decrypts the encrypted DEK with the private key and decrypts the encrypted data using the decrypted DEK. In the cloud with a proxy, only the encrypted data, encrypted DEK, and re-encryption key are stored, and the data remain encrypted (not decrypted) during the re-encryption process. Therefore, it is impossible to decrypt the data with only the information available in the cloud. Consequently, there is no key escrow problem in the CL-PRE scheme.

However, the CL-PRE scheme requires data owners and recipients to encrypt and decrypt the data, which is an unsuitable requirement concerning the IoT environments. In addition, when adding new data recipients, the data owner needs to generate an additional re-encryption key based on DEK. In the IoT environment, it is difficult for IoT devices to continuously manage and maintain DEK of the generated data.

Khan et al. proposed an incremental version of aproxy re-encryption scheme [26] called I-PReS. I-PReS allows data owners to perform incremental proxy re-encryption when editing an encrypted file, instead of re-encrypting the entire file. While I-PReS allows for efficient partial editing of encrypted data when the data owner is a mobile device, it suffers from the same issue as CL-PRE when an IoT environment is assumed. Specifically, this method is not suitable for use in an IoT environment because the data sender and receiver must encrypt and decrypt the data, and additional data recipients cannot be added without the cooperation of the data owner.

Seo et al. proposed efficient certificateless encryption for secure data sharing in public clouds [12], which allowed improving CL-PRE. They noted that a method called certificateless public key cryptography and used by CL-PRE was based on pairing operation and is computationally expensive. They also stated that CL-PRE only achieved the chosen-plaintext attack (CPA) safety, whereas having stronger chosen-ciphertext attack (CCA) safety was required in a practical system. Their method was rather similar to CL-PRE, except that DEK was encrypted directly using the recipient's public key. As a result, no re-encryption operation was performed so that the pairing operation was not required. They also stated that their method achieved IND-CCA safety that was better than the CPA safety of CL-PRE. However, considering that CL-PRE, encryption, and decryption operations have to be performed by the data owner and recipient, it is evident that when a new recipient is added, the data owner needs to additionally encrypt DEK with the public key of the added recipient. These requirements are not suitable due to limitations on the IoT environment as described above in CL-PRE.

There is a fast and secure data sharing method proposed by Ali et al. [13], which uses a public cloud and called SeDaSC. In SeDaSC, a random key Ki, which has the same length as DEK, is generated for all recipients, and $Ki' = DEK \oplus Ki$ is calculated. Here, $\oplus$ represents the XOR operation for each bit. Then, it is necessary to store Ki in the proxy, send Ki' to each recipient, and discard DEK. When the recipient requests the data, Ki' is sent to the proxy, and the proxy calculates $DEK = Ki' \oplus Ki$, restores the DEK, and decrypts the data. As random bit generation and XOR operation can be processed efficiently, SeDaSC can generate keys for recipients much faster than other methods using public key encryption. In addition, there is an advantage that data can be encrypted and decrypted in a proxy. However, since the key for the recipient is generated for each DEK, there arises the issue of defining a way to distribute the key to the recipient. Normally, the key can be distributed to recipients via e-mail, instant messenger, and SNS, etc. However, in the IoT environment, continuously maintaining a real-time communication channel with recipients is difficult.

There is a secure data sharing method for the IoT environment using a public cloud proposed by Mollah et al. [14]. In their method, the proxy encrypts DEK with the public key received from the key generation server and stores it in the cloud along with the recipient information. When the recipient requests the data, the proxy decrypts the DEK with the private key received from the key generation server, decrypts the data, and provides it to the recipient. This method has important advantages such as being able to search data without decryption and applicability to the IoT environment. However, only a small part of a storage, which contains the encrypted data and recipient information is addressed to the key escrow problem. Therefore, if the proxy is compromised for some reason, decrypting the information already stored with the private key that can be received from the key generation server becomes possible, and consequently, there is no forward secrecy. Furthermore, the information corresponding to all recipients is stored in the relational database, and it may be a bottleneck with regard to scaling the number of the IoT devices.

Our method solves the issues of the previous approaches discussed in this section as follows:

- The proxy mediates data encryption and decryption. This keeps the data secure even if the data sender, e.g., IoT devices, and receiver do not support data encryption and/or decryption.
- Access privileges are managed by the proxy on a policy basis. This eliminates the need for key delivery between the data sender and receiver. In addition, our method enables the addition of data receivers without any action by the data sender.
- The information necessary for decryption is not left on the storage managed by the proxy. Even if the control of the proxy is taken over by an attacker, the data processed before that happened is kept secure. Therefore, our method provides forward secrecy.

3. IoT-oriented Secure Data Sharing Using Public Cloud

In this section, we provide a detailed description about the proposed IoT oriented secure data sharing method based on the public cloud scheme.

3.1 Components

As shown in **Fig. 1**, the proposed scheme consists of the four entities: client, service, cloud storage, and proxy. A detailed description of each entity is provided as follows.

(1) Client

The client is an entity that generates the data to be shared.

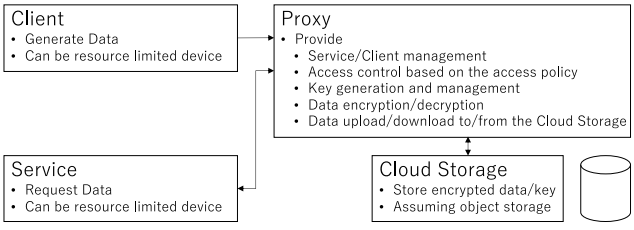


Fig. 1 Components of our IoT-oriented secure data sharing scheme.

