Privacy-Aware Cloud-based Input Method Editor

Kawamoto, Junpei
Kyushu University

Sakurai, Kouichi
Kyushu University

出版情報：Proceedings of the IEEE/CIC ICCC 2014 Symposium on Privacy and Security in Commutations, pp.209-213, 2014. IEEE Computer Society
バージョン：
権利関係：

Privacy-Aware Cloud-based Input Method Editor

Junpei Kawamoto
Kyushu University
744 Motooka, Nishiku
Fukuoka, Japan
Email: kawamoto@inf.kyushu-u.ac.jp

Kouichi Sakurai
Kyushu University
744 Motooka, Nishiku
Fukuoka, Japan
Email: sakurai@inf.kyushu-u.ac.jp

***Abstract*—Cloud services are useful but privacy concerns are still one of the most important problem of them. In this paper, we focus cloud-based input method editor and introduce a privacy-aware framework of it. Input method editor (IME) is a kind of software to help us to input Japanese, Chinese, Korean, etc. It receives input words consists of alphabets and returns transformed words consists of Kanji in Japanese, etc. We assume to deploy this IME on a cloud server. In this case, input words received IME may include people's sensitive information eg. credit card number. We employ a seachable encryption scheme and introduce a framework that people send encrypted inputs and IME computes transformed words without decrypting them.**

I. Introduction

Cloud services are useful but privacy concerns are still one of the most important problem of them. In this paper, we focus cloud-based input method editor and introduce a privacy-aware framework of it. Input method editor (IME) is a kind of software to help us to input Japanese, Chinese, Korean, etc. It receives input words consists of alphabets and returns transformed words consists of Kanji in Japanese, etc. Actually, some major companies developed cloud-based IME.

In the cloud-based IME, a user sends an input word to a cloud server and the server returns a set of candidate words of transformation. Service provide can update the mapping table of input words and transformed words on the fly because the mapping table is in the cloud server. Note that, traditional IME stores the mapping table in the user's computer and for updating the user needs to download a new table. Using cloud server, users do not need to download and update.

However, cloud-based services have a privacy concern. Actually, a cloud-based IME collect users' input in secret and it became a big problem in Japan[1]. Input words received IME may include people's sensitive information eg. credit card number. In order to use cloud-based IME without privacy concern, we need to hide any input words to cloud-server but obtain transformed words.

In this paper, we design a cloud-based IME and introduce a new searchable encryption scheme to archive privacy-aware cloud-based IME. Our IME is based on range queries on a database and our encryption scheme is a searchable encryption for range queries. The rest of this paper is organized as follows: we introduce related work in section 2; and design our cloud-based IME in section3; then we introduce our searchable encryption in section 4 and 5; finally we conclude this paper in section 6.

[1]http://www.techrepublic.com/blog/asian-technology/japanese-government-warns-baidu-ime-is-spying-on-users/

II. Related work

Among many kinds of PIR work [1], the most related work to our problem is cPIR (Computational Private Information Retrieval) [2] which assumes only one database server and ensures a private query based on complexity theory. Basic cPIR approach does not support range queries nor allow tuples have same *Key* attribute values. cPIR requires $O(n)$ computational cost on servers with the total number of tuples n. bbPIR [3] is a weak cPIR protocol which relaxes the security to reduce the computational cost on servers. It brings the idea of k-anonymity [4] in cPIR. Clients on bbPIR protocol sends a k-width bounding box containing the index which the clients want to request. After that, the clients and the server communicate with basic cPIR protocol on the sub-database consists of tuples in the bounding box. bbPIR is able to hide only which tuples in the k tuples the clients exactly request but the computational cost on servers is smaller than basic cPIR protocol.

For location-based services (LBS), there is a kind of private range query using cPIR protocol [5]. In LBS, users request some region to obtain points of interests (POIs) such as restaurants, gas stations, etc. On the approach in [5], the 2-dimensional space in the LBS is divided into n sub areas, in such a way that each sub area has at most one POI. Clients firstly compute the index of the sub area which contains the really requesting region and then the clients request the index using basic cPIR protocol. Therefore, this approach support a limited range query and users cannot request flexible regions.

