Address Permutation for Privacy-Preserving Searchable Symmetric Encryption

Daeseon Choi, Seung-Hyun Kim, and Younho Lee

This paper proposes a privacy-preserving database encryption scheme that provides access pattern hiding against a service provider. The proposed scheme uses a session key to permute indices of database records each time they are accessed. The proposed scheme can achieve access pattern hiding in situations in which an adversary cannot access the inside of the database directly, by separating the entity with an index table and data table and permuting both the index and position where the data are stored. Moreover, it is very efficient since only O(1) server computation and communication cost are required in terms of the number of the data stored. It can be applied to cloud computing, where the intermediate entities such as cloud computing service provider can violate the privacy of users or patients.

Keywords: Database security, privacy, searchable symmetric encryption, access pattern.

Manuscript received Apr. 22, 2011; revised Aug. 15, 2011; accepted Sept. 6, 2011.

This work was supported by Yeungnam University research grants in 2009. Also, this research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science, and Technology (2010-0006355).

Daeseon Choi (phone: +82 42 860 1308, sunchoi@etri.re.kr) and Seung-Hyun Kim (ayo@etri.re.kr) are with the Cyber Security-Convergence Research Department, ETRI, Deajeon, Rep. of Korea.

Younho Lee (corresponding author, younholee@yu.ac.kr) is with the Department of Information and Communication Engineering, Yeungnam University, Gyeongsan, Rep. of Korea.

http://dx.doi.org/10.4218/etrij.12.0111.0243

I. Introduction

Cloud services delegate local data to a remote third party and allow users to access their data anywhere, anytime. While cloud services have become more popular recently, they have many performance, security, and privacy requirements.

Searchable encryption schemes can be used for cloud services to support security and privacy. There should be a way to separate users' privacy from their service providers. In this case, it is good to use searchable encryption because it separates users' information from their service providers and preserves the searchability of the encrypted data. Thus, an efficient data search on the encrypted data is possible.

However, previous existing searchable encryption schemes are focused on performance, not high-level privacy such as hiding the access pattern [1]-[6]. Even if a database is encrypted, an access pattern can be leaked by monitoring data transmitted between a user and third party over time. An access pattern refers to the sequences and frequency of documents accessed by a user.

Leaking an access pattern causes a significant loss of privacy to the user. For example, in stock trading applications, even if the messages about stock trading are encrypted, trading bias in a particular event, or the frequency of buying/selling a specific stock, can be exposed. Moreover, an access pattern helps predict the actual data, as in [3].

A few schemes that support an access pattern hiding property suffer from extreme inefficiency [7]. Thus, they are not used in practice.

In this paper, we introduce a new encryption scheme that supports access pattern hiding, as well as searchability. We consider how the protocol can be made using the proposed scheme. The proposed scheme can be used in a specific environment, where a cloud computing service provider is in the middle of the users and the data servers, and thus it can access all intermediate information transferred between them. The proposed scheme achieves the access pattern hiding property even with O(1) server computations and O(1) communication. This has not been previously suggested. Therefore, the proposed scheme provides a high level of privacy and efficiency.

The organization of this paper is as follows. We show the related works in section II. Section III shows the motivation and presumed environment of the proposed scheme. Section IV details the proposed scheme itself. Section V analyzes its security and performance. Finally, we offer some concluding remarks in section VI.

II. Related Works

We categorize related work into three topics. Each topic is summarized in its own corresponding subsection.

1. Access Pattern Hiding

Much research has dealt with access pattern hiding thus far. The authors in [4] considered a secure database system that can hide query patterns. It employed the trusted computing device (TCD), and thus the secret values to encrypt the data and the codes for encrypting/decrypting the records are stored in TCD, while the actual data are stored in conventional storage. The hiding of access patterns is achieved since the cryptographic operations are done in TCD, and even the server cannot access its internals. However, there are some countries, such as Russia, that do not allow the use of the trusted platform module (TPM). Also, there are some privacy issues regarding the use of TPM. For example, it may occur that a user cannot use the application she wants to use because TPM does not allow it. Thus, it cannot be scaled in practice yet. The authors in [7] proposed oblivious RAM, where devices using oblivious RAM cannot know which part of the memory they are accessing. Thus, this supports access pattern hiding. However, a unit operation consumes a huge amount of cost. That is, it requires O($\log^3 n$) cost to access a single memory unit, where n is the number of unit items stored in the memory. Therefore, this approach does not seem practical.

