Electronic Preprint for Journal of Information Processing Vol.28

Regular Paper

Public-key Searchable Encryption with Index Generation for Shared Database

Nori Matsuda[1,2,a)] Takato Hirano[1] Yutaka Kawai[1] Takashi Ito[1] Mitsuhiro Hattori[1]
Tadakazu Yamanaka[1] Masakatsu Nishigaki[2]

Received: December 9, 2019, Accepted: June 1, 2020

Abstract: Recently, use of cloud computing has expanded to include tasks such as data sharing among multiple enterprises for open innovations and IoT systems for device management, etc. Public-key searchable encryption is particularly useful for cases requiring protection of shared secret data. However, there is no public-key searchable encryption scheme that simultaneously achieves: (A) efficient search performance, (B) multi-user support and (C) implementation on database management systems, simultaneously. In this paper, we propose a new public-key searchable encryption scheme which (A) provides an index generation mechanism for public-key searchable encryption, and (B) generates different secret keys for multiple users. In our scheme, (A) is achieved by a portion of the deterministic values generated from a keyword, and (B) is achieved by hierarchical inner-product predicate encryption. Our scheme in particular, forms wildcards as the user identity from hierarchical inner-product predicate encryption in order to easily represent hierarchical identities. To achieve (C), we further propose an integration method which implements our scheme into database management systems through a user-defined function so that its search functionality can be easily used via SQL.

Keywords: searchable encryption, public-key encryption, index generation, database integration

1. Introduction

1.1 Background

In recent years, the utilization of cloud services for the early launch of services and lowering enterprise IT costs has become common. Use of the cloud has also expanded to include other areas. In the past, a single company made use of the cloud for its internal use or providing commercial services. Now however, usage has expanded beyond single company use to also include multi-enterprise data sharing. For example, as open innovation becomes more fully utilized among multiple enterprises, usage of the cloud is also increasing to share development documents among the enterprises. By using the cloud, information can not only be shared more efficiently among multiple enterprises, but there is also the advantage that information can be securely shared by granting access permission according to the affiliation of users. Another example is IoT systems that support device management. A lot of IoT systems that are built on the cloud can be accessed from anywhere to gather information from devices all over the world. In addition, all stakeholders involved in the product life cycle, such as designers, manufactures, sales and maintenance personnel, will be able to view data within their own affiliations, thus facilitating the provision of various services.

In order to realize the above examples, highly secure cloud services should be used. One method for securing cloud services is encryption. Especially, *searchable encryption* which can perform keyword search over encrypted data without decryption is suitable for cloud services. The first searchable encryption scheme proposed in Ref. [28] is based on symmetric-key cryptography. After that, many schemes have been proposed to improve the performance or security [13], [14], [31], or to realize order preserving encryption and its variants by relaxing IND-CPA security [2], [7], [8], [17], [19]. In addition to algorithmic constructions of symmetric-key searchable encryption, its integration into database management systems was also studied in Refs. [24], [29].

On the other hand, a lot of symmetric-key searchable encryption schemes have to share among users a unique secret key which is used for performing keyword search. This key-sharing limitation is acceptable for data sharing within a single company but not for the multi-enterprise data sharing described in our examples. This is because fine-grained access control for each data is strictly required under multi-enterprise (i.e., multi-user) situations so that only appropriate users can access (in other words, decrypt) the data. Furthermore, a data owner would designate access permissions of his data by using not only the user IDs but also affiliations or project names and then encrypt the data with the access permission. Then, anyone who has the above access permissions can search the encrypted data with each individual's private key. However, constructing a searchable encryption scheme that supports the above situation from symmetric-key primitives is an extremely difficult task. Therefore, symmetric-

[1] Mitsubishi Electric Corporation, Kamakura, Kanagawa 247–8501, Japan
[2] Graduate School of Science and Technology, Shizuoka University, Hamamatsu, Shizuoka 432–8011, Japan
[a)] Matsuda.Nori@ea.MitsubishiElectric.co.jp

© 2020 Information Processing Society of Japan

key searchable encryption is not suitable for those situations.

In those situations, public-key searchable encryption is suitable for ensuring security in multi-enterprise cases. Performance improvements have been proposed for public-key searchable encryption schemes to make them more practical [5], [6] while most searchable encryption needs to check all encrypted data sequentially because of its strong security (i.e., indistinguishability). Public-key searchable encryption that supports multi-user data sharing and its concrete scheme has been also proposed [18]. However, there is currently no public-key searchable encryption scheme which satisfies both efficient performance and multi-user data sharing. Moreover, previous works focused only on algorithmic improvements, but not on database integration for deployment to Web systems.

This paper therefore considers the following requirements which should be satisfied to make public-key searchable encryption practical:

Requirement A: Sufficient Search Performance.
Provide schemes in which the computational complexity of performing keyword search is sub-linear of the amount of shared data.

Requirement B: Data Sharing among Multiple Users.
Provide schemes which support data sharing according to users' identities.

In order to apply the above to actual web systems, the following requirement should be also satisfied:

Requirement C: Integration into Database.
Provide integration methods into database management systems so that web applications can call public-key searchable encryption via SQL.

1.2 Issues of Previous Searchable Encryption Schemes

The first searchable encryption scheme based on public-key cryptography was proposed in Ref. [9]. A major advantage of the scheme is that anyone can encrypt data with a public key and that only a user having its corresponding secret key can perform keyword search over the encrypted data. Almost all public-key searchable encryption schemes extract keywords from data that a user wants to store, and generate its encrypted data and their encrypted keywords (called *tag*s) in the storing phase. The user conducts keyword search of the tags by using an encrypted keyword (called *trapdoor*) in the search phase. After its proposal, various public-key searchable encryption schemes with rich functionalities such as AND/OR search or range search, have been proposed [10], [15], [20], [21], [22]. A typical example of them is constructed from hidden vector encryption (HVE). While all of the schemes achieve high security (IND-CPA secure), they, however, have efficiency and usefulness problems. Informally, these search performances linearly depend on the amount of encrypted data stored in the clouds (Requirement A) and data sharing among multiple users is not possible (Requirement B). In addition to the problems, none of the papers (except for Refs. [23] and [29]) describe application of the schemes to a conventional database (Requirement C).

There are two approaches for performance improvement of public-key searchable encryption. One approach is (a) improving the keyword matching algorithm itself, and the other approach is (b) avoiding sequential search of the tags in the search phase. (e.g. using an index) Especially, the approach described in (b) is necessary for Requirement A because a large amount of data is stored on the database in actual use-cases.

Table 1 Previous searchable encryption schemes.

	Requirement A	Requirement B	Requirement C
Shi et al.			
Bellare et al.	✓		
Popa et al.	✓		✓
Hattori et al.		✓	
Suzuki et al.			✓
Our proposal	✓	✓	✓

In order to improve the search performance regarding to Requirement A, Shi et al. [27] proposed single or multi-dimensional range query schemes which can perform range matching between a tag and a query in $O(\log |S|)$ time, where $S \subseteq \mathbb{Z}$ is their message space. To realize the schemes, they proposed a new security definition called "match-revealing" by relaxing IND-CPA security. The "match-revealing" means that the hidden values of encrypted data which are within a queried range, might be leaked after the range search, while the other hidden values outside of the range have achieved IND-CPA security, yet. However, the search complexity for all tags stored in database is $O(N \times \log|S|)$, where N is the number of tags in the database.

Bellare et al. [5], [6] proposed deterministic searchable encryption schemes based on RSA-OAEP. Although the search process of the schemes can work in logarithmic order of the amount of encrypted data, there is no data sharing capability among multiple users.

Popa et al. [24] proposed the database system called CryptDB for improving performance and usefulness of searchable encryption and for realizing its database integration. Since it deployed symmetric-key searchable encryption, data sharing among multiple users is realized by sharing the same key. Therefore, CryptDB cannot essentially provide secure data sharing among multiple users.

Regarding Requirement B, Hattori et al. [18] proposed the public-key searchable encryption scheme which can support secure data sharing among multiple users. However, their scheme has the disadvantage that the search performance linearly depends on the amount of encrypted data, similarly to the scheme [9].

Regarding Requirement C, we [23] proposed the database integration of public-key searchable encryption in 2013. Suzuki et al. [29] proposed generalized methodology to integrate searchable encryption into database management systems. However, their methodology focused only on symmetric-key searchable encryption, but not public-key searchable encryption. This method could also apply to this encryption, however they never implemented a prototype for this encryption.

Previous works are summarized in **Table 1**. As shown in the table, no previous work meets all the requirements, simultaneously, although some of the requirements have been achieved.

1.3 Our Contribution

In this paper, we propose *Public-key Searchable Encryption*

© 2020 Information Processing Society of Japan

with Index Generation schemes. The first step scheme satisfies Requirement A, and the second step scheme satisfies both Requirements A and B. We also propose an implementation for integration of the proposed scheme into database management systems to satisfy Requirement C.

Regarding Requirement A, by applying the concept of generalization used in data anonymization [25], we propose the first step scheme which has an index generation mechanism for public-key searchable encryption and an adjusting mechanism between security and performance. Concretely, at the storing phase, keywords are generalized and grouped, and then a group ID is given to each group. The group ID is our index value. We construct the group ID from which the keywords cannot be uniquely identified. At the search phase, the group ID is also used to improve the search performance. In other words, a user specifies a target group ID for search, and then performs a keyword search only for encrypted data with the group ID.

In addition to the index construction above, the group ID is encrypted and then stored in the server. The group ID is partially disclosed by an index extraction key as necessary. By adjusting the number of index extraction keys, the user can adjust the balance between the security and the performance.

We also propose a new security definition which ensures that keywords are not distinguishable from one another if both keywords have the same index value. This is a less severe IND-CPA security definition which combines IND-CPA with anonymity.

Regarding to Requirement B, we propose the second step scheme which has a multi-user mechanism based on the concept of hierarchical ID. By using the hierarchical ID, not only the user ID but also the affiliation or project name to which the user belongs can be expressed. However, only applying a hierarchical ID is not enough for the multi-enterprise situations since typical hierarchical ID-based encryption (HIBE) can designate a specific user, but not multiple group members as recipients. In other words, if we use HIBE to share the data, a data owner has to encrypt the data to match every recipient's identity. The size of the encrypted data then grows linearly depending on the number of recipients. In order to extend the concept of hierarchical ID to multi-user, we apply the concept of wild-card to the hierarchical ID. Since the wildcards "$*$" corresponds to all user IDs, the data owner can simultaneously designate multiple users who belong to an affiliation or a project. We use inner-product predicate encryption [22] to realize such an HIBE scheme supporting the wild-cards. Then, we assume there is a key administrator who manages the master secret key and issues user private keys, similarly to ordinary IBE and its variants. In addition, we also extend our security definition from single-user setting to multi-user setting.

Regarding Requirement C, we propose integrating our proposed scheme into conventional database management systems in order to apply our scheme to web application systems. More precisely, a tag (including an encrypted index) is stored together with their corresponding encrypted data in the database. When a search query is received, an index value is extracted from the search query. Since selecting rows with the index value can be achieved, it is easy to reduce the amount of the encrypted data that we need to check. These processes can achieve user-defined functions or user-defined data types, and therefore web applications can easily call searchable encryption functions via extended SQL statements.

1.4 Organization of the Paper

In Section 2, we describe details of related works. In Section 3, we describe a definition of conventional searchable encryption schemes. In Sections 4, 5 and 6, we present step by step, our public-key searchable encryption with index generation. For the first step, we present a scheme that satisfies only Requirement A in Section 4. For the second step, we extend our scheme to satisfy both Requirements A and B in Section 5. In Section 6, for the final step, we propose database integration to satisfy all the Requirements A, B and C. Evaluations of our index function and performance are also shown in this section.

2. Related Work

Although we briefly show that there is no proposal that satisfies all the Requirements A, B, and C in Section 1.2, we describe more details.

Shi et al. [27] proposed single or multi-dimensional range query schemes which can perform range matching between a tag and a query in $O(\log|S|)$ time, where $S \subseteq \mathbb{Z}$ is their message space, while an HVE based range search scheme proposed by Boneh et al. [10] requires $O(|S|)$ computations. Their idea is that they build a balanced tree which represents integer values as a leaf node. Instead of searchable encrypting the integer value directly, leaf node IDs and all of its ancestor node IDs are encrypted by searchable encryption. To search the values, a search range is converted into the combination of node IDs, and trapdoors are generated from all of the node IDs corresponding to the search range. Since the balanced tree is used for expressing the integer values, computational complexity becomes logarithmic order. To realize the schemes, they proposed a new security definition called "match-revealing" by relaxing IND-CPA security. The "match-revealing" means that the hidden values of encrypted data which are within a queried range, may be leaked after the range search, while the other hidden values outside of the range are still guaranteed IND-CPA security. However, the search complexity for all tags stored in database is $O(N \times \log|S|)$, where N is the number of data in the database.