Some work about encrypted database (EDB) [6] is also achieving weak private query. In the context of EDB, all tuples on servers are encrypted and the main interest is how to execute queries over the encrypted tuples. Therefore, queries also do not contain plain values and in this meaning, EDB is achieving a kind of private query. Bucketization described in [6][7] splits the domain of each attribute into some labeled

buckets and each attribute value is updated to the label of which the bucket contains the plain value by clients. This approach needs statistic information about the domains to make each bucket have almost same number of tuples, and re-builting buckets may be necessary after lots of new tuples are inserted. [8] points a problem that attackers can obtain some information about plain values by comparing the re-builted buckets and old buckets. Order Preserving Encryption Scheme (OPES) [9] provides range queries over EDB, i.e. weak private range query, and needs statistic information about domains. SCONEDB [10] provides a kind of private query using a matrix based encryption. The main purpose of SCONEDB is to achieve k-nearest neighbor (kNN) query over encrypted vector databases. SCONEDB provides a querying approach without decrypting encrypted vectors on servers. Those approaches about EDB are able to hide plain values from queries. However, in those approaches, queries which request a same range are transformed to a same query for EDB. It means those approach cannot protect the frequency analysis attack so that attackers, who know the distribution of plain queries, are able to compute the plain queries from queries for EDBs.

On the other hands, our proposal IPP method adds perturbations to queries in addition to matrix based encryption. As a result, IPP method makes transformed queries associated with a same plain query varied, so that it ensure to protect the frequency analysis of attackers. Additionaly, IPP method does not need any statistical information about domains so that IPP method is also suitable to growing databases. Thus, we choose IPP method as a basic algorithm for cloud-based input method editor we introduce in this paper.

III. Design of Cloud-based Input Method Editor

In this section, we design a cloud-based input method editor and its security model.

A. Input Method Editor

In our input method editor (IME), a user sends a word consists of alphabets to a cloud server and receives a set of transformed words. We assume there are N alphabets and denote the alphabets by $a_1, a_2, \cdots, a_N$. We also denote the set of alphabets by $\Sigma = \{a_1, a_2, \cdots, a_N\}$. Let ϵ be the *empty alphabet* and we extend the set of alphabet as $\Sigma^+ = \Sigma \cup \{\epsilon\}$.

Our IME accepts at longest L-length words. $\mathbf{w}$ denotes a word and W denotes a set of words which IME accepts as input. Letting the length of $\mathbf{w}$ be ℓ, we denote it by

$$\mathbf{w} = w_1 w_2 \cdots w_\ell \quad (\forall i, w_i \in \Sigma).$$

Where $\ell < L$, we can normalize $\mathbf{w}$ adding $L - \ell$ empty alphabets and make the length L. We denote the normalized word by

$$\tilde{\mathbf{w}} = \mathbf{w} \underbrace{\epsilon\epsilon \cdots \epsilon}_{L-\ell},$$

and the set of normalized words by $\tilde{W}$. Note that $\tilde{w}$ dose not include empty word ϵ.

We define an order on the normalized words. We firstly define an order on set of alphabets Σ^+. The order function, $\mathrm{ord} : \Sigma^+ \to \mathbb{Z}$, is as follows;

$$\mathrm{ord}(a) = \begin{cases} 0 & (a = \epsilon), \\ i & (a = a_i). \end{cases} \tag{1}$$

We define an index function, $\mathrm{index} : \tilde{W} \to \mathbb{N}$, for normalized word $\tilde{\mathbf{w}} = w_1 w_2 \cdots w_L$.

Definition 3.1 (Index of normalized words): Letting $\tilde{\mathbf{w}} \in \tilde{W}$ be normalized word $\tilde{\mathbf{w}} = w_1 w_2 \cdots w_L$, the index of $\tilde{\mathbf{w}}$ is defined as

$$\mathrm{index}(\tilde{\mathbf{w}}) = \sum_{i=1}^{L} \mathrm{ord}(w_i) N^{L-i}. \tag{2}$$

We next define neighborhoods of words by the index.