Among the research related with hiding an access pattern, [8], [9], and [10] deal with single-database private information retrieval (PIR). In [8], a server does not know the real index of the database a user accesses. According to [8], a user does not expose the access pattern. However, the scheme in [8] always handles unencrypted data and should touch every data item to hide the access pattern. Therefore, this scheme requires a number of operations which depend linearly on the size of the database. Consequently, the scheme in [8] requires much time applying the searchable symmetric encryption (SSE) because it runs on plaintexts.

2. Searchable Encryption

Some searchable encryption researches do not provide access pattern hiding [1]-[6], [11]. As far as we know, the scheme in [6] is the first approach dealing with searching encrypted data in a symmetric key setting. In [6], the scheme reveals the access pattern of keywords. The access pattern is revealed to the database manager, unlike in the proposed scheme, since it uses deterministic encryption when the keyword is encrypted.

Two of the most representative schemes can be found in [1] and [5]. The authors in [5] mention the limitations of IND-CKA, which is used to prove the security in the previous related works, and modifies the definition of adaptive security. In addition, [1] proposes the SSE-1 and SSE-2 algorithms. SSE-1 was proved to be non-adaptive secure and manages the document's index using a hash table and a linked list. However, SSE-1 does not meet the adaptive security [1] because the linked list in SSE-1 fixes the association to documents and exposes the order and list of returned documents for the same word.

For adaptive security, SSE-2 makes trapdoors that are 1:n mapped with a keyword, and does not use the linked list in SSE-1. To return the index of same sized documents, SSE-2 fixes the number of trapdoors as the size of the longest plaintext document 'max'. Therefore, there is some overhead. For example, in the case in which the number of documents containing word w is 3, the trapdoors are $T_{w;1}, T_{w;2}, T_{w,3}, \ldots, T_{w,max}$. The scheme in [1] hides the search pattern and satisfies the adaptive security, as there are different 'max' trapdoors for a word. However, the indexes of returned documents and trapdoors for the same word are always identical. Therefore, the scheme in [1] exposes the access pattern.

Recently, [11] was suggested to support the verifiability of search result using trie and keyed-hash function. Unfortunately, it does not support the access pattern hiding either.

Apart from the above approaches, some keyword searching schemes are based on public key cryptography [12]-[14]. In general, they support high levels of security and privacy such as access pattern hiding. However, as [1] mentioned already, this requires heavy computational cost for a single search operation. Thus, they do not look practical as symmetric searchable encryption schemes due to this property.

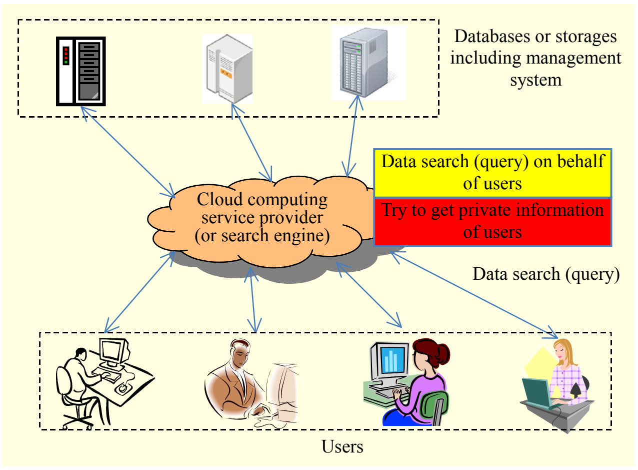
3. Other Related Works on Database Security

Many works deal with database security. The scheme in [15] has a somewhat different setting from the proposed scheme, in the sense that servers can access the data while the search keywords are hidden from the servers. In the proposed scheme, both the search keywords and data are regarded as sensitive information, and therefore they are encrypted. Also, [15] is based on Paillier's homomorphic cryptosystem in [16]. This is a public key cryptosystem, and therefore it takes more resources than a symmetric cryptosystem.

Unlike the proposed scheme, whose main target is searching based on exact matching, [17] is aimed at similarity-based matching. The schemes in [18], [19] can be classified into this type. Using the similarity measure, they suggested a ranked search algorithm to retrieve up to k related documents. In [18], the scheme employs the order-preserving symmetric encryption [20] to protect the similarity score and to keep maintaining ranked searchability. In [19], the authors proposed a multikeyword search, where a query has more than a single word. As the authors show in [18], [19], they do not support access pattern hiding. Similarity-based matching is useful for text retrieval, where exact matching does not produce the information the user needs. Thus, there is little relevance between [17]-[19] and the proposed work.