Bellare et al. [5], [6] proposed deterministic searchable encryption schemes based on RSA-OAEP, and showed that their search process can be performed in the logarithmic order of the amount of data. In this method, a keyword is used as a seed of the pseudo-random number generator for OAEP padding, and thus their algorithms become deterministic. Then, conducting a keyword search can be done by using binary comparison operation between a tag and a trapdoor Therefore, search operation can be performed at high speed by utilizing the optimization functionality that the ordinary database has. However, since the algorithm is based on RSA encryption, data sharing among multiple users cannot be realized.

Popa et al. [24] have proposed a database called CryptDB for improving the performance of searchable encryption and its

© 2020 Information Processing Society of Japan

database integration. In that paper, they proposed an onion encryption that uses both deterministic and probabilistic searchable encryption schemes to provide a trade-off mechanism between performance and security. Passing an onion slice key from the client to the server that converts the probabilistic encryption into the deterministic encryption can add to search capability, together with degrading security. Since it was implemented as a proxy, secret-key is shared among multiple users in an enterprise, and then data can be shared. However, since AES and symmetric-key searchable encryption are used, it is not possible to securely share data among multiple enterprises. In addition, these processes were realized by user-defined functions, and they showed that symmetric-key searchable encryption can be applied to ordinary databases. However, they did not discuss whether public-key searchable encryption can be applied or not.

Hattori et al. [18] proposed a scheme to realize data sharing in public-key searchable encryption. The scheme is constructed by extending hierarchical ID-based encryption with wildcards to satisfy ID confidentiality (attribute-hiding). Therefore, based on the general construction scheme of searchable encryption from attribute-hiding identity based encryption proposed by Abdalla et al. [1], searchable encryption for multi-user setting can be realized. Although this scheme realizes data sharing among multiple users, it has performance issues similar to ordinary public-key searchable encryption schemes.

Suzuki et al. [29] also proposed a general construction method for applying searchable encryption to Web applications. A DB plug-in is used to integrate searchable encryption into a database and a DB interface wrapper is used to convert an ordinary SQL command into a special SQL command which supports searchable encryption in order to reduce modification of the Web application. Encryption and decryption are performed by the HTTP Proxy on the client side. Their implementation method was confirmed by the symmetric-key searchable encryption. This method may be applicable to public-key searchable encryption but has not been verified by prototype implementation.

3. Preliminaries

3.1 Notations

When A is a set, $y \xleftarrow{U} A$ denotes that y is uniformly selected from A. $y := z$ denotes that y is set, defined or substituted by z. When a is a fixed value and A is an algorithm, $A(x) \to a$ denotes the event that A outputs a on input x. We denote a set of all natural numbers by $\mathbb{N}$. For a prime number $q \in \mathbb{N}$, we denote a finite field of order q by $\mathbb{F}_q$. A vector symbol denotes a vector representation over $\mathbb{F}_q$, e.g., $\overrightarrow{x}$ denotes $(x_1, \ldots, x_n) \in \mathbb{F}_q^n$. We denote the message space for HIPE by M_{HIPE}, and the keyword space for searchable encryption by $W \subseteq \mathbb{F}_q \setminus \{0, q-1\}$. We denote a hierarchical identity by $\{identity_1, \cdots, identity_n\}$, where $identity_i$ is an element of $\mathbb{F}_q \setminus \{0, q-1\}$ for $1 \leq i \leq n$. A single quoted string '*string*' is represented as an element in $\mathbb{F}_q \setminus \{0, q-1\}$. We denote the concatenation of strings or identities by $|$. We say that a probability is negligible if the probability is smaller than $1/p(\lambda)$ for any positive polynomial p and any security parameter λ. We say that a probability is overwhelming if the probability is at least $1 - 1/p(\lambda)$ for any positive polynomial p and any security parameter λ.

3.2 Searchable Encryption

Searchable encryption is an encryption scheme that allows keyword search over encrypted data without decryption. There are two types in searchable encryption. One is based on public-key cryptography, and the other is based on symmetric-key cryptography. In this paper, we focus on the former searchable encryption. We show the definition of public-key searchable encryption proposed in Ref. [9].

Definition 1. A non-interactive public-key searchable encryption scheme consists of the following polynomial time algorithms:

- **KeyGen** takes as input the security parameter 1^λ, and generates a public/secret key pair (mpk, msk).
- **GenTag** takes as input the public key mpk and a keyword w, and generates an encrypted tag of w.
- **GenTrapdoor** takes as input the public key mpk, the secret key msk, and a keyword w', and generates a trapdoor $Td_{w'}$ of w'.
- **Test** takes as input the public key mpk, the encrypted tag Tag_w, and the trapdoor $Td_{w'}$, and outputs TRUE if $w = w'$ and FALSE otherwise.

The correctness of the searchable encryption scheme is as follows. For any security parameter 1^λ and any keyword w, it holds that $\mathbf{Test}(mpk, Tag_w, Td_w) = TRUE$ where $\mathbf{KeyGen}(1^\lambda) \to (mpk, msk)$, $\mathbf{GenTag}(mpk, w) \to Tag_w$ and $\mathbf{GenTrapdoor}(mpk, msk, w) \to Td_w$. In addition, the consistency is $\mathbf{Test}(mpk, Tag_w, Td_{w'}) = FALSE$ with overwhelming probability, where $w \neq w'$, $\mathbf{GenTag}(mpk, w) \to Tag_w$ and $\mathbf{GenTrapdoor}(mpk, msk, w') \to Td_{w'}$.

As described above, the searchable encryption scheme enables the user, who has the master secret key, to generate trapdoors.

The indistinguishable security of the searchable encryption scheme is defined as follows.

Definition 2. A searchable encryption scheme is secure in the sense of indistinguishability against adaptive chosen keyword attacks if for any polynomial time adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ in the following experiment is negligible in the security parameter.

(1) The challenger runs the $\mathbf{KeyGen}(1^\lambda)$ algorithm to generate a public key mpk and a secret key msk, and gives mpk to the adversary $\mathcal{A}$.
(2) The adversary $\mathcal{A}$ can adaptively ask the challenger for any keyword w of his choice in order to obtain its trapdoor Td_w.
(3) At some point, the adversary $\mathcal{A}$ sends two challenge keywords w_0^* and w_1^* to the challenger. The only restriction is that the adversary $\mathcal{A}$ did not previously ask for the trapdoor $Td_{w_0^*}$ or $Td_{w_1^*}$. The challenger picks a random bit $b \in \{0, 1\}$ and gives the adversary the challenge ciphertext $C = \mathbf{GenTag}(mpk, w_b^*)$.
(4) The adversary can continue to ask for any keyword w of his choice to obtain its trapdoor as long as $w \neq w_0^*$ and $w \neq w_1^*$.
(5) Finally, the adversary $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$ and wins the game if $b' = b$.

The advantage of $\mathcal{A}$ in this game is defined to be $\mathbf{Adv}_{\mathcal{A}}^{SE}(\lambda) = |\Pr[b' = b] - 1/2|$.

© 2020 Information Processing Society of Japan

3.3 Hierarchical Inner-product Predicate Encryption

We give the syntax of hierarchical predicate encryption proposed in Ref. [22]. In this paper, we focus on hierarchical inner-product predicate encryption (HIPE).

Definition 3. Let $\overrightarrow{\mu} := (n, d; \mu_1, \ldots, \mu_d)$ s.t. $0 < \mu_1 < \mu_2 < \cdots < \mu_d = n$ be a format of hierarchy of depth d attribute spaces. A hierarchical predicate encryption (HPE) scheme for the class of hierarchical inner-product predicates F over the set of hierarchical attributes Σ consists of probabilistic polynomial-time algorithms **Setup**, **KeyGen**, **Enc**, **Dec**, and **Delegate**$_\ell$ for $\ell = 1, \ldots, d-1$. These are given as follows:

- **Setup** takes as input a security parameter 1^λ and a format of hierarchy $\overrightarrow{\mu}$, and returns a (master) public key mpk and a (master) secret key msk.
- **KeyGen** takes as input the master public key mpk, the secret key msk, and predicate vectors $(\overrightarrow{v}_1, \cdots, \overrightarrow{v}_\ell)$. It returns a corresponding secret key $sk_{(\overrightarrow{v}_1, \cdots, \overrightarrow{v}_\ell)}$.
- **Enc** takes as input the master public key mpk, attribute vectors $(\overrightarrow{x}_1, \cdots, \overrightarrow{x}_h)$, where $1 \le h \le d$, and a plaintext m in the associated plaintext space, **msg**. It returns a ciphertext c.
- **Dec** takes as input the master public key mpk, the secret key $sk_{(\overrightarrow{v}_1, \cdots, \overrightarrow{v}_\ell)}$, where $1 \le \ell \le d$, and a ciphertext c. It returns a either plaintext m or the distinguished symbol $\perp$.
- **Delegate**$_\ell$ takes as input the master public key mpk, an ℓ-th level secret key $sk_{(\overrightarrow{v}_1, \cdots, \overrightarrow{v}_\ell)}$, and an $(\ell+1)$-th level predicate vector $\overrightarrow{v}_{\ell+1}$. It returns $(\ell+1)$-th level secret key $sk_{(\overrightarrow{v}_1, \cdots, \overrightarrow{v}_{\ell+1})}$.

The correctness of the HIPE is: For any security parameter 1^λ, any predicate vector $(\overrightarrow{v}_1, \cdots, \overrightarrow{v}_\ell)$ and any attribute vector $(\overrightarrow{x}_1, \cdots, \overrightarrow{x}_\ell)$, it holds that:

$$\begin{aligned}
&\textbf{Dec}(mpk, sk_{(\overrightarrow{v}_1, \cdots, \overrightarrow{v}_\ell)}, c) \to m,\\
&\text{s.t. } \overrightarrow{v}_i \cdot \overrightarrow{x}_i = 0 \ (1 \le i \le \ell),\\
&\text{where } \textbf{Setup}(1^\lambda, \overrightarrow{\mu}) \to (mpk, msk),\\
&\textbf{Enc}(mpk, (\overrightarrow{x}_1, \cdots, \overrightarrow{x}_\ell), m) \to c,\\
&\textbf{KeyGen}(mpk, msk, (\overrightarrow{v}_1, \cdots, \overrightarrow{v}_k)) \to sk_{(\overrightarrow{v}_1, \cdots, \overrightarrow{v}_k)},\\
&\text{and } \textbf{Delegate}_j(mpk, sk_{(\overrightarrow{v}_1, \cdots, \overrightarrow{v}_j)}) \to sk_{(\overrightarrow{v}_1, \cdots, \overrightarrow{v}_{j+1})}.
\end{aligned}$$

In addition, the consistency is: $\textbf{Dec}(mpk, sk_{(\overrightarrow{v}_1, \cdots, \overrightarrow{v}_\ell)}, c) \to \perp$ with overwhelming probability, in case $\overrightarrow{v}_i \cdot \overrightarrow{x}_i \neq 0$ for at least one of i.

The following security notions *payload-hiding* and *attribute-hiding* for HIPE are important to construct our scheme. The definitions are almost the same except for the challenge phase. More specifically, the adversary $\mathcal{A}$ chooses the challenge messages m_0^* and m_1^* for payload hiding security while the adversary $\mathcal{A}$ chooses the challenge attributes $\overrightarrow{x}_0^*$ and $\overrightarrow{x}_1^*$ for attribute hiding security.

Definition 4. A hierarchical inner-product predicate encryption scheme is payload-hiding (PH) against chosen plaintext attacks if for all probabilistic polynomial-time adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the following experiment is negligible in the security parameter.

(1) **Setup** is run to generate a pair of mpk and msk, and mpk is given to $\mathcal{A}$.

(2) $\mathcal{A}$ may adaptively makes a polynomial number of queries of the following type:
- Create key: $\mathcal{A}$ asks the challenger to create a secret key for a predicate $f \in F$. The challenger creates a key for f without giving it to $\mathcal{A}$.
- Create delegated key: $\mathcal{A}$ specifies a key for predicate f that has already been created, and asks the challenger to perform a delegation operation to create a child key for $f' \le f$. The challenger computes the child key without giving it to the adversary.
- Reveal key: $\mathcal{A}$ asks the challenger to reveal an already-created key for predicate f.

(3) $\mathcal{A}$ outputs challenge attribute vectors $\overrightarrow{x} := (\overrightarrow{x}_1, \ldots, \overrightarrow{x}_h)$ and challenge plaintext m_0^* and m_1^*, subject to the restriction that all the revealed keys which have been already queried cannot decrypt the challenge message.

(4) A random bit b is chosen. $\mathcal{A}$ is given $c^{(b)} := \textbf{Enc}(mpk, m_b^*, \overrightarrow{x})$.

(5) The adversary may continue to request keys for additional predicate vectors subject to the restriction that all the revealed keys which have been already queried cannot decrypt the challenge message.