Definition 3.2 (δ-neighbor word): We say two words $\tilde{\mathbf{w}}$ and $\tilde{\mathbf{w}}'$ are δ-neighbor iff

$$\mathrm{index}(\tilde{\mathbf{w}}) - \mathrm{index}(\tilde{\mathbf{w}}') = \delta,$$

and we denote them by

$$\tilde{\mathbf{w}} = \tilde{\mathbf{w}}' + \delta, \quad \tilde{\mathbf{w}}' = \tilde{\mathbf{w}} - \delta.$$

We design IME in this paper as a system receiving word $\mathbf{w}$ from a user and returning transformed words associated with the input. We assume any inputs would be normalized and have length of L. Since there could be homonyms, some inputs would have some candidates of transformed words. We denote the set of candidates of transformed words associated with normalized input $\tilde{\mathbf{w}}$ by $R(\tilde{\mathbf{w}})$. Finally, we would define the IME system formally.

Definition 3.3 (Simple Input Method Editor): Simple input method editor is a system receiving input $\tilde{\mathbf{w}}$ and returning set of transformed words $R(\tilde{\mathbf{w}})$.

Actually, real IMEs are more complicated. One of the features usual IMEs have is collecting wrong inputs i.e. typos. Users sometimes use wrong inputs but IME should return transformed words associated with collect inputs. We design our IME would return not only transformed words associated with input $\mathbf{w}$ but also ones associated with δ-neighbors. In other words, we extend the simple IME as follows.

Definition 3.4 (δ-typing-error-friendly Input Method Editor): δ-typing-error-friendly Input Method Editor is a system receiving input $\mathbf{w}$ and returning transformed words defined as

$$R_\delta(\tilde{\mathbf{w}}) = \bigcup_{\mathbf{w} \in [\tilde{\mathbf{w}} - \delta, \tilde{\mathbf{w}} + \delta]} R(\mathbf{w}), \tag{3}$$

where δ is a given parameter.

In order to achieve the δ-typing-error-friendly IME on a cloud server by considering users' privacy, i.e. keeping inputs secret, we need a searchable encryption scheme. In the following section, we introduce a scheme which satisfies our requirement.

B. Searchable Encryption for Range Queries

We assume a service provide (*SP*) of our cloud IME stores a dictionary to compute transformed words on a cloud server (*server*). A user of our IME (*user*) sends normalized input $\tilde{\mathbf{w}}$ and receives transformed words $U_\delta(\tilde{\mathbf{w}})$. In this scenario, SP and the user does not trust the server completely thus the user wants to encrypt the input. Searchable encryption for range queries (SERQ) is a scheme that allows the server to compute results without decryption the input from the user.

We assume SP stores a set of data t in the server and we denote the set of data by D. We also assume data t consists of key attribute k and value attribute v. In other words, $t = (k, v)$. Our SERQ scheme supports searching only key attribute values. We write key attribute v and value attribute v of data t by two functions key and value, respectively, i.e. $\mathrm{key}(t) = k$, $\mathrm{value}(t) = v$. We only consider a search to find data t of which key attribute $\mathrm{key}(t) \in [\alpha, \beta] \quad (\alpha \leq \beta \in \mathbb{N})$. Thus, we denote the range queries by $q_D(\alpha, \beta)$ and

$$q_D(\alpha, \beta) = \{t \in D \mid \alpha \leq \mathrm{key}(t) \leq \beta\}.$$

Our SERQ consists of four algorithms; key generation, encryption, query generation, testing.

Definition 3.5 (Key Generation): SP generates secret key sk_c shared with the user and secret key sk_s for the server. The algorithm of key generation (KeyGen) receives parameter n and returns the pair of secret keys (sk_c, sk_s), i.e.

$$(sk_c, sk_s) \leftarrow \mathrm{KeyGen}(n).$$

Note that the server must maintain the secret key sk_s carefully.

Definition 3.6 (Key Encryption): SP encrypts key attribute $k = \mathrm{key}(t) \in \mathbb{N}$ for any data t to encrypted key vector $\mathbf{k_e}$. The encryption algorithm (SERQ) has plain key attribute k and secret key for SP and the user sk_c as inputs and computes the encrypted key vector $\mathbf{k_e}$, i.e.

$$\mathbf{k_e} \leftarrow \mathrm{SERQ}(k, sk_c).$$

SP, then, sends $(\mathbf{k_e}, v)$ to the server.

Definition 3.7 (Query Generation): The user computes encrypted query vector $\mathbf{q_e}$ and additional information x associated with range query $[\alpha, \beta](\alpha \leq \beta \in \mathbb{N})$. The algorithm to compute queries (Query) receives querying range $[\alpha, \beta]$ and secret key of user sk_c, and returns encrypted query vector $\mathbf{q_e}$ and additional information x i.e.

$$(\mathbf{q_e}, x) \leftarrow \mathrm{Query}([\alpha, \beta], sk_c).$$

The user sends $(\mathbf{q_e}, x)$ to the server.