We assume that the main type of client is an IoT sensor device. Therefore, the proxy manages the data encryption processing, and the client can be considered as a resource limited device.

(2) Service
The service is an entity that consumes the data produced by the client. The service can be either an IoT device or a hosted service. Similarly as in the case of the client, the proxy manages the processing tasks such as data decryption, and the service can be considered as a resource limited device. The service however must be a device that holds a small size of private key and has minimum resources to be able to decrypt a small size of DEK with its private key.

(3) Cloud Storage
The cloud storage is an entity that stores the encrypted data generated by the client and keys for the service. We can use any object storage service as the cloud storage. Moreover, the stored data are encrypted, and therefore, these object storage services do not require being fully trusted by the user. In the proposed scheme, when the client or service needs to access the cloud storage, the proxy always mediates these accesses. Therefore, the cloud storage only needs to be accessible by the proxy, and there is no need to implement functions such as issuing a signed URL with an embedded access authority.

(4) Proxy
The proxy is a semi-trusted entity that mediates access between the client, the service, and the cloud storage. The proxy manages the following processes: access control based on the access policy, data encryption and decryption, and data download and upload. When sharing the data between different organizations, connecting multiple proxies to a single public cloud is possible.

3.2 Basic Scheme

3.2.1 Client/Service Registration and Key Distribution

First, the client and service must be registered in the proxy and then the attribute information and keys exchanged. **Figure 2** illustrates the client and service registration, and key distribution process. The client registration and key distribution process is defined, as follows:

(1) The client sends the attribute information to the proxy.
(2) When the proxy receives the attribute information from the client, it generates ID and the access token as authentication information.
(3) The proxy stores the generated authentication information in association with the received attribute information.
(4) The proxy sends the generated authentication information to the client in response to the client's attribute information.

ID and the access token are used to uniquely identify and authenticate the clients and services. These should be sufficiently resistant to brute-force attacks. For example, a randomly generated 256 bits value can be used as an ID and access token.

Unlike the client, in the registration and key distribution process corresponding to the service, the authentication information generated by the proxy contains a public key and private key pair.

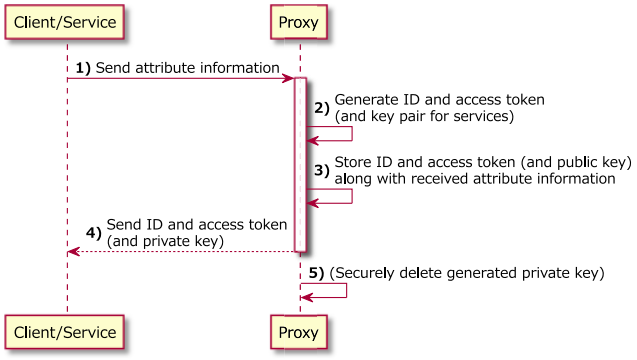


Fig. 2 Client/Service registration and key distribution.

Then, the proxy sends the private key to the service and stores the public key and thereafter securely deletes the private key. This key pair can be used to encrypt and decrypt the DEK which corresponds to the service. It is possible to use any public key cryptographic algorithm such as RSA or ECC. The key should have a sufficiently secure length as recommended by the public key cryptographic scheme used. A detailed description about data encryption is provided in Section 3.2.4.

Under a premise that the cloud storage is not fully trusted, the proxy must store the access token and public key locally since there is a possibility that information stored in the cloud storage can be modified by a malicious third party.

3.2.2 Authentication

The authentication between the client or service and the proxy is based on ID and the access token received as a result of the key distribution process as described in Section 3.2.1.

Exposing the authentication information in a form of plaintext during the communication between the client or service and the proxy may lead to leakage of the authentication information and spoofing of the client or service by a malicious third party. Therefore, the communication between the client or service and the proxy should be performed through a secure channel. To establish a secure channel to the proxy deployed on the local network, the administrator can register the proxy's certificate signature to the clients and services. To establish a secure channel to the proxy deployed on the cloud or external network, the client and service can check the certificate trust chain in the same way as with standard TLS scheme. If the client or service cannot establish a secure channel for communication with the proxy due to limitations of IoT devices, it is recommended to use appropriate technology to prevent exposing the plaintext authentication information. For example, using hash-based message authentication code (HMAC) [15] to protect data integrity is also recommended if the communication is performed in the plaintext format. Furthermore, using the time-based one-time password [16] or HMAC-based one-time password [17] or challenge response authentication such as challenge handshake authentication protocol [18], can be used to prevent exposing the plaintext authentication information during communication.

3.2.3 Access Policy Configuration

To establish the policy-based access control in the secure data sharing, the access policy configuration must be created in advance. **Figure 3** illustrates the access policy employed in the

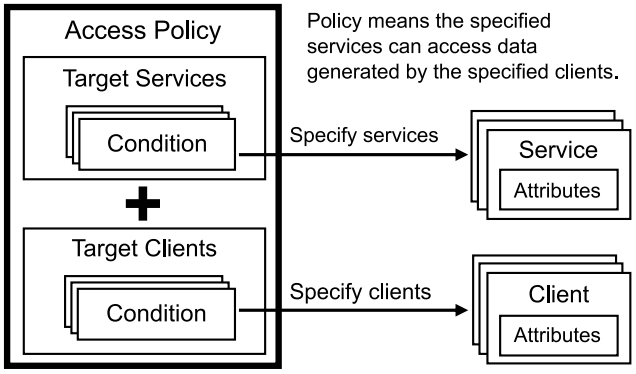


Fig. 3 Access policy.