The authentication of data in an outsourced database is dealt with in [21], [22]. The service provider may forge the stored data due to the separation of data owner and the service provider (outsourced database). Some type of security mechanism should be supported to prevent such forgery and provide assurance of the query result. This issue is orthogonal to the proposed work because our work is about privacy preservation on an encrypted database, while [21], [22] are about the authentication of data that are in clear plaintext. The manner in which to verify the query result in a database as a service model is dealt with in [13]. It employs the aggregate signature and Merkle hash chain to show the correctness of the result of the range query.

Gennaro and others proposed non-interactive verifiable computing [21] based on a fully homomorphic encryption scheme [22], [23] for secure cloud computing. In [23], the server can let the client compute some functions with some input values without letting the client know them. The client does not even know the actual results of the computation. This property can be achieved using a fully-homomorphic encryption scheme [24], [25]; even without decrypting ciphertexts, it is possible to compute any kind of function taking the values of the plaintexts hidden in the ciphertexts as inputs. In addition, the client can prove to the server that the function has been computed correctly, with very little computation overhead compared to the original work to be done. Unfortunately, it cannot be used in practice as it takes a significant amount of time, that is, on the order of tens of minutes, to compute even very primitive computations, such as bit-wise addition and multiplication, because the underlying homomorphic encryption scheme is very slow [23] even when using a strong contemporary machine such as an IBM System x3500 server.


Databases or storages including management system
Data search (query) on behalf of users
Try to get private information of users
Cloud computing service provider (or search engine)
Data search (query)
Users
Fig. 1. Motivating environment.

III. Motivation and Problem Definition

This section shows the need for a new scheme and states the problem definition. We also explain the environment assumed in this paper. Finally, we define the protocol we need to solve the problem.

1. Motivation

Suppose there are three types of entities: users, a service provider, and a database. Users store their data in a third-party database in an encrypted form. They perform this action via the service provider. Therefore, the service provider does not have information about the databases, and vice versa. Instead, the service provider links the users to the databases. The service provider executes data search/add/delete operations after receiving the users' requests, and it returns the results to them after the third-party databases send the search/add/delete results to the service provider.

In this situation, we want to preserve the privacy of the users from third-party databases and their service provider. In this case, we can provide searchability and privacy by employing searchable encryption [1]-[6]. Unfortunately, the existing schemes cannot hide the access pattern of users from the service provider. In view of third-party databases, the threat of a privacy violation is not a big issue because they do not know the identities of the users behind the service provider. However, the service provider can be hostile in terms of the users' privacy because the private information of users has a commercial value under certain situations.

Figure 1 shows the environment we explained thus far. It is well suited to a cloud-computing environment, where users outsource the search ability as well as the databases to external companies.

We have termed the scheme searchable symmetric encryption with access pattern hiding (SSE-APH) because it should support searchability and access pattern hiding.

2. Scenario of Proposed Scheme

We first introduce the entities appearing in the scenario. *Users* have documents, make indices for them, and store them in the *index server* and *document server*. In addition, all communication between the user and servers is via the *service provider*. Therefore, there are four types of entities in the system.

Figure 2 shows the scenario of the proposed scheme. The scenario has two phases. At the initial phase, the secret keys are shared among the user and servers. The service provider is assumed not to execute an active attack but perform passive attacks. We suppose this is done with an existing scheme such as a tripartite key exchange protocol [26]. Also, the user generates indices for the documents stored and sends both the documents and indices to the document server. The document server stores the documents in the appropriate places indicated by the indices. It removes the indices after storage. The index server only has the indices coming from the user. We separate the index server and document server to support flexibility. Therefore, it can be combined into a single entity under certain environments.

The document searching procedure is shown at the bottom section of Fig. 2. Whenever a user wants to retrieve documents that are mapped to a key *w*, she generates a trapdoor for the keyword. The trapdoor is sent to the service provider, and it retrieves the document IDs that correspond to the trapdoor from the index server. They are forwarded to the document server and the server replies with the corresponding encrypted documents to the service provider. The service provider sends them to the user after they receive them.