Definition 3.8 (Testing): The server which receives query $(\mathbf{q_e}, x)$ from the user tests all data $(\mathbf{k_e}_i, v_i) \quad (i = 1, 2, \cdots)$ satisfying the range query by an algorithm (Test). The algorithm receives encrypted key vector $\mathbf{k_e}_i$, query $(\mathbf{q_e}, x)$ and secret key of the server sk_s, and returns $res_i \in \{0, 1\}$ i.e.

$$res_i \leftarrow \mathrm{Test}(\mathbf{k_e}_i, \mathbf{q_e}, x, sk_s)$$

The server collect data t_i of which res_i and make a set of v_i, finally the server sends the set to the user.

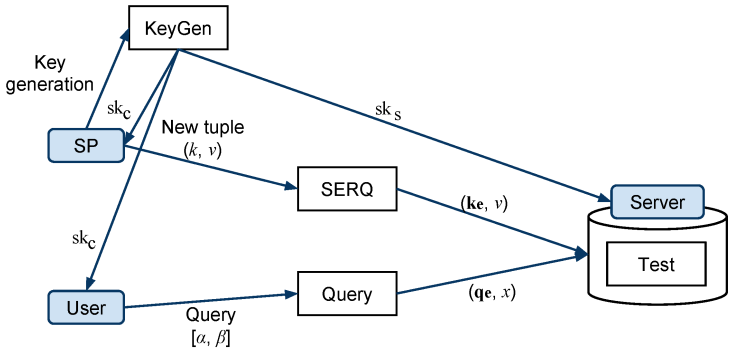

Fig. 1: Framework of the searchable encryption for range queries.

Algorithm 1 Server process of our cloud-based input method editor.

Require: Query $(\mathbf{q_e}, x)$
Require: Set of candidate data D
Require: Encryption key of the server sk_s
1: $Res \leftarrow \emptyset$
2: **for** each tuple t in D **do**
3: $\quad \mathbf{k_e} \leftarrow \mathrm{key}(t)$
4: $\quad$ **if** $\mathrm{Test}(\mathbf{k_e}, \mathbf{q_e}, x, sk_s) = 1$ **then**
5: $\quad\quad Res \leftarrow Res \cup \mathrm{value}(t)$
6: $\quad$ **end if**
7: **end for**
8: **return** Res

C. Secure Input Method Editor

Figure **??** shows the framework of SERQ we introduced in section III-B. In this section, we design a privacy-aware cloud-based IME based on the framework.

We assume the service provide of our cloud-based IME has set of transformed words $R(\tilde{\mathbf{w}})$ for each normalized word $\tilde{\mathbf{w}}$. The assumption is as same as usual IME. SP stores those sets into the server as $t = (\tilde{\mathbf{w}}, R(\tilde{\mathbf{w}}) \quad (\forall w \in W)$. In order to employ the SERQ framework we introduce the previous section, key attribute values must be a natural number. Thus, we employ the index we defined in equation (2) and use encrypted key vector

$$\mathbf{k_e} = \mathrm{SERQ}(\mathrm{index}(\tilde{\mathbf{w}}), sk_c).$$

SP computes $(\mathrm{SERQ}(\mathrm{index}(\tilde{\mathbf{w}}), sk_c), R(\tilde{\mathbf{w}}))$ for all $\tilde{w} \in \tilde{W}$ and sends them to the server.

When the user inputs word $\mathbf{w}$ and translates it by our cloud-based IME, the client of the IME normalizes the input and makes query $(\mathbf{q_e}, x)$ by the algorithm Query, i.e.

$$(\mathbf{q_e}, x) = \mathrm{Query}(\mathrm{index}(\tilde{\mathbf{w}}) - \delta, \mathrm{index}(\tilde{\mathbf{w}}) + \delta, sk_c).$$

Then, the user sends the query to the server.