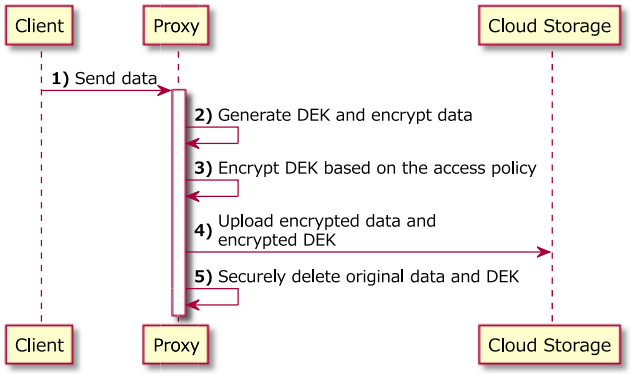


Fig. 4 Data encryption and upload.

scheme. The access policy represents which services can access the data produced by which client. To describe the access policy, we can use the attribute information of the client and service exchanged during the registration process as discussed in Section 3.2.1. A query language or policy description language such as extensible access control markup language [19], can be used to describe the access policy. The detailed approach of automated registration of clients and services and the access control based on the attribute information in the IoT environment have been studied in our previous work [6].

3.2.4 Data Encryption and Upload

Figure 4 illustrates the data encryption and upload process. The data encryption and upload process has the following steps:

(1) The client uploads the data to the proxy. The client can specify the name of the data.
(2) The proxy generates DEK and encrypts the received data.
(3) The proxy generates a list of services with access authority to the data based on the access policy. Then, it encrypts DEK for each service in the generated list using each service's public key.
(4) The proxy uploads the encrypted data and encrypted DEK for each service to the cloud storage.
(5) The proxy securely deletes the original data and generated DEK after uploading the data into the cloud storage.

The proxy can use a fast and secure symmetric key cryptosystem such as AES to encrypt the received data. DEK should be sufficiently resistant to brute-force attacks. For example, a randomly generated 256 bits value can be used as DEK.

3.2.5 Data Download and Decryption

Figure 5 illustrates the data download and decryption process,

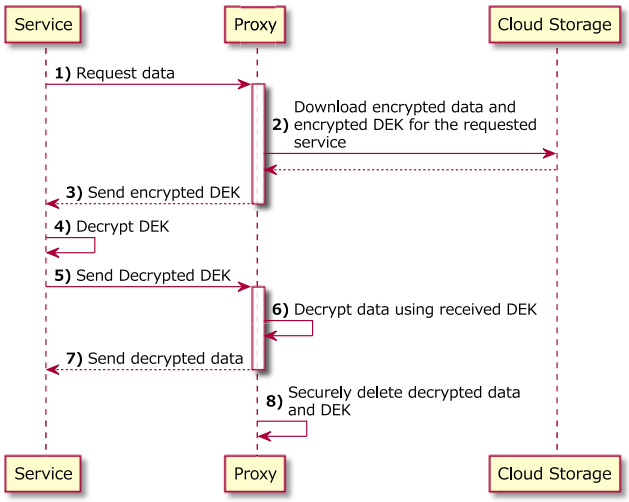


Fig. 5 Data download and decryption.

which includes the following steps:

(1) The service sends a request for the required data to the proxy.
(2) When the proxy receives the request from the service, it downloads the corresponding encrypted data, and encrypted DEK for the service which sent the request from the cloud storage.
(3) The proxy sends the encrypted DEK to the service in response to step 1.
(4) The service decrypts the received DEK using their own private key.
(5) The service sends the decrypted DEK to the proxy.
(6) The proxy decrypts the encrypted data using the received DEK.
(7) The proxy sends the decrypted data to the service as the DEK response in step 5.
(8) The proxy securely deletes the original data and DEK after sending the data to the service.

In step 2, if there is no DEK encrypted for the corresponding service in the cloud storage, this means that the service that requested the data does not have appropriate access authority to the requested data. Therefore, the proxy needs to interrupt the process.

If the service is a hosted service with sufficient computing resources to decrypt the data, it is possible to send the encrypted data at the same time as the encrypted DEK in step 3, and the service can be responsible for the entire decryption process.

3.2.6 Search and Obtain The Accessible Data List for The Service

In the data download and decryption phase described in Section 3.2.5, the service needs to provide the client's ID and the name of the required data to the proxy. Except special cases, such as when data name rules (for example, the name based on the date) are defined in advance, providing the required data with a name having no information is difficult for the services. Therefore, the proxy may need to provide an accessible data list to the service. This requirement can be easily achieved in a way such as preparing a relational database and the proxy stores and managing the uploaded data information when the client uploads the data. Furthermore, in addition to the proxy managing the uploaded data

information directly, the same requirement can be achieved by managing the indexes of objects that exist in the cloud storage. As one example, there is a way to search in the cloud object storage based on using a metadata model [20] proposed by Imran et al. In this method, a server is installed between the object storage and the system that accesses it, and the metadata related to the stored object is simultaneously received and managed while mediating the access.