(6) $\mathcal{A}$ outputs a bit b', and succeeds if $b' = b$.

The advantage of $\mathcal{A}$ in this game is defined to be $\textbf{Adv}_{\mathcal{A}}^{HIPE,PH}(\lambda) = |\Pr[b' = b] - 1/2|$.

Definition 5. A hierarchical inner-product predicate encryption scheme is attribute-hiding (AH) against chosen plaintext attacks if for all probabilistic polynomial-time adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the following experiment is negligible in the security parameter.

(1) **Setup** is run to generate a pair of mpk and msk, and mpk is given to $\mathcal{A}$.

(2) $\mathcal{A}$ may adaptively makes a polynomial number of queries of the following type:
- Create key: $\mathcal{A}$ asks the challenger to create a secret key for a predicate $f \in F$. The challenger creates a key for f without giving it to $\mathcal{A}$.
- Create delegated key: $\mathcal{A}$ specifies a key for predicate f that has already been created, and asks the challenger to perform a delegation operation to create a child key for $f' \le f$. The challenger computes the child key without giving it to the adversary.
- Reveal key: $\mathcal{A}$ asks the challenger to reveal an already-created key for predicate f.

(3) $\mathcal{A}$ outputs challenge attribute vectors $\overrightarrow{x}_0^* := (\overrightarrow{x}_1^{(0)}, \ldots, \overrightarrow{x}_{h_0}^{(0)})$ and $\overrightarrow{x}_1^* := (\overrightarrow{x}_1^{(1)}, \ldots, \overrightarrow{x}_{h_1}^{(1)})$ and a challenge plaintext m, subject to the restriction that $f(\overrightarrow{x}_0^*) = f(\overrightarrow{x}_1^*)$ for all revealed keys of predicate f.

(4) A random bit b is chosen. $\mathcal{A}$ is given $c^{(b)} := \textbf{Enc}(mpk, m, \overrightarrow{x}_b^*)$.

(5) The adversary may continue to request keys for additional predicate vectors subject to the restriction that $f(\overrightarrow{x}_0^*) = f(\overrightarrow{x}_1^*)$.

(6) $\mathcal{A}$ outputs a bit b', and succeeds if $b' = b$.

The advantage of $\mathcal{A}$ in this game is defined to be $\textbf{Adv}_{\mathcal{A}}^{HIPE,AH}(\lambda) = |\Pr[b' = b] - 1/2|$.

The HIPE scheme proposed in Ref. [22] satisfies both payload-hiding and attribute-hiding. Therefore, we use it as a building block of our scheme later on.

© 2020 Information Processing Society of Japan

3.4 Construction of Searchable Encryption from HIPE

Similarly to the generic construction [1] which can transform ID-based encryption into searchable encryption, we can construct a single-user searchable encryption scheme from HIPE schemes, as follows.

- **Setup**$(1^\lambda) \to (mpk, msk)$
  $\overrightarrow{\mu} := (n = 2, d = 1; \mu_1 = 2)$
  return $(mpk, msk) :=$ **Setup**$_{HIPE}(1^\lambda, \overrightarrow{\mu})$
- **GenTag**$(mpk, w) \to Tag_w$
  $r \xleftarrow{U} M$, $c :=$ **Enc**$_{HIPE}(mpk, \overrightarrow{x} := (w, 1), r)$
  return $Tag_w := (r, c)$
- **GenTrapdoor**$(mpk, msk, w') \to Td_{w'}$
  return $Td_{w'} :=$ **KeyGen**$(mpk, msk, \overrightarrow{v} := (1, -w'))$
- **Test**$(mpk, Tag_w, Td_{w'}) \to \{0, 1\}$
  $m' :=$ **Dec**$(mpk, Td_{w'}, c)$
  If $m' = r$, $result :=$ TRUE, otherwise $result :=$ FALSE
  return $result$

Broadly speaking, the encrypted tag is an HIPE ciphertext generated by the keyword w and the randomly chosen message r, where w is embedded as the attribute and r is also included in the encrypted tag. The trapdoor is a secret key generated by the keyword w' which is set as the predicate. Therefore, if both the attribute w and the predicate w' are identical, then the message r randomly chosen in **GenTag** appears in the decryption process of **Test**. Here, no information on w is revealed from the **Test** process if the HIPE scheme used in the construction satisfies attribute-hiding. The construction in this way realizes that an encrypted keyword search is to be conducted. While the construction leads only to single user setting, we extend it to multi-user setting in Section 5.

4. Performance Improvement

In this section, we present the first step scheme satisfying Requirement A.

4.1 Idea for Performance Improvement

We explain our idea for realizing the index generation capabilities. In conventional searchable encryption schemes, it is necessary to perform **Test** for all the encrypted tags with the trapdoor in one search process, since no information on keywords can be obtained from the encrypted tags. Therefore, the computation complexity of the search process becomes $O(N)$ in case the number of encrypted tags is N.

To improve the performance issues, we deploy index generation capabilities into public-key searchable encryption. Our key idea is that a user who has mpk generates, in addition to an encrypted tag, an index value derived from a keyword, deterministically, and stored together it with the encrypted tag in a server. Also, a user who has msk generates, in addition to a trapdoor, an index value from a keyword, deterministically, similarly to the above index generation. Hereafter, we call the former index *tag index* and the latter index *trapdoor index*. Prior to processing all encrypted tags, the server firstly selects candidates of encrypted tags which may match the hidden keyword of the trapdoor, from all the encrypted tags stored in the server, by using the tag index values and the trapdoor index value. After that, the server runs **Test** among the candidates and the trapdoor. In this situation, the server runs **Test** only for a part of the encrypted tags, but not for all encrypted tags. Thus, we can improve the search performance of public-key searchable encryption.

This key idea is similar to generalization technique well-known in anonymization literature. Here, we explain the above by using examples of Japanese addresses. Let a database have an address column. For privacy protection, detailed address information should be encrypted because the address information is a kind of personal information which might identify an individual. On the other hand, it might not be a problem if non-detailed address information is disclosed. For example, even if the prefecture information is disclosed, the risk of identifying the individual is extremely small. Therefore, the prefecture information may not need to be encrypted. In such a case, it is possible to use prefecture information as an index value for encrypted address. Even in Tokyo, which has the largest population in Japan, its population is only about 10% of the total population of Japan. Therefore, even if "Tokyo" is specified as an index value, the encrypted tags to be searched can be reduced to 10% of all encrypted tags, and the performance can be improved by 10 times while ensuring a certain level of anonymity.

We also propose a mechanism to control the disclosure depth of the index. This means that we can control the generalization levels. If a system is used for a long time, it is generally difficult to predict how large the database grows. In other words, it is difficult for an administrator to determine a suitable generalization level in advance. A mechanism which can adjust the generalization level according to the size of the database is useful for this situation. In order to realize the mechanism, our idea is that in the encryption process, index values are generated for each depth, encrypted, and embedded into an encrypted tag. If the administrator would like to improve the search performance, he permits the database to extract some index values by disclosing some decryption keys and decrypting the encrypted index values contained in the encrypted tags. Search performance is then appropriately improved depending on the disclosed depth of the index.

For example, in a case where the number of encrypted tags is small, the search process may finish in a practical amount of time even if all the encrypted tags are tested with a trapdoor. In this case, there is no need to disclose any index value about which prefecture the hidden address information of each encrypted tag belongs to. On the other hand, it takes time for a search since encryption tags gradually accumulate. Then, the search performance can be improved by disclosing which area (e.g., East Japan or West Japan) the hidden address information of each encrypted tag belongs to. Similarly, when further encrypted tags are accumulated and search processing time becomes long, region information such as Kanto region, Kinki region, etc. or more information such as for prefectures can be further disclosed in order to improve the search performance. In this way, we can control the disclosure depth of our index. In other words, our mechanism can control the tradeoff between security and search performance.

While we have explained our idea using Japanese address information so far, our idea can be applied to various data includ-

© 2020 Information Processing Society of Japan

ing the address information. Let an index function used in our **GenTag** and **GenTrapdoor** be **Index** : $\{0,1\}^* \times \{1,\ldots,L_{index}\} \to \{0,1\}^j$, where j is a size of the index values, L_{index} is the maximum depth of the index. That is, **Index**$(w,\ell) \to idx_\ell$, where w is a keyword and idx_ℓ is an element in $\{0,1\}^j$ for $1 \le \ell \le L_{index}$. Here, we assume that for a keyword w, there exists a keyword $w'(\neq w)$ such that **Index**(w,ℓ) = **Index**(w',ℓ) and that we can compute the keyword w' in polynomial time. We discuss these assumptions in Section 4.3. By disclosing the i-th element idx_i, we can divide the keyword space W into k_i groups. In our address example, $L_{index} = 3$, $k_1 = 2$, $k_2 = 8$, and $k_3 = 47$ because there are 2 areas, 8 regions, and 47 prefectures in Japan, respectively. In other words, $idx_1 \in \{0,1\}$, $idx_2 \in \{0,\ldots,7\}$ and $idx_3 \in \{0,\ldots,46\}$. Here, we assume that the index function equally divides the keyword space W within the range $\pm\delta$, where $\delta \in \mathbb{N}$. Then, each group of k_1 groups contains at least $|W|/k_1 - \delta$ keywords. This is because it holds in our example, $|\{w \in W \mid idx_1 = 0\}| \ge |W|/k_1 - \delta$ and $|\{w \in W \mid idx_1 = 1\}| \ge |W|/k_1 - \delta$ due to the above assumption. Therefore, even if ind_1 of any encrypted tag (i.e. tag index) is disclosed, there are at least $|W|/k_1 - \delta$ candidate keywords for the hidden keyword of the encrypted tag, and then $(|W|/k_1 - |\delta|)$-anonymity holds. The above situation also holds in the case where ind_1 of any trapdoor (i.e. trapdoor index) is disclosed. The number of the encrypted tags to be tested are reduced from $O(N)$ to $O(N/k_1)$, and therefore the search performance is improved.

4.2 Syntax

We show the syntax of our first step scheme for Requirement A.

Definition 6. A public-key searchable encryption with index generation for single-user setting consists of probabilistic polynomial-time algorithms **Setup**, **GenTag**, **GenTrapdoor**, **Test**, **GenIndexKey**, **ExtractTagIndex** and **ExtractTrapdoorIndex**. These are given as follows:

- **Setup** takes as input a security parameter 1^λ and a depth of index L_{index}. It returns a master public key mpk and a master secret key msk.
- **GenTag** takes as input the master public key mpk and a keyword w in some associated keyword space W. It returns encrypted tag Tag_w.
- **GenTrapdoor** takes as input the master public key mpk, the master secret key msk and a keyword w in some associated keyword space W. It returns trapdoor Td_w.
- **Test** takes as input the master public key mpk, an encrypted tag Tag_w and a trapdoor Td_w. It returns a test result, TRUE (=1) or FALSE (=0). **Test** returns TRUE if both hidden keywords in the encrypted tag and the trapdoor are the same, and otherwise returns FALSE.
- **GenIndexKey** takes as input the master public key mpk, the master secret key msk and an index depth ℓ. It returns an index extraction key $IKey_\ell$.
- **ExtractTagIndex** takes as input the master public key mpk, an encrypted tag Tag_w and an index extraction key $IKey_\ell$. It returns either an index value $\{0,1\}^*$ or the distinguished symbol $\perp$.
- **ExtractTrapdoorIndex** takes as input the master public key mpk, a trapdoor Td_w and an index extraction key $IKey_\ell$. It returns either an index value $\{0,1\}^*$ or the distinguished symbol $\perp$.

Let us consider our correctness conditions of the first step scheme. The **Test** algorithm returns TRUE if both a tag and a trapdoor are generated from the same keyword.

Further, the index values extracted from a tag and a trapdoor must be the same when these are generated from the same keyword, and the depth ℓ index values extracted from a tag and a trapdoor must be the same when the ℓ-th elements of output of **Index** evaluated by the keywords w and w' are the same, where the tag and trapdoor are generated from w and w', respectively. Therefore, the following correctness conditions must be satisfied.

Definition 7 (Correctness conditions)**.** For any security parameter 1^λ , any $\ell \le L_{index}$, and any different keywords w and w' which have the same index value on ℓ-th depth of index, it holds that:

(1) **Test**$(mpk, Tag_w, Td_w) = TRUE$

(2) **ExtractTagIndex**$(mpk, Tag_w, IKey_\ell)$
= **ExtractTrapdoorIndex**$(mpk, Td_w, IKey_\ell)$

(3) **ExtractTagIndex**$(mpk, Tag_w, IKey_\ell)$
= **ExtractTagIndex**$(mpk, Tag_{w'}, IKey_\ell)$

where **KeyGen**$(1^\lambda) \to (mpk, msk)$, **GenTag**$(mpk, w) \to Tag_w$, **GenTag**$(mpk, w') \to Tag_{w'}$, **GenTrapdoor**$(mpk, msk, w) \to Td_w$, **GenIndexKey**$(mpk, msk, \ell) \to IKey_\ell$.