3. Problem Definition

We first describe the properties that the proposed scheme should meet in terms of security and functionality. Then, we discuss why such requirements are needed.

i) **Searchability.** The service provider can obtain the documents that the user wants and send them back to the user.

ii) **Confidentiality**

- No entities other than the user can see the plain form of the documents.
- The service provider cannot make a new trapdoor without the user's intervention.

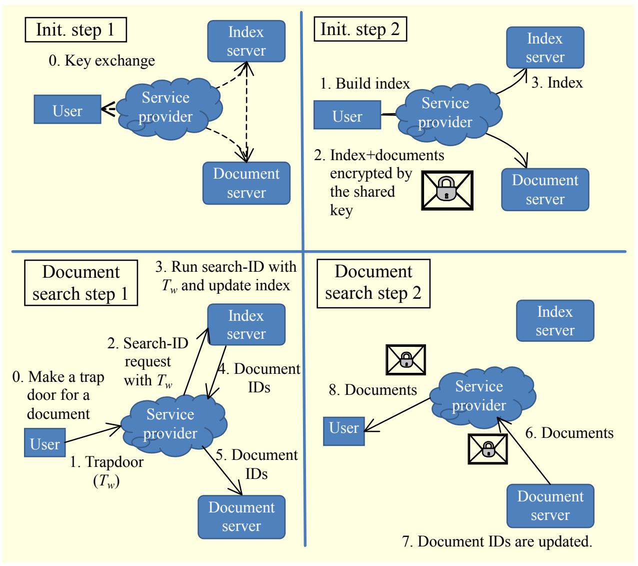


Fig. 2. Scenario of proposed scheme.

iii) **Access pattern hiding** (from the service provider). The service provider cannot obtain the document access pattern of the user without the help of other entities.

iv) **Efficiency.** The search complexity should not depend on the size of the stored documents.

The requirements are analogous to previous works [1], [2], [6], [18], [19] except the access pattern hiding requirement and the efficiency requirement. Since access pattern hiding requirement includes weak privacy requirements such as keyword privacy and trapdoor unlinkability in [18], we only address access pattern hiding. The efficiency requirement is somewhat stronger than those in the previous works [11], [18], [19] in the sense that the communication and computation of the server caused by searching is independent of the number of data stored. To our knowledge, no methods support both the third and fourth properties [1]-[6], [11], [18], [19]. Our main contribution is to propose a scheme meeting all the above requirements.

We assume that all of the entities in the scenario maintain the protocol. As an active attack is out of the scope of this paper, we do not consider that type of attack and only consider passive attacks. The servers can try to break the confidentiality of a user's data. The service provider is the main possible adversary in the scenario. It can try to break the confidentiality including the trapdoor and access pattern hiding.

IV. Proposed Scheme

This section proposes a searchable encrypted database scheme that satisfies security and privacy. We formally define the proposed scheme and its security model and prove that the proposed scheme meets the security model. The proposed scheme and its security model are based on the discussion thus far.

1. Notations

We borrow the notations from the authors in [1] and the data structures used by their construction. Let $\Delta = \{w_1,\ldots, w_d\}$ be a dictionary of d words and 2^Δ be the set of all possible documents. Further, let $D \subseteq 2^\Delta$ be a collection of n documents where $D = (D_1,\ldots,D_n)$ and 2^{2^Δ} is the set of all possible document collections. Let $id(D)$ be the identifier of document D, where the identifier can be any string that uniquely identifies a document. Let Δ', $\Delta' \in \Delta$, be the set of distinct words that exist in document collection D, and $D(w)$ be the set of identifiers of documents in D that contain the w, such as a memory location. We denote by $D(w)$ the lexicographically ordered list consisting of the identifiers of all documents in D that contain the w.

We write $x \leftarrow X$ to represent an element x being sampled from a distribution X, and $x \overset{R}{\leftarrow} X$ represents an element x being sampled uniformly from set X. The output x of an algorithm A is denoted by $x \leftarrow A$. We write $||$ to denote a string concatenation. We say a function $v:N \rightarrow N$ is negligible if for every polynomial $p(\cdot)$ and all sufficiently large k, $v(k) < 1/p(k)$.

We sometimes refer to $D(w)$ as the outcome of a search for w and to the sequence $(D(w_1), \ldots, D(w_n))$ as the access pattern of a client. We also define the search pattern of a client as any information that can be derived from knowing if two arbitrary searches were performed for the same word.

In addition to encryption schemes, we also make use of a pseudo-random permutation, which is a polynomial-time computable function that cannot be distinguished from a random function by any probabilistic polynomial-time adversary.