3.3 Changing Access Authority

When the access policy is changed or a new client or service is added, it is rnecessary to change the access authority of the existing data. To revoke services' access to the data that have already been granted, the required action is to simply delete the encrypted DEK in the cloud storage. However, when granting access privileges for the existing data to new services, it is necessary to generate a new encrypted DEK for these services and to store it in the cloud storage. In methods without the key escrow problem mentioned in Section 2, the operation by the data owner is indispensable for granting new access authority to the existing data. However, when assuming it in the IoT environment, it is difficult to encourage the data owner to keep the original data or its DEK. In the proposed scheme, the authority can be changed by using the administrator who manages the client and service registration and access policy. Specifically, the administrator registers its public key in the proxy, and when the data are uploaded, its DEK is also encrypted for the administrator in the same way as for other services. Service registration and access policy changes are performed by the administrator's operation. Currently, the proxy decrypts DEK of the related data using the administrator's private key and encrypting again to satisfy the newly granted access authority.

3.4 Extension for Support Data Sharing Between Multiple Organizations

The proxy can cooperate with other proxies to share the data between different organizations by connecting multiple proxies to a single public storage and sharing the stored data. When the data are uploaded by the client, the proxy requests the public key of the service (the final step of sharing) with the other cooperating proxy and encrypts DEK using the obtained public key. To prevent an MITM attack, communication between proxies should always be performed over an end-to-end encrypted secure channel. By caching the obtained public key for a reasonable period, it is possible to reduce the amount of communication between proxies and the corresponding overhead. However, introducing a public key cache may cause a delay in revocation of access authority for services. To eliminate this delay, the proxy that receives the data request from the service must check not only whether the encrypted DEK for the service is stored in the cloud storage but also whether the access policy allows the requested service accessing the requested data. If there is no access authority anymore, performing additional processing such as deleting the encrypted DEK, is necessary for the service stored in the cloud storage.

4. Security Analysis

4.1 Threats to the Cloud

The cloud is subject to various threats including insiders, vulnerabilities and miconfigurations. Due to these factors, we do not consider the cloud to be a secure and trustful entity in the context of our proposed method. Specifically, the proposed method assumes that the cloud lacks confidentiality. In the following sections, we show that third parties cannot access the original data even when both the proxy and storage are on the cloud which lacks confidentiality.

However, we do not cover integrity and availability since these depend on the specific system implementation. Specifically, users' requirements for integrity and availability can be satisfied by making backups and making storage redundant.

4.2 Threats to the Proxy

The first threat to the proxy is the risk from the intercepting or tampering with communications between the client, the service, and the proxy. However, this threat can be addressed simply by encrypting each communication.

The second threat is the risk of an attacker taking over control of the proxy. As a countermeasure against this risk, our method provides forward secrecy as described in Section 4.3.

However, when uploading and downloading data after an attacker takes control of the proxy, the content of the data cannot be protected from the attacker. The attacker can obtain the plaintext of the data during uploads made by the client and downloads made by the service after taking control of the proxy. On the other hand, our method provides the proxy with the minimum information needed to decrypt the data only when the service downloads the data. As a result, since the data on the cloud storage is encrypted, our method does not allow access to data stored in cloud storage, even if an attacker takes control of the proxy. In addition, in order to mitigate damage when the control of the proxy is taken away by an attacker, it is important to detect it as soon as possible through monitoring.

4.3 Key Escrow and Forward Secrecy

The method proposed in the present paper is not associated with the key escrow problem because the original data cannot be decrypted by using just the information on the proxy or cloud storage. The keys required to decrypt the data in the cloud storage are encrypted, and decryption is possible only by using the private key of the authoritative services or administrator.

In addition, since our scheme does not leave any information necessary for decryption on the storage managed by the proxy, it is impossible to use information about the proxy to decrypt previously encrypted data even if the proxy is compromised by an attacker for some reason. Therefore, our proposed scheme provides forward secrecy.

4.4 Security Model

In this section, we evaluate the security of the proposed method as the public key cryptosystem by considering an attack-based formulation.

Among two similar length plaintexts m_0 and m_1, if the adversary cannot distinguish between the ciphertext of m_0 and m_1 these plaintext are said to be indistinguishably (IND) secure. In addition, when being IND secure for an adversary who can perform a CPA, it becomes IND-CPA secure. Furthermore, even in the cases when the adversary manages to access a decryption oracle, if the adversary's advantage (probability of distinguishing the chosen ciphertext being greater than $1/2$) is sufficiently small to be ignored, it becomes IND-CCA secure. The decryption oracle is an oracle that inputs the sent value to a decryption algorithm and returns the result. If the adversary can access the decryption oracle adaptively, it is considered to be IND-CCA2 secure. The formulation of the IND-CCA game for the proposed method is the following:

(1) An adversary is registered in the proxy as a client and a service. In addition, the adversary has unlimited access to the cloud storage:
- Being registered as a client implies obtaining the public key in a public key cryptosystem.
- Accessing to the cloud storage is equivalent to obtaining a challenge ciphertext in a public key cryptosystem.
- Being registered as a service and accessing the cloud storage is equivalent to accessing the decryption oracle (However, because of restrictions on the IND-CCA game, the adversary's service does not have access to the data generated by the adversary's client).