Further, it is necessary that the **Test** algorithm returns FALSE if both a tag and a trapdoor are generated from different keywords, and that the depth ℓ index values extracted from a tag and a trapdoor are different when the ℓ-th elements of outputs of **Index** evaluated by the keywords w and w' are different, where the tag and the trapdoor are generated from w and w', respectively. Therefore, the following consistency conditions must be satisfied.

Definition 8 (Consistency conditions)**.** For any different keywords w and w' which have different index values on ℓ-th depth of index, it holds that:

(1) **Test**$(mpk, Tag_w, Td_{w'}) = FALSE$ with overwhelming probability

(2) **ExtractTagIndex**$(mpk, Tag_w, IKey_\ell)$
$\neq$ **ExtractTrapdoorIndex**$(mpk, Td_{w'}, IKey_\ell)$

(3) **ExtractTagIndex**$(mpk, Tag_w, IKey_\ell)$
$\neq$ **ExtractTagIndex**$(mpk, Tag_{w'}, IKey_\ell)$

where **KeyGen**$(1^\lambda) \to (mpk, msk)$, **GenTag**$(mpk, w) \to Tag_w$, **GenTag**$(mpk, w') \to Tag_{w'}$, **GenTrapdoor**$(mpk, msk, w') \to Td_{w'}$, **GenIndexKey**$(mpk, msk, \ell) \to IKey_\ell$.

4.3 Security Definition

In the previous section, we constructed index values by using the concept of k-anonymity for performance improvements while security is also ensured. In this subsection, we discuss the security of the first step scheme. In the scheme, while index values are initially kept secret, the index values will be disclosed partially for improving search performance. This situation is formulated as follows: Even if an adversary knows plain keyword candidates (hereafter, keyword group) for each encrypted tag, it is impossible to specify which keywords chosen from the corresponding keyword group are encrypted. We define security as hidden keywords

© 2020 Information Processing Society of Japan

being indistinguishable within the same keyword group identified by the index values. First, the security definition of the searchable encryption for a single-user setting is shown as follows:

Definition 9. Let L_{index} be a constant. A public-key searchable encryption with index generation scheme up to L_{index} depth is adaptively secure against chosen keyword attacks if for all probabilistic polynomial-time adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the following experiment is negligible in the security parameter.

(1) **Setup** is run to generate a pair of mpk and msk, and mpk is given to $\mathcal{A}$.

(2) $\mathcal{A}$ may adaptively makes a polynomial number of queries of the following type:
- Trapdoor query: $\mathcal{A}$ asks the challenger to create a trapdoor for a keyword $w \in W$. The challenger creates a trapdoor Td_w for w and gives it to $\mathcal{A}$.
- Index key query: $\mathcal{A}$ asks the challenger to create an index extraction key for $\ell \leq L_{index}$ depth. The challenger creates an index extraction key $IKey_\ell$ for depth ℓ and gives it to $\mathcal{A}$.

(3) $\mathcal{A}$ chooses two challenge keywords $w_0^*, w_1^* \in W$, and sends them to the challenger. The only restriction is that the $\mathcal{A}$ cannot choose keywords which can be distinguished by any trapdoors or index extraction keys which $\mathcal{A}$ has already queried.

(4) The challenger chooses a random bit b and creates a challenge message $Tag_{w_b^*}$

(5) $\mathcal{A}$ may adaptively make a polynomial number of trapdoor queries or index key query, similar to phase (2). The only restriction is that the $\mathcal{A}$ cannot query trapdoors or index extraction keys which can identify the challenge message $Tag_{w_b^*}$.

(6) $\mathcal{A}$ outputs a bit b', and succeeds if $b' = b$.

The advantage of an adversary in this game is defined to be $\mathbf{Adv}_{\mathcal{A}}^{SE}(\lambda) = |\Pr[b' = b] - 1/2|$.

We have to set several restrictions on queries in our adaptive game, similarly to a lot of other adaptive games in public-key encryption, in order to avoid trivial attacks. Also, we put a restriction on the index depth L_{index} in our adaptive game. This detail is given as follows.

The number of the equivalence classes $S_{Idx} = \{w \in W | \mathbf{Index}(w) = Idx\}(Idx \in Image(\mathbf{Index}))$ might be increased exponentially as L_{index} is increased linearly. In our adaptive game, a polynomial time adversary has to choose two challenge keywords w and w' such that $\mathbf{Index}(w) = \mathbf{Index}(w')$ (i.e., w and w' belong to the same equivalence class). If L_{index} (more precisely, ℓ) is large, it might be hard for the adversary to choose the keywords. Therefore, we additionally place a restriction that L_{index} is a constant in the security parameter in our adaptive game.

Fortunately, this restriction is useful for various cases. Let the keyword space for example be a set of Japanese family names, and the index function be a part of SHA256 hash values. It is well known that the cardinality of the keyword space is nearly 100,000. In this case, if we set $L_{index} = 5$ (or namely, we use the first 5 bits of hash values), then it will be about 2^5 times faster than an ordinary search process, while about 3000-anonymity is ensured. If $L_{index} = 10$, then it will be about 2^{10} times faster while about 100-anonymity is ensured.

4.4 Construction

Details of our algorithms in the first step scheme are described in **Fig. 1**. The **GenTag** function generates an encrypted tag, which contains both a searchable encryption part, Tag_{SE}, (SE part for short) and an encrypted index, $\{EncIdxTag_i\}_{L_{index}}$. The SE part is generated in the same manner explained in Section 3.4. The encrypted index is generated by encrypting each index value which is output by the **Index** function. Similarly, the **GenTrapdoor** function generates a trapdoor which contains an SE part and an encrypted index similar to that of the encrypted tag. The **Test** function compares the SE part of an encrypted tag with the SE part of a trapdoor to test if the hidden keywords match/mismatch. The **GenIndexKey** function generates an index extraction key according to the index depth to be disclosed. Using the generated index extraction key, the index value can be extracted from an encrypted tag and a trapdoor by the **ExtractTagIndex** function and the **ExtractTrapdoorIndex** function, respectively. By using this as index, it is possible to extract the candidates of encrypted tags for which the **Test** function should be executed at the search time.

Though the search performance is improved, there are the following tradeoffs:

(1) The size of the tags and the trapdoors grows linearly according to L_{index}. That is, our proposed scheme requires a larger storage size than that of ordinary searchable encryption scheme.

(2) After an index extraction key $IKey_\ell$ is disclosed, running our tag index extraction function **ExtractTagIndex** for all the tags stored in the server is required to obtain the index values of depth ℓ, and the server stores them into database. The complexity of this process is roughly $O(TAG \times Dec_{HIPE})$, where TAG is the number of the stored tags and Dec_{HIPE} is the decryption complexity of an HIPE scheme. Also, the server has to perform the above index extraction for all the tags after an index extraction key is further disclosed. When a tag is stored after the above index extraction, the server also has to perform the same process for the tag.

(3) After the index extraction process (2), the server has to perform our trapdoor index extraction function **ExtractTrapdoorIndex** for a trapdoor by using index extraction keys already disclosed in the keyword search phase. The complexity of this process is roughly $O(\ell \times Dec_{HIPE})$, where ℓ is the number of the index extraction keys disclosed already.

Therefore, it is necessary to estimate the maximum size of the data and determine the index depth L_{index} appropriately. We discuss the **Index** function more detail in Section 6.3.

The **EncodeAttribute** function vectorizes a 2 dimensional attribute $\{attr_1, attr_2\}$ into a 4 dimensional vector $(\sigma_1 attr_1, \sigma_1, \sigma_2 attr_2, \sigma_2)$, and the **EncodePredicate** function vectorizes a 2 dimensional predicate $\{pred_1, pred_2\}$ into a 4 dimensional vector $(\rho_1, -\rho_1 pred_1, \rho_2, -\rho_2 pred_2)$. The decryption of HIPE succeeds only if the inner product value of both vectors is 0, or namely, $\sigma_1\rho_1(attr_1 - pred_1) + \sigma_2\rho_2(attr_2 - pred_2) = 0$ over $\mathbb{F}_q$ is satisfied. This equation holds with overwhelming probability, where $attr_1 = pred_1$ and $attr_2 = pred_2$, which mean

© 2020 Information Processing Society of Japan

Algorithm:

Setup$(1^\lambda, L_{index}) \to (mpk, msk)$

$\overrightarrow{\mu}_{HIPE} := (n = 2d, d = 2; \mu_1 = 2, \mu_2 = 4)$, $(mpk_{HIPE}, msk_{HIPE}) :=$ **Setup**$_{HIPE}(1^\lambda, \overrightarrow{\mu}_{HIPE})$

choose suitable index function, **Index**$(w \in W, \ell \in \{1, \ldots, L_{index}\}) \to \{0,1\}^*$, which returns an ℓ-th level of index value.

$mpk := (mpk_{HIPE}, L_{index},$ **Index**$(,))$, $msk := (msk_{HIPE})$

return (mpk, msk)

GenTag$(mpk, w) \to Tag_w$

$r \xleftarrow{U} M_{HIPE}$, $c_{SE} :=$ **Enc**$_{HIPE}(mpk_{HIPE},$ **EncodeAttribute**$(mpk_{HIPE}, \{\text{'SE'}, w\}), r)$, $Tag_{SE} := (r, c_{SE})$

For all $i \in \{1, \ldots, L_{index}\}$, $EncIdxTag_i :=$ **Enc**$_{HIPE}(mpk_{HIPE},$ **EncodeAttribute**$(mpk_{HIPE}, \{\text{'IdxTag'}, i\}),$ **Index**$(w, i))$

return $Tag_w := (Tag_{SE}, \{EncIdxTag_i\}_{L_{index}})$

GenTrapdoor$(mpk, msk, w) \to Td_w$

$Td_{SE} :=$ **KeyGen**$(mpk_{HIPE}, msk,$ **EncodePredicate**$(mpk_{HIPE}, \{\text{'SE'}, w\}))$

For all $i \in \{1, \ldots, L_{index}\}$, $EncIdxTd_i :=$ **Enc**$_{HIPE}(mpk_{HIPE},$ **EncodeAttribute**$(mpk_{HIPE}, \{\text{'IdxTd'}, i\}),$ **Index**$(w, i))$

return $Td_w := (Td_{SE}, \{EncIdxTd_i\}_{L_{index}})$

Test$(mpk, Tag_w, Td_w) \to \{TRUE, FALSE\}$

$m' :=$ **Dec**$(mpk_{HIPE}, Td_{SE}, C_{SE})$

If $m' = r$, $result$ = TRUE, otherwise $result$ = FALSE

return $result$

GenIndexKey$(mpk, msk, \ell) \to IKey_\ell$

For all $i \in \{1, \ldots, \ell\}$, $IKTag_i :=$ **KeyGen**$(mpk_{HIPE}, msk_{HIPE},$ **EncodePredicate**$(mpk_{HIPE}, \{\text{'IdxTag'}, i\}))$,

$IKTd_i :=$ **KeyGen**$(mpk_{HIPE}, msk_{HIPE},$ **EncodePredicate**$(mpk_{HIPE}, \{\text{'IdxTd'}, i\}))$

return $IKey_\ell := (\{IKTag_i\}_\ell, \{IKTd_i\}_\ell)$

ExtractTagIndex$(mpk, Tag_w, IKey_\ell) \to \{0,1\}^*$

For all $i \in \{1, \ldots, \ell\}$, $IdxTag_i :=$ **Dec**$(mpk_{HIPE}, IKTag_i, EncIdxTag_i)$

return $result := IdxTag_1 | \ldots | IdxTag_\ell$

ExtractTrapdoorIndex$(mpk, Td_w, IKey_\ell) \to \{0,1\}^*$

For all $i \in \{1, \ldots, \ell\}$, $IdxTd_i :=$ **Dec**$(mpk_{HIPE}, IKTd_i, EncIdxTd_i)$

return $result := IdxTd_1 | \ldots | IdxTd_\ell$

Functions:

EncodeAttribute$(pk, attr) \to (\overrightarrow{x}_1, \overrightarrow{x}_2)$

For all $i \in \{1,2\}$, $\sigma_i \xleftarrow{U} \mathbb{F}_q \setminus \{0\}$, $\overrightarrow{x}_i = \sigma_i(attr_i, 1)$

return $(\overrightarrow{x}_1, \overrightarrow{x}_2)$

EncodePredicate$(pk, pred) \to (\overrightarrow{v}_1, \overrightarrow{v}_2)$

For all $i \in \{1,2\}$, $\rho_i \xleftarrow{U} \mathbb{F}_q \setminus \{0\}$, $\overrightarrow{v}_i = \rho_i(1, -pred_i)$

return $(\overrightarrow{v}_1, \overrightarrow{v}_2)$

Fig. 1 Algorithm of the first step scheme (single-user setting).

that both attributes and predicates are the same.