The server runs the algorithm Test for all data $t \in D$ and collect a set of data t of which the result of Test is 1. Finally, the server makes a set of $\mathrm{value}(t)$ in the set and returns it to

user. Algorithm 1 shows the process in the server. Considering discussions in this section, we can say Res which the server sends to the user consists of transformed words associated with words in $[\tilde{\mathbf{w}}-\delta, \tilde{\mathbf{w}}+\delta]$, where $\tilde{\mathbf{w}}$ is the normalized input. In other words, Res is equal to $R_\delta(\tilde{\mathbf{w}})$ defined in equation (3).

IV. Query Processing by Inner Product Predicate

We introduce a scheme of searchable encryption for range queries named IPP method [11]. IPP method adds random perturbations to key attribute values and queries in order to keep them secret. We extend the scheme for our cloud-based IME. In this section, we introduce the basic idea of IPP method.

IPP method adds random perturbation r_k to key attribute value k. It is chosen from $0 < r_k < 1/2$ and the perturbed value is $k+r_k$. We denote the function of adding perturbation by

$$\mathrm{per} : \mathbb{N} \times \mathbb{R} \to \mathbb{R},\ \mathrm{per}(x; r_k) = x + r_k.$$

IPP method adds perturbations all key attribute value in the set of data D. We denote the new data set consists of perturbed key attribute by PD, thus, PD is defined by

$$PD = \{(\mathrm{per}(\mathrm{key}(t); r_k \underset{r}{\leftarrow} [0, \frac{1}{2}]), \mathrm{value}(t)) \mid \forall t \in D\},$$

where $r_k \underset{r}{\leftarrow} [0, 1/2]$ denotes a process to choose random value r_k from $[0, 1/2]$. Query $q_D(\alpha, \beta)$ for perturbed data set PD is also modified and

$$q_{PD}(\alpha, \beta) = \{t \in PD | \alpha - \frac{1}{2} \le \mathrm{key}(t) \le \beta + \frac{1}{2}\}.$$

Note that since $k \in \mathbb{N}$ and $0 < r_k < 1/2$, $q_{PD}(\alpha, \beta)$ exact requesting data t of which key attribute $\mathrm{key}(t)$ is in $[\alpha, \beta]$.

A. Polynomial Predicate

In order to add perturbations to queries, we represent queries by *inner product predicates*. We introduce *polynomial predicate* which is the base of inner product predicate.

In polynomial predicate, we represent queries using a kind of polynomial function $p : \mathbb{N} \to \mathbb{R}$. The range of the function is rial number, but we employ the following *sign function*,

$$\mathrm{sign}(x) = \begin{cases} 1 & (x \ge 0) \\ 0 & (\text{otherwise}), \end{cases}$$

and employ $\mathrm{sign} \circ p$ in order to design the algorithm Test we introduce in section III-B.

There are possible polynomial functions suitable for representing range query q_D. We choose one of the simplest one i.e.

$$p_{[\alpha,\beta]}(k) = -(k-\alpha)(k-\beta).$$

For this function, $\mathrm{sign} \circ p_{[\alpha,\beta]}(k) = 1$ iff $k \in [\alpha, \beta]$.

We extend the query q_D to q_{PD} in order to handle perturbed data. Thus, we also extend the polynomial function $p_{[\alpha,\beta]}(k)$ to represent q_{PD}, and that is

$$p'_{[\alpha,\beta]}(k) = -(k - \alpha + \frac{1}{2})(k - \beta - \frac{1}{2}).$$

We next add a perturbation to the polynomial function $p'_{[\alpha,\beta]}(k)$. We choose positive random value r_q and multiply $(k + r_q)$ to $p'_{[\alpha,\beta]}(k)$. Thus, the perturbed query function $\tilde{p}'_{[\alpha,\beta];r_q}(k)$ associated with $p'_{[\alpha,\beta]}(k)$ is

$$\tilde{p}'_{[\alpha,\beta];r_q}(k) = \mathrm{per}(p'_{[\alpha,\beta]}(k); r_q) = -(k-\alpha+\frac{1}{2})(k-\beta-\frac{1}{2})(k+r_q). \tag{4}$$

Since we assume $k \in \mathbb{N}$, equation (4) will be grater than or equal to zero if $k \in [\alpha, \beta]$. In other words, $\mathrm{sign} \circ \tilde{p}'_{[\alpha,\beta];r_q}(k) = 1$ iff $k \in [\alpha, \beta]$.