(2) The adversary sends plaintext m_0 and m_1 of the same length to the proxy as the client.

(3) The proxy randomly generates b from $\{0, 1\}$ and determines the plaintext m_b to be encrypted.

(4) The proxy encrypts plaintext m_b for one or more services excluding the adversary's one and uploads ciphertext to the cloud storage.

Let us consider the two scenarios as follows:

(1) Access the cyphertext by manipulating the data in the cloud storage. Here, if the adversary's advantage of discriminating b is sufficiently small to be ignored then the system is IND-CPA secure.

(2) Let the proxy decrypt any ciphertext adaptively and repeatedly by manipulating the data in the cloud storage. Here, when the system is IND-CPA secure and the adversary's advantage of discriminating b is sufficiently small to be ignored then system is IND-CCA2 secure.

First, in the case of 1, the adversary can obtain the encrypted data, which are the result of encrypting either plaintext m_0 or m_1 with a random key obtained using AES. Since the adversary cannot obtain the random key used for encryption assuming that AES is sufficiently secure, the adversary cannot obtain any advantage for discriminating the original plaintext. As of 2019, AES has not shown vulnerabilities to any attack so the system is considered to be IND-CPA secure.

Next, in the case of 2, the adversary can obtain the arbitrary ciphertext and key to the proxy, perform AES decryption, and derive the result. However, this would be the same as the adversary performing AES decryption himself. In addition, as the random key used for encryption is not stored in the proxy or cloud storages, the adversary cannot obtain any advantage for discriminating the original plaintext. Therefore, the system is considered to be IND-CCA2 secure.

5. Performance Evaluation

5.1 Experimental Setup

To evaluate the proposed scheme, we implemented its proxy in ASP.NET Core Web application using .NET Core 2 framework. The core part of encryption and decryption was realized using the NETCore.Encrypt library [21]. The Amazon S3 compatible object storage MinIO [22] was employed as the cloud storage. In addition, we implemented a console application acting as the client and service and measured processing times using the BenchmarkDotNet library [23]. We evaluated the performance in emulab d430 node using the image of Docker container host on Ubuntu. We used a d430 node for each client, proxy, and cloud storage, and interconnected them with a 10 Gbps network. In this implementation, the proxy employed the AES key with the length of 256 bits for data encryption and the RSA key with the length of 2,048 bits based on the optimal asymmetric encryption padding [24] encryption standard using the secure hash algorithm (SHA-512) [25] to encrypt DEK for the recipient services.

In the experimental results shown in the following subsections, the values of the experimental results of previous studies are listed for reference. However, please note that a simple comparison cannot be made for the experimental result values from each method because the experiments were conducted under different conditions.

5.2 Uploading and Downloading

Tables 1 and **2** represent the total time durations corresponding to uploading and downloading operations and provide a comparison with those observed in the alternative methods. The values of the previous studies in Tables 1 and 2 are cited from Refs. [13] and [14]. Additionally, **Fig. 6** is a graph of the upload and download times of the proposed method. As shown in the results, our method does not show an exponential increase in data transfer time as the size of the download/upload data increases, as in Ref. [26], but the transfer time increases almost in proportion to the data size. To measure the time durations of uploading and downloading operations, we employed three nodes corresponding to the client, proxy, and cloud storage. The size of the data to be

Table 1 Comparison of total uploading time in seconds.

Data [MB]	[11]	[12]	[13]	[14]	[26]	Our Scheme
10	13.05	14.95	6.43	0.59	14.59	1.06
50	53.68	58.56	9.01	2.72	60.37	2.52
100	99.69	112.41	17.37	4.48	155.15	4.73
500	369.72	492.03	33.24	17.43	872.09	19.03

Table 2 Comparison of total downloading time in seconds.

Data [MB]	[11]	[12]	[13]	[14]	[26]	Our Scheme
10	9.91	9.90	6.48	0.81	10.45	0.44
50	33.45	35.57	10.24	2.52	35.90	0.90
100	57.14	59.14	20.68	2.79	61.59	1.57
500	215.3	229.81	39.25	13.65	400.21	7.07

uploaded and downloaded was defined from 10 to 500 MB. The results measuring the upload time durations included the time period starting from the moment when the client initiated uploading until the data were encrypted on the proxy and uploaded into the cloud storage. Since data encryption was performed only once, regardless of the number of services that could access the data, in this measurement we set the number of services that could access the data as equal to one. The total key generation time for each service, which depended on the number of services that could access the data, was evaluated in the following Section 5.3. The results from measuring the download time included the time period starting from the moment when the service initiated requesting the data and until the data and key were downloaded from the cloud storage to the proxy, through decrypting the key by using the service's private key, decrypting the data by employing the decrypted key, and then, sending to the service.