Theorem 1. The first step scheme satisfies Definition 7.

Proof. We explain the correctness condition 1 is satisfied. The **Test** function decrypts the random number r encrypted with the attribute $\{'SE', w\}$ by HIPE in the **GenTag** function, by using the HIPE secret key generated with the predicate $\{'SE', w\}$ in the **GenTrapdoor**, and therefore the random number r appears as its decryption result. This function compares the decryption result with the un-encrypted random number r contained in the encrypted tag, then returns $TRUE(= 1)$. Therefore, the correctness condition 1 is satisfied.

We explain the correctness condition 2. Both the **GenTag** function and the **GenTrapdoor** function generate the same index value for the same keyword w by the **Index** function, and encrypt them with the attributes $\{'IdxTag', i\}$ and $\{'IdxTd', i\}$, respectively. The **GenIndexKey** uses the predicates $\{'IdxTag', i\}$ and $\{'IdxTd', i\}$ to generate the secret key for extracting index values from encrypted tags and a trapdoor, respectively. The **ExtractTagIndex** and **ExtractTrapdoorIndex** functions respectively decrypt the encrypted index values contained in the encrypted tags and the trapdoor using the secret key, and return the same decryption result, that is, output of **Index** function. Therefore, the correctness condition 2 also holds.

We explain the correctness condition 3. Let the index values of the keywords w, w' be $Idx_w, Idx_{w'}$, respectively. From the assumption, both $Idx_w, Idx_{w'}$ are the same index value Idx at ℓ-th depth. As shown in the correctness condition 2, our construction simply encrypts the index value with the fixed attribute $\{'IdxTag', i\}$ and decrypts it with the fixed predicate $\{'IdxTag', i\}$. Thus, if HIPE satisfies the correctness condition of HIPE, the output of **ExtractTagIndex** is the same index value Idx for both keywords. In other words, the correctness condition 3 also holds. □

Theorem 2. The first step scheme satisfies Definition 8.

Proof. First, we show the consistency condition 1 is satisfied. We used the construction scheme proposed by Abdalla et al. [1] to construct a searchable encryption from HIPE. Since their construction satisfies computationally consistent, it follows that our first step scheme also satisfies computationally consistent. We show the outline of the proof. Let $\mathcal{U}$ be any polynomial time adversary attacks the computational consistency of our scheme, and consider the following polynomial time adversary $\mathcal{A}$ attacking the IND-CPA security of HIPE. The adversary $\mathcal{A}$ runs $\mathcal{U}$ to obtain keywords w, w'. It returns $\{'SE', 1, w, 1\}$ as the challenge attribute vector and randomly chosen challenge messages R_0, R_1

© 2020 Information Processing Society of Japan

Game0: Original game. The challenge message is created as follows:

$r \xleftarrow{U} M_{HIPE}$, $c_{SE} := \mathbf{Enc}_{HIPE}(mpk_{HIPE}, \mathbf{EncodeAttribute}(mpk_{HIPE}, \{\text{'SE'}, w_b\}), r)$, $Tag_{SE} := (r, c_{SE})$

For all $i \in \{1, \ldots, L_{index}\}$, $EncIdxTag_i := \mathbf{Enc}_{HIPE}(mpk_{HIPE}, \mathbf{EncodeAttribute}(mpk_{HIPE}, \{\text{'IdxTag'}, i\}), \mathbf{Index}(w_b, i))$

Game1-1: Same as Game0 except L_{index}-th encrypted index:

$r \xleftarrow{U} M_{HIPE}$, $c_{SE} := \mathbf{Enc}_{HIPE}(mpk_{HIPE}, \mathbf{EncodeAttribute}(mpk_{HIPE}, \{\text{'SE'}, w_b\}), r)$, $Tag_{SE} := (r, c_{SE})$

For all $i \in \{1, \ldots, L_{index} - 1\}$, $EncIdxTag_i := \mathbf{Enc}_{HIPE}(mpk_{HIPE}, \mathbf{EncodeAttribute}(mpk_{HIPE}, \{\text{'IdxTag'}, i\}), \mathbf{Index}(w_b, i))$

$\boxed{R_{L_{index}} \xleftarrow{U} M_{HIPE}}$, $EncIdxTag_{L_{index}} := \mathbf{Enc}_{HIPE}(mpk_{HIPE}, \mathbf{EncodeAttribute}(mpk_{HIPE}, \{\text{'IdxTag'}, L_{index}\}), \boxed{R_{L_{index}}})$

Game1-k ($k = 1, \cdots, L_{index} - \ell$): Same as Game1-(k-1) except ($L_{index} - (k-1)$)-th encrypted index:

$r \xleftarrow{U} M_{HIPE}$, $c_{SE} := \mathbf{Enc}_{HIPE}(mpk_{HIPE}, \mathbf{EncodeAttribute}(mpk_{HIPE}, \{\text{'SE'}, w_b\}), r)$, $Tag_{SE} := (r, c_{SE})$

For all $i \in \{1, \ldots, L_{index} - k\}$, $EncIdxTag_i := \mathbf{Enc}_{HIPE}(mpk_{HIPE}, \mathbf{EncodeAttribute}(mpk_{HIPE}, \{\text{'IdxTag'}, i\}), \mathbf{Index}(w_b, i))$

For all $\boxed{i \in \{L_{index} - (k-1), \ldots, L_{index}\}}$, $\boxed{R_i \xleftarrow{U} M_{HIPE}}$, $EncIdxTag_i := \mathbf{Enc}_{HIPE}(mpk_{HIPE}, \mathbf{EncodeAttribute}(mpk_{HIPE}, \{\text{'IdxTag'}, i\}), \boxed{R_i})$

Game2: Same as Game1-($L_{index} - \ell$) except attribute vector for SE:

$r \xleftarrow{U} M_{HIPE}$, $\boxed{R_w \xleftarrow{U} \mathbb{F}_q}$, $c_{SE} := \mathbf{Enc}_{HIPE}(mpk_{HIPE}, \mathbf{EncodeAttribute}(mpk_{HIPE}, \{\text{'SE'}, \boxed{R_w}\}), r)$, $Tag_{SE} := (r, c_{SE})$

For all $i \in \{1, \ldots, L_{index} - \ell\}$, $EncIdxTag_i := \mathbf{Enc}_{HIPE}(mpk_{HIPE}, \mathbf{EncodeAttribute}(mpk_{HIPE}, \{\text{'IdxTag'}, i\}), \mathbf{Index}(w_b, i))$

For all $i \in \{L_{index} - (\ell+1), L_{index}\}$, $R_i \xleftarrow{U} M_{HIPE}$, $EncIdxTag_i := \mathbf{Enc}_{HIPE}(mpk_{HIPE}, \mathbf{EncodeAttribute}(mpk_{HIPE}, \{\text{'IdxTag'}, i\}), R_i)$

Fig. 2 Sequence of games for the first step scheme.

from M_{HIPE}. In the guess stage, given the challenge ciphertext C_b (that encrypts the R_b under identity $\{'SE', 1, w, 1\}$), $\mathcal{A}$ uses its key-derivation oracle to obtain a private key $sk_{\{1,-'SE',1,-w'\}}$ for $\{1, -'SE', 1, -w'\}$. If $\mathbf{Dec}_{HIPE}(sk_{\{1,-'SE',1,-w'\}}, C_b) = R_1$, then it returns 1, otherwise 0. In other words, the adversary $\mathcal{A}$ wins the game. This is a contradiction since HIPE used in our scheme satisfies IND-CPA security, and then our scheme satisfies the consistency condition 1.

We explain the consistency condition 2. Let the index values of the keywords w, w' be $Idx_w, Idx_{w'}$, respectively. From the assumption, both keywords w, w' have different index values at ℓ-th depth, then $Idx_w \neq Idx_{w'}$. As shown in the correctness condition 2, our construction simply encrypts the index value with the fixed attribute $\{'IdxTag', i\}$ and decrypts it with the fixed predicate $\{'IdxTag', i\}$. Thus, if HIPE satisfies correctness condition, the outputs of **ExtractTagIndex** and **ExtractTrapdoorIndex** are different index values $Idx_w, Idx_{w'}$, respectively. Namely, the consistency condition 2 also holds.

We explain the consistency condition 3. Similar to the consistency condition 2, If both keywords w, w' have different index values at ℓ-th depth, then the outputs of **ExtractTagIndex** are also different. In other words, the consistency condition 3 also holds. □

4.5 Security Proof

In this section, we show that our first step scheme described in Fig. 1 satisfies Definition 9.

Theorem 3. Let L_{index} be a constant. The proposed searchable encryption scheme with index generation is adaptively secure against chosen keyword attacks under the attribute-hiding and payload-hiding HIPE scheme. For any adversary $\mathcal{A}$, the advantages of $\mathcal{A}$ is negligible and evaluated as follows:

$$\mathbf{Adv}^{SE}_{\mathcal{A}}(\lambda) \leq L_{index}\mathbf{Adv}^{HIPE,AH}_{\mathcal{B}}(\lambda) + \mathbf{Adv}^{HIPE,PE}_{\mathcal{B}}(\lambda)$$

Proof. To prove the theorem, we consider the sequence of games described in **Fig. 2**. In these games, ℓ is the number of index queries which are adaptively called by the adversary $\mathcal{A}$. Let $\mathbf{Adv}^{(0)}_{\mathcal{A}}(\lambda)$, $\mathbf{Adv}^{(1-k)}_{\mathcal{A}}(\lambda)$ and $\mathbf{Adv}^{(2)}_{\mathcal{A}}(\lambda)$ be $\mathbf{Adv}^{SE}_{\mathcal{A}}(\lambda)$ in Game0, Game1 $-k$ and Game2, respectively. It is clear that $\mathbf{Adv}^{(2)}_{\mathcal{A}}(\lambda) = 0$ because there is no information from which an adversary can distinguish between two challenge keywords.

First, we discuss the gaps between the advantages of Game0 and Game1 $-$ 1. The difference of the games is the index value in L_{index}-th encrypted index tag ($EncIdxTag_{L_{index}}$). The distribution of $EncIdxTag_{L_{index}}$ in Game0 and Game1 $-$ 1 cannot be distinguished by the attacker because HIPE has payload-hiding property. Therefore, the gaps can be evaluated as follows:

$$|\mathbf{Adv}^{(0)}_{\mathcal{A}}(\lambda) - \mathbf{Adv}^{(1-1)}_{\mathcal{A}}(\lambda)| \leq \mathbf{Adv}^{HIPE,PE}_{\mathcal{B}}(\lambda)$$

Similarly, the gaps between the advantages of Game1 $-$ k and Game1 $-$ $(k+1)$ can be evaluated as follows:

$$|\mathbf{Adv}^{(1-k)}_{\mathcal{A}}(\lambda) - \mathbf{Adv}^{(1-(k+1))}_{\mathcal{A}}(\lambda)| \leq \mathbf{Adv}^{HIPE,PE}_{\mathcal{B}}(\lambda)$$

The same as in the above discussion, we discuss the gaps between the advantages of Game1 $-$ $(L_{index} - \ell)$ and Game2. The difference in the games is the attributes from which c_{SE} is created. Because HIPE has an attribute-hiding property, the gaps can be evaluated as follows:

$$|\mathbf{Adv}^{(1-(L_{index}-\ell))}_{\mathcal{A}}(\lambda) - \mathbf{Adv}^{(2)}_{\mathcal{A}}(\lambda)| \leq \mathbf{Adv}^{HIPE,AH}_{\mathcal{B}}(\lambda)$$

Finally, we obtain

$$\begin{aligned}
\mathbf{Adv}^{SE}_{\mathcal{A}}(\lambda) \leq\ & |\mathbf{Adv}^{(0)}_{\mathcal{A}}(\lambda) - \mathbf{Adv}^{(1-1)}_{\mathcal{A}}(\lambda)| \\
& + |\mathbf{Adv}^{(1-1)}_{\mathcal{A}}(\lambda) - \mathbf{Adv}^{(1-2)}_{\mathcal{A}}(\lambda)| \\
& \cdots \\
& + |\mathbf{Adv}^{(1-(L_{index}-(\ell-1)))}_{\mathcal{A}}(\lambda) - \mathbf{Adv}^{(1-(L_{index}-\ell))}_{\mathcal{A}}(\lambda)| \\
& + |\mathbf{Adv}^{(1-(L_{index}-\ell))}_{\mathcal{A}}(\lambda) - \mathbf{Adv}^{(2)}_{\mathcal{A}}(\lambda)| \\
& + |\mathbf{Adv}^{(2)}_{\mathcal{A}}(\lambda)| \\
\leq\ & L_{index}\mathbf{Adv}^{HIPE,AH}_{\mathcal{B}}(\lambda) + \mathbf{Adv}^{HIPE,PE}_{\mathcal{B}}(\lambda).
\end{aligned}$$

This completes the proof of the theorem. □

5. Multi-user Support

For the second step, we extend the first step scheme to satisfy both Requirements A and B.