B. Inner Product Predicate

We now introduce the *inner product predicate*. Generally, any polynomial functions of a single variable can be represented by inner product of two vectors. We use the fact and represent equation (4) as an inner product of constant vector

$$\mathbf{q}_{[\alpha,\beta];r_q} = \begin{pmatrix} -1 \\ \alpha + \beta - r_q \\ -(\alpha - \frac{1}{2})(\beta + \frac{1}{2}) + (\alpha + \beta) r_q \\ -(\alpha - \frac{1}{2})(\beta + \frac{1}{2}) r_q \end{pmatrix}^t \tag{5}$$

and variable vector $\mathbf{k} = (k^3, k^2, k, 1)^t$ i.e. $\mathbf{q}_{[\alpha,\beta];r_q} \cdot \mathbf{k}$, where $\mathbf{x}^t$ denotes a transposed vector of $\mathbf{x}$ and $\mathbf{x} \cdot \mathbf{y}$ denotes an inner product of $\mathbf{x}$ and $\mathbf{y}$.

We use key vector $(k^3, k^2, k, 1)^t$ instead of key attribute value k. Then, an inner product predicate associated with range query q_D is

$$\mathrm{IPP}_{[\alpha,\beta],r_q}(\mathbf{k}) = \mathbf{q}_{[\alpha,\beta],r_q} \cdot \mathbf{k}.$$

We add a perturbation to the key attribute value and $\mathrm{per}(k, r_k) = k + r_k$, thus key vector associated with the perturbed key value $\mathrm{per}(k, r_k)$ is

$$\mathbf{k}' = (\mathrm{per}(k; r_k)^3, \mathrm{per}(k; r_k)^2, \mathrm{per}(k; r_k), 1)^t. \tag{6}$$

As we discussed in previous section, polynomial predicate $\tilde{p}'_{[\alpha,\beta],r_q} \ge 0$ iff $k \in [\alpha, \beta]$. Since the inner product predicate $\mathrm{IPP}_{[\alpha,\beta];r_q}$ is as same as $\tilde{p}'_{[\alpha,\beta],r_q}$, $\mathrm{IPP}_{[\alpha,\beta];r_q} \ge 0$ iff $k \in [\alpha, \beta]$. Finally, $\mathrm{sign} \circ \mathrm{IPP}_{[\alpha,\beta];r_q}(\mathbf{k}) = 1$ iff $k \in [\alpha, \beta]$.

V. Searchable Encryption by Cyclic Matrix

IPP method [11] employs a regular matrix M to encrypt key vector $\vec{k'}$ and query vector $\vec{q}_{[\alpha,\beta];r_q}$. The encrypted vectors of them are $M^{-1}\vec{k'}$ and $M^t\vec{q}_{[\alpha,\beta];r_q}$, respectively, and the inner product of them can be computed as

$$M^t\vec{q}_{[\alpha,\beta];r_q} \cdot M^{-1}\vec{k'} = \vec{q}^t_{[\alpha,\beta];r_q} M M^{-1} \vec{k'} = \vec{q}_{[\alpha,\beta];r_q} \cdot \vec{k'}.$$

It means the encryption scheme does not change inner product by encryption.

On the other hand, in this framework, an attacker who obtains a key vector and a query vector would know the key vector satisfies the query by computing inner product of them. We introduce secret key of the server and avoid attackers compute inner products.

Algorithm 2 KeyGen

Require: Big prime parameter n
1: Generate n-ordered cyclic matrix A
2: Choose $c, s < n$, where c and s are relatively prime
3: $sk_c \leftarrow (A, n, c, s)$, $sk_s \leftarrow A^s$
4: **return** (sk_c, sk_s)

A. Encryption by Cyclic Matrix

We employ a n-ordered cyclic matrix as an encryption key.

Definition 5.1: We say matrix A is a n-ordered cyclic matrix iff

$$A^n = E, \quad A^i \neq E \quad (i < n),$$

where E be the unit matrix.

We define secret key of the user sk_c by $sk_c = (A, n, c, s)$, where c and s are random numbers less than n and must be relatively prime. The secret key of the server is $sk_s = A^s$.

The encryption algorithm of key vector $\mathbf{k}'$ is as follows. At first, it generates a random number r_e, then computes $r_e A^c \mathbf{k}'$ as the encrypted key vector $\mathbf{k_e}$. On the other hand, the encryption algorithm of query vector $\mathbf{q}$ generates a random number r and computes $(A^r)^t \mathbf{q}$ as the encrypted query vector $\mathbf{q_e}$. It also solves x of the formulation

$$sx + c + r = 0 \pmod{n}, \tag{7}$$

and use x as the additional information of the query.