2. Definitions

Here, we define the proposed SSE-APH. We extend the definition of SSE given in [1]. Note that some of the names of the algorithms such as BuildIndex and Trapdoor are borrowed from previous works [1], [18], [19]. However, the details of the proposed scheme are different from them.

Definition 1. Searchable symmetric encryption scheme with access pattern hiding. An SSE-APH scheme is a collection of five polynomial-time algorithms and a protocol (Keyshare, BuildIndex, Trapdoor, Search_Index, and Search_Document):

- KeyShare(1^k) is a probabilistic key sharing protocol between the user U, the index server, and the document server. The result of this protocol is that all the entities share a common secret key K. The length of K is polynomially bounded in k.
- BuildIndex(K,D) is a (possibly probabilistic) algorithm run by user U to generate indexes. It takes a secret key K and D, which is a document collection polynomially bounded in k, as inputs, and returns an index I such that the length of I is polynomially bounded in k.
- Trapdoor(K, w, i) is run by the user U to generate a trapdoor for a given word. It takes a secret key K, a w, and an index i as inputs, and returns a trapdoor $T_{w,i}$.
- Search_Index(IT, $T_{w,i}$) is run by the index server IS to search for the i-th document in D that contains a w. It takes an index table IT for a collection D, and a trapdoor $T_{w,i}$ for a w, as inputs, and returns the i-th identifier of documents containing w.
- Search_Document(DT, *document_id*) is run by the document server DS to retrieve an encrypted document identified by the id. DS searches a data table DT with id and returns a corresponding document to the user U.

The following definitions are given to define the access hiding property and other security properties. These definitions are similar to those in [1]. However, there is a slight difference due to the differences of the setting. We borrow the adaptive security definition of SSE-APH from [1] because SSE-APH is included in SSE and needs to have the same property in terms of confidentiality. Therefore, definition 5 is borrowed from [1].

Definition 2. History [1]. Let D be a collection of n documents and Δ be a dictionary. A history H_q, $H_q \in 2^{2^\Delta} \times \Delta^q$, is an interaction between a client and the document server via the service provider over q queries. The partial history $H_q^t \in 2^{2^\Delta} \times \Delta^t$ of a given history $H_q = (D,w_1,\ldots,w_q)$, is the sequence $H_q^t = (D,w_1, \ldots ,w_t)$, where $t \leq q$.

Definition 3. View [1]. Let D be a collection of n documents, index I for collection D, and $H_q = (D, w_1, \ldots ,w_q)$ be a history over q queries. An adversary's view of H_q under secret key K is defined as $V_K(H_q) = (I, T_{w,1}, \ldots, T_{w,q})$. The partial view $V_K^t(H_q)$ of a history H_q under secret key K is the sequence $V_K^t(H_q) = (D, T_{w,1}, \ldots, T_{w,t})$, where $t \leq q$.

Definition 4. Trace [1]. Let D be a collection of n documents $= (d_1,\ldots,d_n)$, $D(w)$ be a list of documents that contain a w, $E(d_i)$ be an encrypted version of document d_i, and $H_q = (D, w_1,\ldots ,w_q)$ be a history over q queries. The trace of H_q is the sequence $Tr(H_q) = (E(d_1),\ldots,E(d_n), D(w_1),\ldots,D(w_q), |D(w_1)|,\ldots,|D(w_q)|, \Pi_q, \Gamma_q)$ (Π: search pattern, Γ: the number of queried words).

Definition 5 [1]. Adaptive semantic security for SSE. An SSE scheme is adaptively semantically secure if for all $q \in N$ and for all (non-uniform) probabilistic polynomial-time adversaries A, there exists a (non-uniform) probabilistic polynomial-time algorithm (the simulator) S such that for all traces Tr_q of length q, all polynomially samplable distributions H_q over $\{\mathrm{H}_q \in 2^{2^{\Delta}} \times \Delta_q\colon \mathrm{Tr}(\mathrm{H}_q) = \mathrm{Tr}_q\}$ (that is, the set of histories with trace Tr_q), all functions $f\colon \{0, 1\}^m \rightarrow \{0, 1\}^{\mathrm{poly}(m)}$ (where $m = |\mathrm{H}_q|$), all $0 \le t \le q$, and all polynomials p and sufficiently large k:

$$\left|\Pr\left[A(V_K^t(\mathrm{H}_q)) = f(\mathrm{H}_q^t)\right] - \Pr\left[S(\mathrm{Tr}(\mathrm{H}_q^t)) = f(\mathrm{H}_q^t)\right]\right| < \frac{1}{p(k)},$$

where $\mathrm{H}_q \overset{R}{\leftarrow} \mathrm{H}_q$, $K \overset{R}{\leftarrow} \mathrm{KeyShare}(1^k)$, and the probabilities are taken over H_q and over the internal coins of KeyShare, A, S and the underlying BuildIndex algorithm.