© 2020 Information Processing Society of Japan

5.1 Idea for multi-user support

The idea for extending the first step scheme to multi-user setting is applying a hierarchical identity with a wildcard into our scheme. We need to consider how to convert the hierarchical identity with a wildcard into vector which HIPE can handle. Our idea is to convert a hierarchical identity $ID := \{ID_1, \ldots, ID_{L_{ID}}\}$ into a vector $(ID_1, 1, \ldots, ID_{L_{ID}}, 1)$ at tag generation, $(1, -ID_1, \ldots, 1, -ID_{L_{ID}})$ at trapdoor generation. The condition that the inner-product of both vectors becomes zero and access control based on a hierarchical identity is then achieved. In addition, to realize a wild-card as identity, we convert the wildcard into the zero-vector, $(0, 0)$, which gives an inner product 0 for any vector, thus matching any ID. For example, if a hierarchical ID, $('Adept.', *)$, can be matched with both hierarchical IDs,$('Adept.','UserA')$ and $('Adept.','UserB')$, then any users belonging to 'A Dept.' can search the encrypted tags or in other words multi-user support can be achieved.

5.2 Syntax

We extend the syntax which can support hierarchical identity with wildcard.

Definition 10. Let user identity be a sequence $\{ID_1, \ldots, ID_{L_{ID}}\} \in (\mathbb{F}_q \setminus \{0, q-1\})^{L_{ID}}$. A public-key searchable encryption with index generation scheme for multi-user setting consists of probabilistic polynomial-time algorithms **Setup**, **GenUserKey**, **GenTag**, **GenTrapdoor**, **Test**, **GenIndexKey**, **ExtractTagIndex** and **ExtractTrapdoorIndex**. These are given as follows:

- **Setup** takes as input a security parameter 1^λ, a depth of identity L_{ID}, and a depth of index L_{index}. It returns a master public key mpk and a master secret key msk.
- **GenUserKey** takes as input the master public key mpk, the master secret key msk, and a user identity ID_U. It returns a corresponding user secret key sk_U.
- **GenTag** takes as input the master public key mpk, a recipient identity ID_R and a keyword w in some associated keyword space W. It returns an encrypted tag Tag_w.
- **GenTrapdoor** takes as input the master public key mpk, a user secret key sk_U and a keyword w in some associated keyword space W. It returns a trapdoor Td_w.
- **Test** takes as input the master public key mpk, an encrypted tag Tag_w and a trapdoor Td_w. It returns a test result, TRUE (=1) or FALSE (=0). **Test** returns TRUE if both hidden keywords in the encrypted tag and the trapdoor are the same, and otherwise returns FALSE.
- **GenIndexKey** takes as input the master public key mpk, the master secret key msk, a recipient identity ID_R and an index depth ℓ. It returns an index extraction key $IKey_{R,\ell}$.
- **ExtractTagIndex** takes as input the master public key mpk, an encrypted tag Tag_w and an index extraction key $IKey_{R,\ell}$. It returns either an index value $\{0,1\}^*$ or the distinguished symbol $\perp$.
- **ExtractTrapdoorIndex** takes as input the master public key mpk, a trapdoor Td_w and an index extraction key $IKey_{R,\ell}$. It returns either an index value $\{0,1\}^*$ or the distinguished symbol $\perp$.

The definition is almost the same as Definition 6 except a hierarchical identity for a group and users are specified in the argument of each algorithm. In addition, we also define the **GenUserKey** function that issues a secret key for each user. Correctness conditions and consistency conditions are similar to those of the single-user setting except that both user identities and recipient identities need to be considered.

5.3 Security Definition

It is easy to extend our security definition from a single-user setting to a multi-user setting as follows:

Definition 11. Let L_{index} be a constant. A public-key searchable encryption with index generation scheme up to L_{index} depth is adaptively secure against chosen keyword attacks if for all probabilistic polynomial-time adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the following experiment is negligible in the security parameter.

(1) **Setup** is run to generate a pair of mpk and msk, and mpk is given to $\mathcal{A}$.

(2) $\mathcal{A}$ may adaptively makes a polynomial number of queries of the following type:
- User key query: $\mathcal{A}$ asks the challenger to create a user secret key for a user identity ID_U. The challenger creates a user secret key sk_U and gives it to $\mathcal{A}$.
- Trapdoor query: $\mathcal{A}$ asks the challenger to create a trapdoor for a keyword $w \in W$ and a user identity ID_U. The challenger creates a trapdoor Td_w for w and gives it to $\mathcal{A}$.
- Index key query: $\mathcal{A}$ asks the challenger to create an index extraction key of depth ℓ for a recipient identity ID_R. The challenger creates an index extraction key $IKey_{R,\ell}$ of depth ℓ and gives it to $\mathcal{A}$.

(3) $\mathcal{A}$ chooses two challenge keywords $w_0^*, w_1^* \in W$, a recipient identity ID_R^*, and sends them to the challenger. The only restriction is that the $\mathcal{A}$ cannot choose keywords and a recipient identity which can be distinguished by any user keys, trapdoors, or index extraction keys which $\mathcal{A}$ has already queried.

(4) The challenger choose a random bit b and create a challenge message $Tag_{w_b^*}$

(5) $\mathcal{A}$ may adaptively make a polynomial number of user key queries, trapdoor queries or index key queries, similar to phase (2). The only restriction is that the $\mathcal{A}$ cannot query user keys, trapdoors or index extraction keys which can distinguish the challenge message $Tag_{w_b^*}$.

(6) $\mathcal{A}$ outputs a bit b', and succeeds if $b' = $ b.

The advantage of an adversary in this game is defined to be $\mathbf{Adv}_{\mathcal{A}}^{SE}(\lambda) = |\Pr[b' = b] - 1/2|$.

Since the scheme supports the hierarchical identity, the adversary needs to specify the challenge recipient identity ID_R^* with challenge keywords, w_0^*, w_1^*. In the index key query, the adversary can specify for which recipients the index extraction key is used. In addition, the adversary can retrieve any user secret key via a user key query. However, the adversary cannot retrieve an index extraction key or a user secret key which distinguishes the challenge keyword.

5.4 Construction

The scheme for multi-user setting is shown in **Fig. 3**. The

© 2020 Information Processing Society of Japan

Algorithm:

Setup$(1^\lambda, L_{ID}, L_{index}) \to (mpk, msk)$

$\overrightarrow{\mu}_{HIPE} := (n = 2d, d = L_{ID} + 2; \mu_1 = 2, \ldots, \mu_d = n)$, $(mpk_{HIPE}, msk_{HIPE}) := \mathbf{Setup}_{HIPE}(1^\lambda, \overrightarrow{\mu}_{HIPE})$

choose suitable index generation function, **Index**$(w \in W, \ell \in \{1, \ldots, L_{index}\}) \to \{0,1\}^*$, which returns an ℓ-th index value.

$mpk := (mpk_{HIPE}, L_{ID}, L_{index}, \mathbf{Index}(,))$, $msk := (msk_{HIPE})$

return (mpk, msk)

GenUserKey$(mpk, msk, ID_U) \to sk_U$

$sk_{SE} := \mathbf{KeyGen}_{HIPE}(mpk_{HIPE}, msk_{HIPE}, \mathbf{EncodePredicate}(mpk_{HIPE}, 1, L_{ID} + 1, \text{'SE'}|ID_U))$

return $sk_U := (sk_{SE}, ID_U)$

GenTag$(mpk, ID_R, w) \to Tag_w$

$r \xleftarrow{U} M_{HIPE}$, $c_{SE} := \mathbf{Enc}_{HIPE}(mpk_{HIPE}, \mathbf{EncodeAttribute}(mpk_{HIPE}, 1, L_{ID} + 2, \text{'SE'}|ID_R|w), r)$, $Tag_{SE} := (r, c_{SE})$

For all $i \in \{1, \ldots, L_{index}\}$, $EncIdxTag_i := \mathbf{Enc}_{HIPE}(mpk_{HIPE}, \mathbf{EncodeAttribute}(mpk_{HIPE}, 1, L_{ID} + 2, \{\text{'IdxTag'}, ID_R, i\}), \mathbf{Index}(w, i))$

return $Tag_w := (Tag_{SE}, \{EncIdxTag_i\}_{L_{index}})$

GenTrapdoor$(mpk, sk_U, w) \to Td_w$

$Td_{SE} := \mathbf{Delegate}(mpk_{HIPE}, sk_{SE}, \mathbf{EncodePredicate}(mpk_{HIPE}, L_{ID} + 2, 1, w))$

For all $i \in \{1, \ldots, L_{index}\}$, $EncIdxTd_i := \mathbf{Enc}_{HIPE}(mpk_{HIPE}, \mathbf{EncodeAttribute}(mpk_{HIPE}, 1, L_{ID} + 2, \{\text{'IdxTd'}, ID_U, i\}), \mathbf{Index}(w, i))$

return $Td_w := (Td_{SE}, \{EncIdxTd_i\}_{L_{index}})$

Test$(mpk, Tag_w, Td_w) \to \{TRUE, FALSE\}$

$m' := \mathbf{Dec}(mpk_{HIPE}, Td_{SE}, c_{SE})$

If $m' = r$, $result$ = TRUE, otherwise $result$ = FALSE

return $result$

GenIndexKey$(mpk, msk, ID_R, \ell) \to IKey_{R,\ell}$

For all $i \in \{1, \ldots, \ell\}$, $IKTag_{R,i} := \mathbf{KeyGen}(mpk_{HIPE}, msk_{HIPE}, \mathbf{EncodePredicate}(mpk_{HIPE}, 1, L_{ID} + 2, \{\text{'IdxTag'}, ID_R, i\}))$,

$IKTd_{R,i} := \mathbf{KeyGen}(mpk_{HIPE}, msk_{HIPE}, \mathbf{EncodePredicate}(mpk_{HIPE}, 1, L_{ID} + 2, \{\text{'IdxTd'}, ID_R, i\}))$

return $IKey_{R,\ell} := (\{IKTag_{R,i}\}_\ell, \{IKTd_{R,i}\}_\ell)$

ExtractTagIndex$(mpk, Tag_w, IKey_{R,\ell}) \to \{0,1\}^*$

For all $i \in \{1, \ldots, \ell\}$, $IdxTag_{R,i} := \mathbf{Dec}(mpk_{HIPE}, IKTag_{R,i}, EncIdxTag_i)$

return $result := IdxTag_1|\ldots|IdxTag_\ell$

ExtractTrapdoorIndex$(mpk, Td_w, IKey_{R,\ell}) \to \{0,1\}^*$

For all $i \in \{1, \ldots, \ell\}$, $IdxTd_i := \mathbf{Dec}(mpk_{HIPE}, IKTd_{R,i}, EncIdxTd_i)$

return $result := IdxTd_1|\ldots|IdxTd_\ell$

Functions:

EncodeAttribute$(pk, pos, len, attr) \to (\overrightarrow{x}_{pos}, \ldots, \overrightarrow{x}_{pos+(len-1)})$

For all $i \in \{pos, \ldots, pos + len - 1\}$, $\sigma_i \xleftarrow{U} \mathbb{F}_q \setminus \{0\}$, $\overrightarrow{x}_i = (0,0)$ $(attr_i = *)$, $\overrightarrow{x}_i = \sigma_i(q-1, 1)$ $(attr_i = *$, for EncIdxTag), $\overrightarrow{x}_i = \sigma_i(attr_i, 1)$ (others)

return $(\overrightarrow{x}_{pos}, \ldots, \overrightarrow{x}_{pos+(len-1)})$

EncodePredicate$(pk, pos, len, attr) \to (\overrightarrow{v}_{pos}, \ldots, \overrightarrow{v}_{pos+(len-1)})$

For all $i \in \{pos, \ldots, pos+len-1\}$, $\rho_i \xleftarrow{U} \mathbb{F}_q \setminus \{0\}$, $\overrightarrow{v}_i = (0,0)$ $(pred_i = *)$, $\overrightarrow{v}_i = \sigma_i(1, -(q-1))$ $(pred_i = *$, for IKTag), $\overrightarrow{v}_i = \rho_i(1, -pred_i)$ (others)

return $(\overrightarrow{v}_{pos}, \ldots, \overrightarrow{v}_{pos+(len-1)})$

Fig. 3 Algorithm for the second step scheme (multi-user setting).

scheme is almost the same as Fig. 1 except hierarchical identities for groups and users are embedded into an encrypted tag, a trapdoor and an index extraction key, respectively. Each function uses the **EncodeAttribute** and the **EncodePredicate** functions to vectorize a hierarchical ID into an attribute or a predicate vector for HIPE so that the search is only possible if both hierarchical IDs are a match. Correctness conditions and consistency conditions can be proved in a similar way.

5.5 Security Proof

Similar to the first step scheme, the following theorem is satisfied.