Theorem 5.1 (Inner product of encrypted vectors): The server computes inner product of plain key vector $\mathbf{q} \cdot \mathbf{k}'$ by computing $\mathbf{q_e} \cdot (A^s)^x \mathbf{k_e}$ with the secret key of the server A^s.

Proof:

$$\begin{aligned} \mathbf{q_e} \cdot (A^s)^x \mathbf{k_e} &= (A^r)^t \mathbf{q} \cdot (A^s)^x r_e A^c \mathbf{k}' \\ &= r_e \mathbf{q}^t A^r A^{sx} A^c \mathbf{k}' \\ &= r_e \mathbf{q}^t A^{sx+c+r} \mathbf{k}' \\ &= r_e \mathbf{q}^t \mathbf{k}' = \mathbf{q} \cdot \mathbf{k}' \end{aligned}$$

■

Thus, the server can find data t satisfies a query by checking

$$\mathrm{sign}(\mathbf{q_e} \cdot (A^s)^x \mathbf{k_e}) \tag{8}$$

is 1. Additionally, attackers who obtain key vectors and query vectors no more compute inner products without the secret key sk_s.

B. Algorithms of SERQ

SP generates secret key of the user sk_c and secret key of the server sk_s by the algorithm KeyGen described in Algorithm 2. KeyGen generates a four dimension n-ordered cyclic matrix A, and random and relatively prime numbers c and s, where n be a given parameter. KeyGen, then, computes two secret keys and outputs them. SP finally sends sk_c to the user and sk_s to the server.

To add new data (k, v), SP encrypts k to encrypted key vector $\mathbf{k_e}$ by the algorithm SERQ shown in Algorithm 3. SERQ at first generates random value $r_k \in [0, 1/2]$ and computes key vector $\mathbf{k}'$ defined by equation (6). SERQ then generates another random value r_e and encrypts the key vector by A and c from the secret key sk_c. SP finally sends data with encrypted key vector $(\mathbf{k_e}, v)$ to the server.

Algorithm 3 SERQ

Require: Attribute value $k \in \mathbb{N}$
Require: Encryption key of clients $sk_c = (A, n, c, s)$
1: $r_k \xleftarrow{r} [0, 1/2]$
2: $\mathbf{k}' = \left(\mathrm{per}(k; r_k)^3, \mathrm{per}(k; r_k)^2, \mathrm{per}(k; r_k), 1\right)^t$
3: $r_e \xleftarrow{r} R^+$
4: $\mathbf{k_e} \leftarrow r_e A^c \mathbf{k}'$
5: **return** $\mathbf{k_e}$

Algorithm 4 Query

Require: Querying range $[\alpha, \beta] (\alpha \leq \beta \in D_K)$
Require: Encryption key of clients $sk_c = (A, n, c, s)$
1: Compute $\mathbf{q}$ by (5)
2: $r \xleftarrow{r} \mathbb{Z}$
3: $\mathbf{q_e} \leftarrow (A^r)^t \mathbf{q}$
4: Solve for x: $sx + c + r = 0 \pmod{n}$
5: **return** $(\mathbf{q_e}, x)$

Algorithm 5 Test

Require: Encrypted key vector $\mathbf{k_e}_i$
Require: Query $(\mathbf{q_e}, x)$
Require: Secret key of the server $sk_s = A^s$
1: **return** $\mathrm{sign}(\mathbf{q_e} \cdot (A^s)^x \mathbf{k_e}_i)$

Algorithm 6 Encrypted Range Search with Query

Require: Query $(\mathbf{q_e}, x)$
Require: Database PD
Require: Encryption key of the server $sk_s = A^s$
1: $Res \leftarrow \emptyset$
2: **for** each tuple t in PD **do**
3: $\quad \mathbf{k_e} \leftarrow \mathrm{key}(t)$
4: $\quad$ **if** $\mathrm{Test}(\mathbf{k_e}, \mathbf{q_e}, x, sk_s) = 1$ **then**
5: $\quad\quad Res \leftarrow Res \cup \{\mathrm{value}(t)\}$
6: $\quad$ **end if**
7: **end for**
8: **return** Res

The user who wants to have data of which key attribute value k in range $[\alpha, \beta]$ makes a query by the algorithm Query shown in Algorithm 4. Query at first computes a query vector defined in equation (5). Query then generates random natural number r and computes encrypted query vector $\mathbf{q_e}$. It also solves x from formulation (7). Finally, it outputs pair $(\mathbf{q_e}, x)$. The user sends $(\mathbf{q_e}, x)$ to the server as the query.