Definition 6. Hiding access pattern/hiding frequency. History H_q for q-th queries, two arbitrary trapdoors T_w and T_w' for an arbitrary w, and the search results $R(T_w)$ and $R(T_w')$, for all polynomial-time circuit families $\{A_k\}$, efficiently large k, and polynomial p, the following inequality should hold:

$$\left|\Pr\left[A_k(\mathrm{H}_q, R(T_w)) = 1\right] - \Pr\left[A_k(\mathrm{H}_q, R)T_w')) = 1\right]\right| < \frac{1}{p(k)}.$$

3. Algorithms

We propose a searchable encryption scheme for a single user. These algorithms are the implementations of those shown in definition 1 to make them work in the scenario shown in section III.

A user U stores n encrypted documents in collection $D = (d_1,\dots,d_n)$ at the document server DS and retrieves a specific document with a *document_id*. DS maintains the data table DT that keeps the encrypted documents. The index server IS manages the index table IT generated by U. U keeps a cache named word call cache (WCC). Let $\pi(\bullet)$ be a pseudo-random permutation and $E(\bullet)$ be a semantically secure symmetric encryption scheme. All the algorithms are shared among U, DS, and IS. Tables 1, 2, and 3 show the schemas of IT, DT, and WCC.

A tuple of the IT is assigned to each document in $D(w)$. $D(w)$ is a collection of documents containing the $w \in \Delta'$. Thus, the IT has $\sum_{i=1}^{|\Delta'|} |D(w_i)|$ tuples. The *trapdoor* is generated with a concatenation of a w and an index number i that represents the order in $D(w)$. For example, suppose that $d_{\text{'coin'},1}$ is the first document in D(*'coin'*), the corresponding tuple of IT uses *'coin'* and the order '1' to make the *trapdoor*. For each w, the IT allocates $|D(w)|$ tuples, and the initial values of each tuple's trapdoor are $\pi_{k\mathrm{user}}(w||1),\dots, \pi_{k\mathrm{user}}(w\ ||\ |D(w)|)$ in order.

The *word_counter* keeps the number of documents that contains each word. An attacker might predict their relationship more easily if the tuples that have the same *word_counter* value may contain the same word. Therefore, the number is

Table 1. Schema of IT.

Name	Initial value	Description
trapdoor	$\pi_{k\mathrm{user}}(w\|\|i)$	-Primary key -Permutated concatenation of a w and an index i, where i is index of $D(w)$, $1 \le i \le |D(w)|$
document_id	$\pi_{k\mathrm{user}}(id(d_i))$	Permutated index of a document d_i. d is the i-th document in $D(w)$, $1 \le i \le |D(w)|$. This is the same as the *document_id* of DT.
word_counter	$E_{kw}(id(d_i)\|\|\|D(w)\|)$	Encrypted concatenation of a document d_i and $|D(w)|$. $|D(w)|$ is the number of documents that contain a w.

Table 2. Schema of DT.

Name	Initial value	Description
document_id	$\pi_{k\mathrm{user}}(id(d))$	-Primary key -Permutated identifier of document d with a user's symmetric key k_{user}
document	$E_{k\mathrm{user}}(d)$	An encrypted document d with a user's symmetric key k_{user}

Table 3. Schema of WCC.

Name	Value	Description
word	w	Word queried by a user
round	r	Number of queries corresponding to a w

combined with the index of a document and then encrypted. It indicates the number of documents that contain a word. Therefore, it allows for an efficient sequential search.

WCC is maintained by U. The round items in all tuples are initialized to zero. Whenever a query corresponding to a w is executed, the *round* in the tuple $WCC[w]$ is increased. If a w is used for the first time, then $WCC[w]$ is 1 after the search is finished. As the number of queries about w increases, the *round* value in $WCC[w]$ also increases.