Theorem 4. The second step searchable encryption scheme with index generation is adaptively secure against chosen keyword attacks under the attribute-hiding and payload-hiding HIPE scheme. For any adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ is negligible and evaluated as follows:

$$\mathbf{Adv}^{SE}_{\mathcal{A}}(\lambda) \le L_{index}\mathbf{Adv}^{HIPE,AH}_{\mathcal{B}}(\lambda) + \mathbf{Adv}^{HIPE,PE}_{\mathcal{B}}(\lambda)$$

We can prove the security of the scheme in a similar manner as that for the single-user setting so we omit the proof here.

6. Implementation and Evaluation

In this section, we describe the implementation especially for satisfying Requirement C and our proposal is then complete. In other words, all the Requirements A, B and C are satisfied. In addition, we describe the evaluation results of our proposed scheme.

6.1 Database Integration

We describe an integration method to apply the public-key searchable encryption with index generation scheme satisfying Requirements A and B, into a database so as to satisfy Requirement C.

[Database schema] Modifying database schema is required to support the searchable encryption. For our proposed scheme, it is necessary to store the encrypted tags and index values and also to store encrypted data itself. Therefore, as shown in the **Fig. 4**, four columns should be defined for each column of data.

© 2020 Information Processing Society of Japan

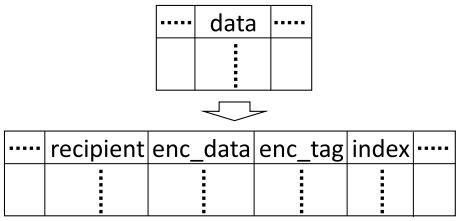

Fig. 4 Encrypted data table.

The encrypted data column is for encrypted data generated by an encryption scheme supporting multi-user functionality such as HIPE. The encrypted tag column is for encrypted tags generated by searchable encryption and is used at the search process. Both the encrypted data column and the encrypted tag column are a binary data type because the encrypted data and the tag are binary data. The index column is for (disclosed) index values and is a string type for comparing as a string. In the case of a string type, the index capability of the database can work or in other words the search performance is high. The recipient column is for recipient identities that indicate a group capable of search for the encrypted tag, and is a character string type. Our implementation uses public key encryption (PKE) and public-key searchable encryption (PEKS), independently. Some literature [11], [26], [30], [32] proposes both PKE and PEKS as secure integration schemes. The previous results on generic construction would be applicable to our schemes.

[Search procedure] First, the user ID of the client is received together with the trapdoor, and only the rows which can be searched by the client are selected, according to comparison of recipient IDs and the user ID. Next, before searching an encrypted tag column, the server extracts index values from the trapdoor using all index extraction keys which the server had already received from a key administrator, and chooses the longest index value from the extracted index values. Then the rows to be searched are further selected according to the comparison of the longest index value (trapdoor index) and its corresponding index column (tag index). It should be noted that since the depth of the index value disclosed for each recipient is different, forward matching is required to compare both the tag index and the trapdoor index. For example, if a trapdoor index value is '0011', not only the row having an index value '0011' but also the row having an index value '001' are selected as a candidate to be searched. After that, the searchable encryption process is executed by running **Test** function for all of the selected encrypted tags by the trapdoor. Then the server specifies all of the encrypted tags which may contain the search keyword and returns the specified encrypted data as the search result. The procedures make it possible to perform a database search using the searchable encryption while reducing the data to be processed using the index.

[Index extraction procedure] When an index extraction key is received from a key administrator, the recipient identity should be received together and appropriate rows selected. Then, the index value is extracted from each encrypted tags and is written into the index column of the corresponding row. In some cases, a search is requested while the index is being updated, but since the index column is compared by forward matching as described in the search procedure, search procedure and index extraction procedure can be parallelly processed.

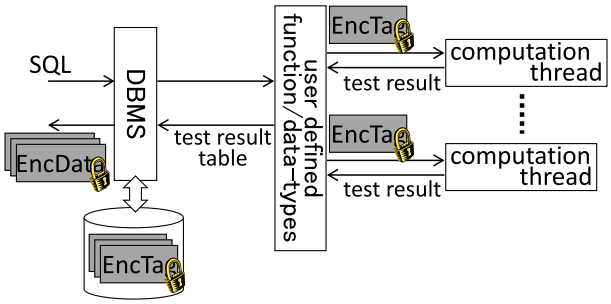

Fig. 5 DBMS integration.

[Implementation of user-defined functions] Modifying an application code to execute such processes is a difficult task. Therefore, some processes of the searchable encryption should be incorporated into the database. The block diagram of the implementation is shown in **Fig. 5**. Encrypted data and encrypted tags are stored on the database, and the time-consuming match process (processing the **Test** function) is performed in parallel by a plurality of threads or computing nodes. Further, in order to call the processing of the searchable encryption via SQL, the matching processing is implemented by a user-defined function or a user-defined data type. As a result, we can use SQL statement so as to call our searchable encryption without calling the **Test** function directly in the application code. At the same time, since parallel processing using a plurality of threads or computing nodes is automatically performed, it is possible to improve search performance. Similarly, we can develop a user-defined function which updates an index value from an encrypted tag. In the searchable encryption, master public parameter and index extraction keys are required for the matching process. Therefore, a management table for storing a master public parameter and index extraction keys must be prepared. In the user-defined function described above, the master public parameter and index extraction keys are referred to in the management table.

In this way, we obtain a public-key searchable encryption scheme satisfying all Requirements A, B, and C.

6.2 Key Management

To operate our schemes securely, key management is important. As an example, a project manager or a leader of an alliance plays a role as a key administrator to manage the master secret key securely, and distribute the master public key to all users. The key administrator also manages user attributes of their project or alliance and issues a user private key which contains attributes corresponding to the target user. The user can perform both encryption and search by using the master public key and his private key. The key administrator also issues index extraction keys. The key administrator periodically monitors the amount of stored data or receives a request from a user to decide whether performance improvement is required or not. If the key administrator determines performance improvement is required, he issues a new index extraction key for specific recipient ID_R and sends it to the database. Therefore, the key administrator has to manage both users and anonymity levels to ensure both security and perfor-

© 2020 Information Processing Society of Japan

mance.

6.3 Implementation of Index function

In order to use our proposed scheme securely, it is important to construct an **Index** function. The idea behind ensuring security is that if an adversary knows an index value and finds a candidate for the keyword from which the encrypted tag was created, then there should at least be a certain number of candidates. This is similar to the concept of k-anonymity. In general, to ensure the k-anonymity, it removes identifiers or quasi-identifiers so that an adversary cannot re-identify the individual. In other words, there area at least k-candidates of individuals, while the data remains practically useful. Our method divides the keyword space into multiple groups using the index values generated by the **Index** function. If the size of each groups is at least k or more, an adversary cannot re-identify the keyword from an encrypted tag and an index value. In other words, if there are at least k keyword candidates then security similar to k-anonymity is ensured.

One of the **Index** functions is an anonymization table. This is useful when the keyword space is fixed in advance or small, such as postal code, municipality name, etc. For example, if we want to use a 4-bit index for municipality name, then the municipality name space is divided into $2^4 = 16$ groups, each of which is the same size. The 4-bit group ID is assigned to each groups shared among users as an anonymization table. The **Index** function is constructed based on the anonymization table. Since there were 1897 city names in 2015 [16], even if a 1-bit index value is disclosed, the size of groups is at least 948, so 948-anonymity is satisfied. Also, even if a 2-bit index value is disclosed, the size of groups is at least 474, so 474-anonymity is satisfied. In this way, even when an adversary knows an encrypted tag and its index values, the adversary cannot re-identify which municipality name in the group is the original keyword, and there are at least k-candidate of municipality name.

The anonymization table is useful when the keyword space is predetermined, but it is not always the case. Another candidate of the **Index** function is to use a part of keyword hash. Since the output bits of hash functions are generally considered to be random, it is expected that they will roughly divide the keyword space into equal size. Therefore, we have experimented with the 4-bit index construction of municipality names by SHA-256 hash function. The result of our experiment is shown in **Table 2**.

The experiment shows that the size of the groups is approximately the same when divided into 8 groups using 3 bits, and the number of municipality names included in the groups is within 5.5% range of the average. However, when the municipality names are divided into 16 groups using 4 bits, it was found that the number of municipality names included in the groups varies up to 18.19% range of the average. The probability of 0 and 1 in each bits converges to $1/2$ when the number of keywords is large, but in this case, the probability does not converge to $1/2$ because the number of keywords in each group is only about 100. Our results shows that even if the number of group elements is 200 or 300, the group size is almost the same within several percent from average even if we use the hash values to divide the keyword space into groups. In general, in case that the keyword space can not be determined in advance, we can consider that the size of the keyword space is sufficiently large. For example, there are almost 250,000 words in Japanese dictionary, Koujien, and it is thought that there are about 100,000 kinds of Japanese family name [12]. In these case, it can be expected that we can use 5 to 10 bits index values.

Table 2 The number of elements in the group.

bit1	bit2	bit3	bit4
956 (bit1=0)	459 (bit2=0)	224 (bit3=0)	120 (bit4=0)
			104 (bit4=1)
		235 (bit3=1)	97 (bit4=0)
			138 (bit4=1)
	497 (bit2=1)	248 (bit3=0)	126 (bit4=0)
			122 (bit4=1)
		249 (bit3=1)	136 (bit4=0)
			113 (bit4=1)
941 (bit1=1)	470 (bit2=0)	230 (bit3=0)	101 (bit4=0)
			129 (bit4=1)
		240 (bit3=1)	131 (bit4=0)
			109 (bit4=1)
	471 (bit2=1)	237 (bit3=0)	105 (bit4=0)
			132 (bit4=1)
		234 (bit3=1)	120 (bit4=0)
			114 (bit4=1)

Table 3 The probability distribution of the group.

bit1	bit2	bit3	bit4
50.7% (bit1=0)	23.6% (bit2=0)	11.2% (bit3=0)	7.0% (bit4=0)
			4.2% (bit4=1)
		12.4% (bit3=1)	4.6% (bit4=0)
			7.9% (bit4=1)
	27.1% (bit2=1)	14.1% (bit3=0)	7.2% (bit4=0)
			6.9% (bit4=1)
		12.9% (bit3=1)	6.9% (bit4=0)
			6.1% (bit4=1)
49.3% (bit1=1)	23.4% (bit2=0)	11.0% (bit3=0)	4.8% (bit4=0)
			6.2% (bit4=1)
		12.4% (bit3=1)	6.3% (bit4=0)
			6.1% (bit4=1)
	25.9% (bit2=1)	13.1% (bit3=0)	6.0% (bit4=0)
			7.1% (bit4=1)
		12.8% (bit3=1)	6.3% (bit4=0)
			6.4% (bit4=1)

Although we show that the keyword space can be divided roughly equally, the distribution of the keywords might not be uniform. The population of each municipality varies greatly, from the largest population, 903,346 residents, to the smallest population, 0 residents. The average population is 67,377, the median is 29,638, and the standard deviation is 98,252. If we assume municipality names appear according to the distribution of the residents, the probability of each index value is shown in **Table 3**.

It can be seen from the table that the distribution of each index value becomes smoother, even though the population distribution is highly different. When a 3 bit index is divided into 8 groups, the probability of each index value becomes smooth to a range of 14.1% to 11.0%. Even in cases where there is a large difference in the distribution of original data as in the present case, the probability of index value becomes smoother, and it is considered difficult to identify a keyword from the index value distribution. Further, since the probability of the index value becomes smooth, it is understood that the search performance can be improved to approximately 2^n in accordance with the number of bits n of the disclosed index value.

© 2020 Information Processing Society of Japan

Table 4 Search performance of our proposed scheme.

the number of tags	the number of extracted indexes				
	none	1 bit	3 bit	5 bit	7 bit
5,000	11.71 [s]	5.31 [s]	1.33 [s]	0.73 [s]	0.27 [s]
10,000	23.29 [s]	15.12 [s]	2.74 [s]	0.67 [s]	0.36 [s]
50,000	114.70 [s]	63.27 [s]	12.41 [s]	2.31 [s]	0.70 [s]
100,000	232.73 [s]	127.06 [s]	30.16 [s]	4.25 [s]	0.73 [s]

6.4 Performance

We show the performance evaluation results from our proposed scheme. We chose the HIPE scheme proposed in Ref. [22], the pairing function over BN-curves [4], and its parameter and computing formulas proposed in Ref. [3]. We implemented our scheme in C programming language, and evaluated its performance on Amazon Linux AMI release 2018.03, T2.micro Xeon family (3.3 GHz maximum, 64 bit, single core), 1 GB of memory, and PostgreSQL 9.6.10 as a database. Three computation threads were used for executing **Test** function. The postal code was used as the data, and encryption tags were created by an algorithm shown in Fig. 3, using two dimensional IDs, affiliation name and user name, and stored on the database. The **Index** function is constructed using SHA-256 hash function. We measured SQL response time in our implementation 10 times, and the average value was used as the measurement result. The evaluation result is shown in **Table 4**.