The server who receives query $(\mathbf{q_e}, x)$ from the user applies the algorithm Test for all data it has in order to check each data should be included into the returning data. Algorithm 5

shows the algorithm Test and it evaluates equation (8). The server returns data for which the Test returns 1 to the user. This process is shown in Algorithm 6.

Applying those algorithms to the discussion in section III-C, we can archive a cloud-based IME of which users do not need expose what they input to the server, i.e. privacy-aware cloud-based IME.

VI. Conclusion

In this paper, we focus cloud-based input method editor and introduce a privacy-aware framework of it. Our cloud-based IME supports miss typing and archiving such IME with encryption, we introduce a searchable encryption for range query scheme. We employ an idea of inner product predicate and our searchable encryption scheme is based on cyclic matrix. We also introduce our querying process over the encryption scheme and a theorem it can compute exact results we want. As a future work, we will implement the IME and evaluate computational costs and actual querying times.

Acknowledgment

This work is partly supported by The Nakajima Foundation, Artificial Intelligence Research Promotion Foundation, and Grant-in-Aid for Young Scientists (B) (26730065), Japan Society for the Promotion of Science (JSPS).

References

[1] R. Ostrovsky and W. E. Skeith, III, "A Survey of Single-Database PIR: Techniques and Applications," in *Proc. of the 10th International Conference on Practice and Theory in Public-key Cryptography*. Beijing, China: Springer, 2007, pp. 393–411.

[2] E. Kushilevitz and R. Ostrovsky, "Replication Is Not Needed: Single Database, Computationally-Private Information Retrieval," in *Proc. of the 38th Annual Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 1997, pp. 364–373.

[3] S. Wang, D. Agrawal, and A. E. Abbadi, "Generalizing PIR for Practical Private Retrieval of Public Data," in *Proc. of the 24th Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy*. Rome, Italy: Springer, 2010, pp. 1–16.

[4] L. Sweeney, "k-Anonymity: A Model for Protecting Privacy," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 5, pp. 1–14, 2002.

[5] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan, "Private Queries in Location Based Services: Anonymizers are Not Necessary," in *Proc. of the 28th ACM SIGMOD International Conference on Management of Data*. Vancouver, BC, Canada: ACM Press, 2008, pp. 121–132.

[6] H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra, "Executing SQL over Encrypted Data in the Database-Service-Provider Model," in *Proc. of the 21st ACM SIGMOD International Conference on Management of Data*. Madison, WI, USA: ACM Press, 2002, pp. 216–227.

[7] B. Hore, S. Mehrotra, and G. Tsudik, "A Privacy-Preserving Index for Range Queries," in *Proc. of the 30th International Conference on Very Large Data Bases*. Toronto, ON, Canada: VLDB Endowment, 2004, pp. 720–731.

[8] R. C.-W. Wong, A. W.-C. Fu, J. Liu, K. Wang, and Y. Xu, "Global Privacy Guarantee in Serial Data Publishing," in *Proc. of the 26th International Conference on Data Engineering*. Long Beach, CA, USA: IEEE Computer Society, 2010, pp. 956–959.

[9] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order Preserving Encryption for Numeric Data," in *Proc. of the 23rd ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM Press, 2004, pp. 563–574.

[10] W. K. Wong, D. W.-L. Cheung, B. Kao, and N. Mamoulis, "Secure kNN Computation on Encrypted Databases Categories and Subject Descriptors," in *Proc. of the 35th SIGMOD International Conference on Management of Data*. Providence, RI, USA: ACM Press, 2009, pp. 139–152.

[11] J. Kawamoto and M. Yoshikawa, "Private Range Query by Perturbation and Matrix Based Encryption," in *Proc. of the Sixth IEEE International Conference on Digital Information Management*. Melbourne, Australia: IEEE Computer Society, 2011, pp. 211–216.