5.3 Key Generation

Table 3 represents the total key generation time duration for each service, as well of those corresponding to the alternative methods. The values of the previous studies in Table 3 are cited from Refs. [13] and [14]. To measure duration of key generation, we extracted the key generation process from the proxy and estimated the time it took to complete the process. The number of services considered in key generation was from 10 to 100. In Ref. [14], the secret key generation time for each portion of data was 1.4 ms, and encryption of the secret key by using the public key, hash value generation, and signing took 4.8 ms. These time

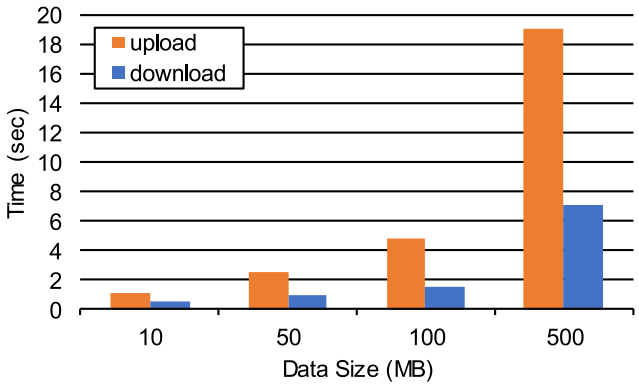


Fig. 6 Total uploading and downloading time.

Table 3 Comparison of key generation time in seconds.

No. of recipients	[11]	[12]	[13]	[26]	Our Scheme
10	1.494	1.594	0.00400	1.534	0.00048
50	1.907	1.952	0.00512	1.866	0.00119
100	2.495	2.887	0.00697	2.545	0.00218

duration values were approximately constant and did not change with an increase in the number of receivers, as the keys were centrally managed by the key generation server. Moreover, there was no need to generate a key for each recipient. Therefore, we did not include [14] in Table 3. It should be noted that the key generation server in Ref. [14] did not manage to resolve the key escrow problem.

5.4 Comparative Analysis

Table 4 shows the response status of each method to issues to be solved by data sharing methods for IoT environments. Our method covers all the functions required for IoT data sharing, and can provide the security required for the system even if the device of the data sender and/or receiver has limited computing resources.

After analyzing the results from comparing the proposed method with the alternative cloud-based secure data sharing schemes [11], [12], [13], [14], [26], we can confirm that the former is sufficiently faster compared with other existing methods since the main data encryption process uses AES, and the key generation for receivers employs simple RSA that does not require the pairing operation. Moreover, almost all encryption processing operations are executed within the proxy.

On the other hand, the values of the experimental results for each method were performed under different conditions. Therefore, note that these results are not intended to directly compare these values, but rather to understand the trends of each method.

5.5 Workload

5.5.1 Data Size

The information required for each client and service includes attribute information, ID, and access token. In addition, the public key information is necessary for services. The data size can be assumed to be at most approximately 3 KB, including the serialization overhead in most cases. This value is large enough for the data size of service information that the proxy needs to hold, assuming a typical IoT environment. **Figure 7** represents a graph of the number of clients or services and the corresponding theoretical value of the required data size. The result indicates that 30 GB storage can be used to store the information on approximately 10,000,000 clients or services.

5.5.2 Network Bandwidth

Since the network bandwidth of the entire system depends on use cases and scenarios, it is difficult to estimate it accurately. Here, as an assumption to be discussed, we assume that all clients generate 1 KB of data every 30 s and send it to the proxy. This value is large enough and frequent enough for data transmission by common IoT devices such as sensor devices. Under this as-

Table 4 Response status of each method to issues to be solved by IoT data sharing.

	[11]	[12]	[13]	[14]	[26]	Our Scheme
Capable of sending and receiving data on devices with limited computing resources.	×	×	✓	✓	×	✓
New recipients can be added without the sender continuously maintain the DEK of encrypted data.	×	×	×	✓	×	✓
No need or easy to distribute data decryption keys to data recipients.	✓	✓	×	✓	✓	✓
Addressing the key escrow problem.	✓	✓	✓	×	✓	✓
Provide forward secrecy for processed data.	✓	✓	✓	×	✓	✓
Easy to scale data store for storing keys.	✓	✓	✓	×	✓	✓

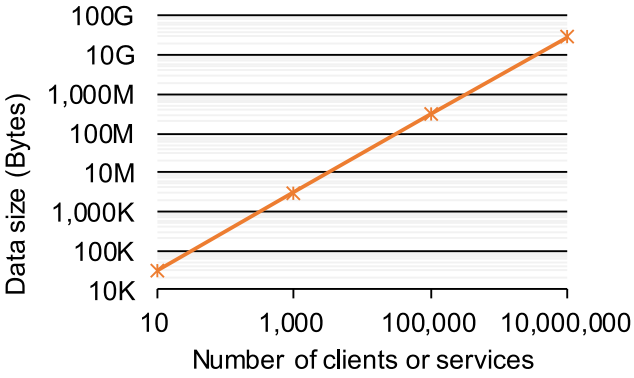


Fig. 7 The number of clients or services and required data size.

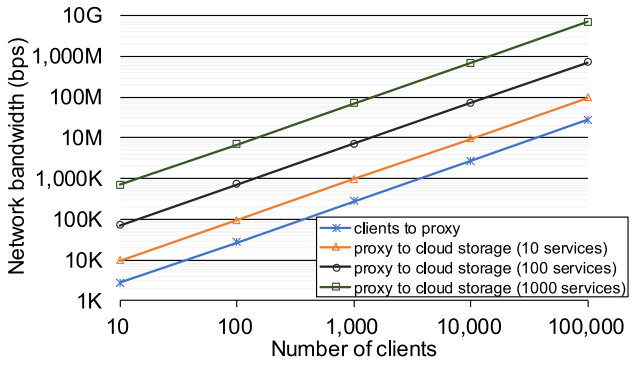


Fig. 8 The required network bandwidth.

sumption, 1 KB of data is required including overheads, such as the HTTP transport layer and AES encryption padding. The proxy uploads the AES encrypted data and additional DEKs encrypted using the public key for recipient services and initialization vector. Assuming the same conditions as in our implementation, the length of the DEK is considered equal to 256 bits and becomes 2,048 bits after being encrypted by RSA, and the length of the initialization vector is 128 bits. On the basis of these conditions, the theoretical values for the estimated network bandwidth required between the clients and proxy and between the proxy and cloud storage are calculated as shown in **Fig. 8**.