As shown in the measurement result, the search time increases as the number of registered data items increases. For example, when 50,000 encrypted tags are stored, the search time requires 114.7 seconds, while it requires only 23.29 seconds for 10,000 encrypted tags, 11.71 seconds for 5,000 encrypted tags. By disclosing 1 bit of the index value, the search time can be shortened to 63.27 seconds in case of 50,000 encrypted tags, which is about half the time. Similarly, by increasing the index to 3 bits and 5 bits, it is found that the search time is also shortened to 12.41 seconds and 2.31 seconds. For example, if the response time could be limited to a few seconds, then the index will still be safe from disclosure even then there are 1,000 to 2,000 data items. When the number of data items increases to about 5,000, it might disclose a 1 bit index value in order to improve the performance. Similarly, when the number of data items increases up to 10,000 or 50,000 then changing to 3 bit or 5 bit index disclosure will improve the performance even more.

7. Conclusion

In this paper, we point out three important subjects on public-key searchable encryption for its practical use: (A) performance improvements, (B) sharing data for multi-users, (C) integration public-key searchable encryption into a database. With regard to (A), we propose a mechanism that achieves both high performance and strong security by applying the concept of k-anonymity. We then propose a scheme to achieve it using HIPE. Similarly, with regard to (B), we also show that a hierarchical ID with wildcard is suitable for multi-user searchable encryption and achieve a wildcard as a two-dimensional zero vector on HIPE. For (C), we propose a mechanism for integrating searchable encryption into a database by using user-defined functions or user-defined data types which can be implemented on major database.

In the future we will need to support more flexible searches such as forward matching searches or partial matching searches. Another topic for future work is considering key management such as who issues the secret key and how to provide it to users.

References

[1] Abdalla, M., Bellare, M., Catalano, D., Kiltz, E., Kohno, T., Lange, T., Malone-Lee, J., Neven, G., Paillier, P. and Shi, H.: Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions, *CRYPTO 2005*, Shoup, V. (Ed.), LNCS, Vol.3621, pp.205–222, Springer Heidelberg (2005).

[2] Agrawal, R., Kiernan, J., Srikant, R. and Xu, Y.: Order preserving encryption for numeric data, *Proc. 2004 ACM SIGMOD International Conference on Management of Data*, *SIGMOD '04*, pp.563–574, ACM (2004).

[3] Aranha, D.F., Karabina, K., Longa, P., Gebotys, C.H. and Hernandez, J.L.: Faster Explicit Formulas for Computing Pairings over Ordinary Curves, *Proc. Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Paterson, K.G. (Ed.), Lecture Notes in Computer Science, Vol.6632, pp.48–68, Springer (2011).

[4] Barreto, P.S.L.M. and Naehrig, M.: Pairing-Friendly Elliptic Curves of Prime Order, *Selected Areas in Cryptography*, Preneel, B. and Tavares, S. (Eds.), pp.319–331, Springer Berlin Heidelberg (2006).

[5] Bellare, M., Boldyreva, A. and O'Neill, A.: Deterministic and efficiently searchable encryption, *CRYPTO 2007*, Menezes, A. (Ed.), LNCS, Vol.4622, pp.535–552, Springer Heidelberg (2007).

[6] Bellare, M., Fischlin, M., O'Neill, A. and Ristenpart, T.: Deterministic Encryption: Definitional Equivalences and Constructions without Random Oracles, *Advances in Cryptology – CRYPTO 2008*, Wagner, D. (Ed.), pp.360–378, Springer Berlin Heidelberg (2008).

[7] Boldyreva, A., Chenette, N. and Adam, O.: Order-Preserving Encryption Revisited: Improved Security Analysis and Alternative Solutions, *Advances in Cryptology – CRYPTO 2011*, Rogaway, P. (Ed.), Lecture Notes in Computer Science, Vol.6841, pp.578–595, Springer (2011).

[8] Boldyreva, A., Chenette, N., Lee, Y. and Adam, O.: Order-Preserving Symmetric Encryption, *Advances in Cryptology - EUROCRYPT 2009*, Joux, A. (Ed.), Lecture Notes in Computer Science, Vol.5479, pp.224–241, Springer (2009).

[9] Boneh, D., Di Crescenzo, G., Ostrovsky, R. and Persiano, G.: Public key encryption with keyword search, *EUROCRYPT 2004*, Cachin, C. and Camenisch, J. (Eds.), LNCS, Vol.3027, pp.506–522, Springer Heidelberg (2004).

[10] Boneh, D. and Waters, B.: Conjunctive, Subset, and Range Queries on Encrypted Data, *TCC 2007*, Vadhan, S.P. (Ed.), LNCS, Vol.4392, pp.535–554, Springer Heidelberg (2007).

[11] Chen, Y., Zhang, J., Lin, D. and Zhang, Z.: Generic constructions of integrated PKE and PEKS, Vol.78, pp.493–526 (2016).

[12] Chida, S. and Mase, S.: Statistical Analysis of Japanese Surnames, *Journal of the Japan Statistical Society*, Vol.35, No.1, pp.55–70 (2005).

[13] Curtmola, R., Garay, J., Kamara, S. and Ostrovsky, R.: Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions, *13th ACM CCS*, pp.79–88, ACM Press (2006).

[14] Curtmola, R., Garay, J., Kamara, S. and Ostrovsky, R.: Searchable symmetric encryption: Improved definitions and efficient constructions, *Journal of Computer Security*, Vol.19, No.5, pp.895–934, IOS Press (2011).

[15] De Caro, A., Iovino, V. and Persiano, G.: Hidden vector encryption fully secure against unrestricted queries, Manuscript (2011).

[16] e Stat, available from ⟨https://www.e-stat.go.jp/regional-statistics/ssdsview/municipality⟩.

[17] Furukawa, J.: Request-Based Comparable Encryption, *Computer Security – ESORICS 2013*, Crampton, J., Jajodia, S. and Mayes, K. (Eds.), pp.129–146, Springer Berlin Heidelberg (2013).

[18] Hattori, M., Hirano, T., Ito, T., Matsuda, N., Mori, T., Sakai, Y. and Ohta, K.: Ciphertext-policy delegatable hidden vector encryption and its application to searchable encryption in multi-user setting, *IMACC 2011*, Chen, L. (Ed.), LNCS, Vol.7089, pp.190–209, Springer Heidelberg (2011).

[19] Horst, C., Kikuchi, R. and Xagawa, K.: Cryptanalysis of Comparable Encryption in SIGMOD '16, *Proc. 2017 ACM International Conference on Management of Data*, *SIGMOD '17*, pp.1069–1084, Association for Computing Machinery (online), DOI: 10.1145/3035918.3035948 (2017).

[20] Hwang, Y.H. and Lee, P.J.: Public key encryption with conjunctive keyword search and its extension to a multi-user system, *Pairing 2007*, Takagi, T., Okamoto, T., Okamoto, E. and Okamoto, T. (Eds.), LNCS,

© 2020 Information Processing Society of Japan

Vol.4575, pp.2–22, Springer Heidelberg (2007).
[21] Katz, J., Sahai, A. and Waters, B.: Predicate encryption supporting disjunctions, polynomial equations, and inner products, *EUROCRYPT 2008*, Smart, N. (Ed.), LNCS, Vol.4965, pp.146–162, Springer Heidelberg (2008).
[22] Lewko, A., Okamoto, T., Sahai, A., Takashima, K. and Waters, B.: Fully secure functional encryption: Attribte-based encryption and (hierarchical) inner product encryption, *EUROCRYPT 2010*, Gilbert, H. (Ed.), LNCS, Vol.6110, pp.62–91, Springer Heidelberg (2010).
[23] Matsuda, N., Ito, T., Shibata, H., Hattori, M. and Hirano, T.: Efficient Searchable Encryption and Its Application to Web Services, *DICOMO'2013*, Vol.2013, pp.2067–2074 (2013).
[24] Popa, R.A., Redfield, C.M.S., Zeldovich, N. and Balakrishnan, H.: CryptDB: Protecting Confidentiality with Encrypted Query Processing, *Proc. 23rd ACM Symposium on Operating Systems Principles*, *SOSP '11*, pp.85–100, ACM (online), DOI: 10.1145/2043556.2043566 (2011).
[25] Samarati, P. and Sweeney, L.: Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression (1998).
[26] Saraswat, V. and Sahu, R.A.: Short Integrated PKE+PEKS in Standard Model, *Security, Privacy, and Applied Cryptography Engineering*, Ali, S.S., Danger, J.-L. and Eisenbarth, T. (Eds.), pp.226–246, Springer International Publishing (2017).
[27] Shi, E., Bethencourt, J., Chan, T.-H.H., Song, D. and Perrig, A.: Multi-Dimensional Range Query over Encrypted Data, *IEEE SP 2007*, pp.350–364, IEEE Press (2007).
[28] Song, D.X., Wagner, D. and Perrig, A.: Practical techniques for searches on encrypted data, *2000 IEEE Symposium on Security and Privacy*, pp.44–55, IEEE Press (2000).
[29] Suzuki, T., Kojima, G., Fujiwara, K. and Yoshino, M.: Proposal and Implementation of Secure Architecture for Web Application, *IEICE Journal*, Vol.J101-D, No.1, pp.68–78 (2018).
[30] Suzuki, T., Emura, K. and Ohigashi, T.: A Generic Construction of Integrated Secure-Channel Free PEKS and PKE, *Information Security Practice and Experience*, Su, C. and Kikuchi, H. (Eds.), pp.69–86, Springer International Publishing (2018).
[31] van Liesdonk, P., Sedghi, S., Doumen, J., Hartel, P. and Jonker, W.: Computationally Efficient Searchable Symmetric Encryption, *Secure Data Management*, Jonker, W. and Petkovic, M. (Eds.), Lecture Notes in Computer Science, Vol.6358, pp.87–100, Springer Berlin/Heidelberg (2010).
[32] Zhang, R. and Imai, H.: Combining Public Key Encryption with Keyword Search and Public Key Encryption, *IEICE Journal*, Vol.E92-D, No.5, pp.888–896 (2009).

Nori Matsuda received his B.E. and M.E. degrees from Chuo University, Japan, in 1995 and 1997, respectively. He joined Mitsubishi Electric Corporation in 1997, and since then he has been engaged in the research and development of information security. He is currently a doctor course student at the Graduate School of Science and Technology of Shizuoka University and a researcher in the Mitsubishi Electric Corporation. He is a member of IEICE (Institute of Electronics, Information and Communication Engineers).

Takato Hirano received his B.A. degree from Osaka Kyoiku University in 2004, his M.S. degree from Tokyo Metropolitan University in 2006, and his D.S. degree from Tokyo Institute of Technology in 2010. He joined Mitsubishi Electric Corporation in 2010, and since then he has been engaged in the research and development of information security, especially in cryptosystems and privacy-preserving protocols. He is a member of IEICE and IPSJ (Information Processing Society of Japan).

Yutaka Kawai received his B.S. and M.S. degrees from the University of Electro-Communications and his Ph.D. degree from the University of Tokyo in 2007, 2009 and 2012, respectively. He is presently engaged in research on cryptography at Information Technology R&D center, Mitsubishi Electric Corporation.

Takashi Ito received his B.E. and M.E. degrees from The University of Tokyo, Japan, in 2000 and 2002, respectively. He joined Mitsubishi Electric Corporation in 2002, and since then he has been engaged in the research and development of information security. He is a member of IPSJ.

Mitsuhiro Hattori received his B.E. and M.E. degrees from Kyoto University, Japan, in 2003 and 2005, respectively. He joined Mitsubishi Electric Corporation in 2005, and since then he has been engaged in the research and development of information security. He is a member of IEICE and IPSJ.

Tadakazu Yamanaka received his B.E. and M.E. degrees from Kyoto Institute of Technology, Japan, in 2000 and 2002, respectively. He joined Mitsubishi Electric Corporation in 2002, and since then he has been engaged in the research and development of information security.

© 2020 Information Processing Society of Japan

Masakatsu Nishigaki received his Ph.D. in Engineering from Shizuoka University, Japan. He served as a Postdoctoral Research Fellow of the Japan Society for the Promotion of Science in 1995. Since 1996 he has been engaged in research at the Faculty of Informatics, Shizuoka University. He is now a Professor at the Graduate School of Science and Technology of Shizuoka University. His research interests are in wide variety of information security, especially in humanics security, media security, and network security. He served as Chief Examiner of IPSJ Special Interest Group on Computer Security from 2013 to 2014, Chair of IEICE Technical Committee on Biometrics from 2015 to 2016, and currently serving as JSSM (Japan Society of Security Management) Director since 2016, and IPSJ Director since 2019, respectively. He is IPSJ fellow.

© 2020 Information Processing Society of Japan