U does not know IS and DS before it stores data at the IS and DS. If the protocol begins, we assume that the IS and DS are authenticated by U. U does not have to be authenticated because there is no reason that the service provider makes U use the servers without being authenticated. Therefore, the service provider certifies the trust of U to the servers. However, we also suppose that the service provider and servers do not collaborate with each other since we assume they do not belong to the same company.

We show the descriptions of the implementations of the algorithms below.

A. KeyShare(1^k)

Let k be security parameters and let (G, E, D) be a semantically secure symmetric encryption scheme with $E : \{0, 1\}^k \times \{0, 1\}^* \rightarrow \{0, 1\}^*$. In addition, we make use of one pseudo-random permutation π with the following parameters:

$$\pi : \{0, 1\}^k \times \{0, 1\}^k \rightarrow \{0, 1\}^k.$$

As we mentioned before in the paper, by using the existing protocol such as tripartite key exchange [26], U, IS, and DS share two keys k_{share} and p_{share}. With both values, a session specific key k^n_{session} for session n is generated by $k^n_{\text{session}} \leftarrow \pi_{pn}(k_{\text{share}})$, where p_n is the result of the n-th consecutive execution of the pseudo random number generator with p_{share} as a random seed. Finally, U generates $k_{\text{user}} \overset{R}{\leftarrow} \{0,1\}^k$. This is only used by U.

B. BuildIndex(K, D)

K denotes the (k_{share}, p_{share}, k_{user}) pair, and D is the entire set of documents. First, U scans D and generates a distinct word list Δ' in which all element words are contained in D. Then, for each $w \in \Delta'$, U generates a list of documents $D(w) \subseteq D$ that contains w. U generates an IT that maintains the list of document indices to be used for searching the DT. Table 1 shows the schema of the IT. Next, U generates a DT to save the documents. Table 2 shows the schema of the DT. From the rest of the algorithm description, we assume the first item in the schema is an index of the table, because it is the primary key. Therefore, we represent a tuple in IT as $IT[\pi_{k\text{user}}(w||i)] = <\pi_{k\text{user}}(id(d_i)), E_{k\text{user}}(id(d_i)|||D(w)|)>$. Similarly, we represent a tuple in DT as $DT[\pi_{k\text{user}}(id(d))] = E_{k\text{user}}(d)$. From the description of the tuples, it is found that the index of DT is stored as the second item in IT. This is the *document_id*. Finally, the *word_counter* is re-encrypted with the same key, and thus the resulting value changes.

C. Trapdoor(K, w, i)

For a given w, a user U generates a trapdoor for the service provider to get the i-th document containing w. If U searches the word for the first time, the trapdoor is $T^1_{w,i} \leftarrow \pi_{k\text{user}}(w||i)$. For the j-th time, $T^j_{w,i} \leftarrow \pi_{kw}(T^{j-1}_{w,i})$, where $k_w \leftarrow \pi_{p\text{share}}(w||0)$. As explained before, *WCC* stores the number of searches for a w. Thus, U can compute $T^j_{w,i}$ from scratch. U can store the trapdoor for the previous search to reduce the amount of computations. In this case, the amount of computations becomes O(1).

D. Search_Index(IT, $T_{w,i}$)

IS uses this algorithm to determine the corresponding index for a given trapdoor $T_{w,i}$. It includes the operations for updating *IT* to support access pattern hiding. Using the trapdoor received, this algorithm returns the corresponding *document_id*. If such a tuple does not exist, then it returns the null message.

Next, the tuple with $T_{w,i}$ as the primary key (index) is updated: the first item, that is, the index, is changed to $\pi_{kw}(T_{w,i})$, where $k_w \leftarrow \pi_{p\text{share}}(w||0)$. The second item, that is, *document_id*, is changed to $\pi_{k'w}$ (*document_id*), where $k'_w \leftarrow \pi_{p\text{share}}(w||1)$.

E. Search_Data(DT, document_id)

DS uses this algorithm to retrieve the document corresponding to the given *document_id* on *DT*. *DS* finds the *document* that is in the same tuple as *document_id*. Then, returns the *document*. If no such *document_id* exists, then it returns a null message. If it does not return a null message, the tuple with the return value in *DT* is updated: the *document_id* is changed to $\pi_{k'w}$ (*document_id*), where $k'_w \leftarrow \pi_{p\text{share}}(w||1)$. In addition, the *document* in the tuple is also re-encrypted, and thus it has a different form. The access hiding property of the proposed scheme is guaranteed due to this procedure.