The obtained result shows that the proposed scheme can be used to perform secure data sharing between IoT devices using the 1 Gbps network bandwidth if the product of the number of clients and the number of services is 10,000,000 or less.

5.5.3 Encryption Processing

As described in Section 3, processing with a large amount of calculations such as data encryption and decryption is fully executed on the proxy side. Therefore, since the client does not need to perform any encryption processing on the generated data, even devices with limited processing resources can be employed as clients. Concerning services, the only cryptographic processing that needs to be executed is to decrypt DEK encrypted for the service with its own private key. Since DEK is rather small (256 bits in our implementation), this calculation amount is also deemed insignificant.

5.5.4 Massive Concurrent Access

When many clients and services access the proxy at the same time, the computational load, such as encryption and network processing performed by the proxy, may become a bottleneck to overall system performance. How much processing the proxy can handle in parallel depends on the performance of the server running the proxy. However, if the proxy has to serve a number of clients or services that cannot be handled by a single server, it can be handled by increasing the number of proxies. The number of proxies can be increased by deploying multiple proxies and load balancing access between proxies as appropriate for the communication protocol used in the proxy and client or service. To use multiple proxies, the access token, public key, and access policy must be replicated between each proxy and must use the same cloud storage for the data store.

6. Conclusion

In the present paper, we describe a novel proposed IoT-oriented secure data sharing scheme which is aimed to address the key escrow problem and many other issues related to the IoT environment presented in the existing secure data sharing methods, such as lack of computational resources, key management and distribution, and changing the access authority. In addition, the proposed method uses the cloud object storage used to manage and store the encrypted data and encrypted keys. Since the object storage is widely used in various cloud infrastructures, the proposed method can be easily applied to the existing cloud providers. The results of security analysis indicate that the proposed scheme is safe in terms of the key escrow problem and is capable of providing forward secrecy and ensuring INC-CCA2 equivalent safety. The performance evaluation denotes that the proposed method has equal or better performance compared with existing secure data sharing methods and is sufficiently practical. Moreover, the estimation based on realistic scenarios shows that the required computational resources are practical even in the case of the large-scale IoT environment. Since all computationally intensive processes such as data encryption and decryption are performed completely on the proxy side, our method can also be used on IoT devices with limited processing resources. One example of a practical application where our method can be implemented is the secure sharing of large-scale sensor data such as in agriculture or smart cities with multiple users based on policy-based access control.

Because IoT devices are distributed in different locations, these devices can be physically accessed by malicious third parties. As a result, the authentication information stored on the IoT devices might also leak out. As future work, we will try to apply our method to address this problem by implementing key rotation or statistical access analysis to prevent abuse of compromised keys or access tokens from IoT devices.

References

[1] Ministry of Internal Affairs and Communications: The 2018 White Paper on Information and Communications in Japan (2018).
[2] Alhamazani, K., Ranjan, R., Mitra, K., Ullah, S., Adnene, K. and Vasudha, G.: An overview of the commercial cloud monitoring tools: Research dimensions, design issues, and state-of-the-art, pp.357–377 (2015).
[3] Khan, A.N., Kiah, M.L.M., Khan, S.U. and Madani, S.A.: Towards secure mobile cloud computing: A survey, Future Generation Computer Systems (2013).
[4] Sandhu, R.S., Coyne, E.J.E., Feinstein, H.L. and Youman, C.E.C.: Role-based access control models, *Computer*, Vol.29, No.2, pp.38–47 (1996).

[5] Yuan., E. and Tong, J.: Attributed Based Access Control (ABAC) for web services, *Proc. 2005 IEEE International Conference on Web Services, ICWS 2005*, Vol.2005, pp.561–569 (2005).

[6] Yokogi, K., Kitagawa, N. and Yamai, N.: Access Control Model for IoT Environment Including Automated Configuration, *Proc. International Computer Software and Applications Conference*, Vol.2, pp.616–621 (2018).

[7] NIST: Advanced encryption standard (AES), US Department of Commerce, *National Institute of Standards and Technology*, Vol.56, pp.57–71 (2001).

[8] Rivest, R.L., Shamir, A. and Adleman, L.M.: Cryptographic communications system and method, US Patent 4,405,829 (1983).

[9] Elgamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Trans. Information Theory*, Vol.31, No.4, pp.469–472 (1985).

[10] Boneh, D. and Franklin, M.: Identity-Based Encryption from the Weil Pairing, *SIAM Journal on Computing*, Vol.32, No.3, pp.586–615 (2003).

[11] Xu, L. Wu, X. and Zhang, X.: CL-PRE: A Certificateless Proxy Re-Encryption Scheme for Secure Data Sharing with Public Cloud, *Proc. 7th ACM Symposium on Information, Computer and Communications Security - ASIACCS '12*, p.87 (2012).

[12] Seo, S.H. Nabeel, M. Ding, X. and Bertino, E.: An Efficient Certificateless Encryption for Secure Data Sharing in Public Clouds, *IEEE Trans. Knowledge and Data Engineering*, Vol.26, No.9, pp.2107–2119 (2014).

[13] Ali, M., Dhamotharan, R., Khan, E. Khan, S.U., Vasilakos, A.V., Li, K. and Zomaya, A.Y.: SeDaSC: Secure Data Sharing in Clouds, *IEEE Systems Journal*, Vol.11, No.2, pp.395–404 (2017).

[14] Mollah, M.B., Azad, A.K. and Vasilakos, A.: Secure data sharing and searching at the edge of cloud-assisted internet of things, *IEEE Cloud Computing*, Vol.4, No.1, pp.34–42 (2017).

[15] Krawczyk, H., Bellare, M. and Canetti, R.: HMAC: Keyed-Hashing for Message Authentication, RFC 2104 (1997).

[16] M'Raihi, D., Machani, S., Pei, M. and Rydell, J.: TOTP: Time-Based One-Time Password Algorithm, RFC 6238 (2011).

[17] M'Raihi, D., Bellare, M., Hoornaert, F., Naccache, D. and Ranen, O.: HOTP: An HMAC-Based One-Time Password Algorithm, RFC 4226 (2005).

[18] Simpson, W.: PPP Challenge Handshake Authentication Protocol (CHAP), RFC 1994 (1996).

[19] Rissanen, E.: eXtensible Access Control Markup Language (XACML) Version 3.0 Plus Errata 01 (2017) (online), available from ⟨http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-en.pdf⟩ (accessed 2018-01-08).

[20] Imran, M. and Hlavacs, H.: Searching in cloud object storage by using a metadata model, *Proc. 2013 9th International Conference on Semantics, Knowledge and Grids, SKG 2013*, pp.121–128 (2013).

[21] MyloveCc: NETCore.Encrypt: NETCore encrypt and decrpty tool. Include aes, des, rsa, md5, sha1, sha256, sha384, sha512 (online), available from ⟨https://github.com/myloveCc/NETCore.Encrypt⟩ (accessed 2019-11-28).

[22] MinIO: minio: MinIO is a high performance object storage server compatible with Amazon S3 APIs (online), available from ⟨https://github.com/minio/minio⟩ (accessed 2019-11-28).

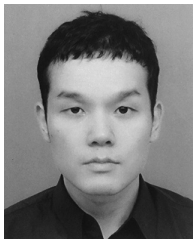
[23] Dotnet: BenchmarkDotNet: Powerful .NET library for benchmarking (online), available from ⟨https://github.com/dotnet/BenchmarkDotNet⟩ (accessed 2019-11-28).

[24] Bellare, M. and Rogaway, P.: Optimal Asymmetric Encryption - How to Encrypt with RSA, *Proc. International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*, *EUROCRYPT 1994*, Vol.950, pp.92–111 (1995).

[25] Dworkin, M.J.: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, Federal Information Processing Standards Publication, No.202 (2015).

[26] Khan, A.N., Kiah, M.L., Madani, S.A., Ali, M., Khan, A.U.R. and Shamshirband, S.: Incremental proxy re-encryption scheme for mobile cloud computing environment, *Journal of Supercomputing*, Vol.68, No.2, pp.624–651 (2014).

Kenta Yokogi received his B.E. and M.E. degrees in computer and information science from Tokyo University of Agriculture and Technology, in 2018 and 2020, respectively. Since April 2020, he has been with CRI Middleware Co., Ltd., Japan. His research interests include access control and management, secure data sharing, and Internet of Things (IoT).



Naoya Kitagawa received his B.Info.Sc. and M.Info.Sc. degrees in information science from Chukyo University, in 2009 and 2011, respectively, and his Ph.D. degree in information science from Nagoya University, in 2014. In April 2014, he joined the Information Technology Center, Nagoya University, as a Postdoctoral Fellow. In October 2014, he joined the Institute of Engineering, Tokyo University of Agriculture and Technology, as an Assistant Professor. Since April 2020, he has been a Project Associate Professor with the Research and Development Center for Academic Networks, National Institute of Informatics. His research interests include the Internet, network security, and distributed systems. He is a member of IEEE and IPSJ.

Nariyoshi Yamai received his B.E. and M.E. degrees in electronic engineering and his Ph.D. degree in information and computer science from Osaka University, Osaka, Japan, in 1984, 1986, and 1993, respectively. In April 1988, he joined the Department of Information Engineering, Nara National College of Technology, as a Research Associate, where he was an Assistant Professor, from April 1990 to March 1994. In April 1994, he joined the Education Center for Information Processing, Osaka University, as a Research Associate. In April 1995, he joined the Computation Center, Osaka University, as an Assistant Professor. From November 1997 to March 2006, he joined the Computer Center, Okayama University, as an Associate Professor. From April 2006 to March 2014, he was a Professor with the Information Technology Center (at present, the Center for Information Technology and Management), Okayama University. Since April 2014, he has been a Professor with the Institute of Engineering, Tokyo University of Agriculture and Technology. His research interests include distributed systems, network architecture, and Internet. He is a member of IEEE, IEICE and IPSJ.