4. Search Protocol

This subsection shows how U initializes the system with other entities. Then, it also shows how U searches documents with a key $w \in \Delta'$.

A. Initialization Protocol

U, *IS*, and *DS* share k_{share} and p_{share} using the KeyShare(1^k) protocol. This protocol is executed via the service provider. Next, U executes the BuildIndex algorithm with K=(k_{share}, k_{user}, p_{share}) and the set of documents D. The result of BuildIndex *IT* and *DT* are sent to the *IS* and *DS*, respectively. Then, U removes them from its storage.

B. Search Protocol

U runs the Trapdoor(K, w, 1) algorithm to obtain the first trapdoor for w, $T^j_{w,1}$. U sends this to the *IS*. *IS* sends *word_counter* back to the user. If such a word does not exist in *IT*, *IS* sends a dummy ciphertext to U so that the service provider cannot distinguish a case in which no such word exists from one in which it does. Next, U runs Trapdoor(K, w, i), where i=1,..,*word_counter*. Thus, it creates the other trapdoors. Suppose $w_{\max}$ is at the word that most documents have. Then, the other $|D(w_{\max})|$ − *word_counter* dummy trapdoors are generated, and thus the service provider cannot determine the actual number of documents. These dummy messages will produce the dummy indices by *IT*. They are sent to *DT* via the service provider. They also produce many dummy encrypted documents that are transferred back to U.

Regardless, the number of messages is fixed to $|D(w_{\max})|$: the service provider cannot exploit the knowledge about the number of messages transmitted to determine the access pattern.

Except in the case of dummy messages, the correct trapdoors are sent to *IS*, and *IS* gives *DS* the corresponding indices of the documents via the service provider after running the Search_Index(IT, $T_{w,i}$) algorithm for each of the (correct) trapdoors. *DS* returns the corresponding documents back to *U* via the service provider.

V. Analysis

In this section, we analyze the proposed scheme by comparing it to previous works.

1. Security and Privacy

Theorem 1. The proposed SSE-APH is an adaptive secure scheme based on definition 5.

Proof. For proof, we follow the simulation-based approach of definition 5. We describe a probabilistic polynomial-time simulator *S* that has the ability to use the trace of a partial history. We prove that for $0 \le t \le q$, V_q^{t*} is indistinguishable from $V_K^t(\mathrm{H}_q)$, which is the output of simulator $S(\mathrm{Tr}(\mathrm{H}_q^t))$.

$V_K(\mathrm{H}_q) = \{DT, IT, T_1, \ldots, T_q\}$,

$V_K^t(\mathrm{H}_q) = \{DT, IT, T_1, \ldots, T_t\}$, where $0 \le t \le q$,

$\mathrm{Tr}(\mathrm{H}_q) = \{E(d_1), \ldots, E(d_n), D(w_1), \ldots, D(w_q), |D(w_1)|, \ldots, |D(w_q)|, \Pi_q, \Gamma_q\}$,

$\mathrm{Tr}(\mathrm{H}_q^t) = \{E(d_1), \ldots, E(d_n), D(w_1), \ldots, D(w_t), |D(w_1)|, \ldots, |D(w_t)|, \Pi_t, \Gamma_t\}$, where $0 \le t \le q$.

The simulator *S* knows about pseudo-random permutations π and *f*. Now, we generate $V_K^{t*} = \{DT^*, IT^*, T_1^*, \ldots, T_t^*\}$ using simulator *S*.

For t=0, there is no communication between server and client. Simulator *S* generates $V_0^* = \{DT^*, IT^*\}$.

- DT^*: for $0 \le i \le m$, simulator *S* generates a random string $\alpha_i \xleftarrow{R} \{0,1\}^k$ and sets $DT^*[\alpha_i] = E(d_i)$.
- IT^*: for $0 \le i \le m$, simulator *S* generates random strings $\beta_i, \gamma_i, \delta_i \xleftarrow{R} \{0,1\}^k$ and sets $IT^*[\beta_i] = \langle \gamma_i, \delta_i \rangle$.

Simulator *S* knows $E(d_i)$ but does not know its index, which is the output of a pseudo-random permutation with the seed k_{user}. Therefore, simulator *S* sets $E(d_i)$ to random locations in the DT^*. If DT^* and *DT* are distinguishable, it means someone can distinguish between the output of a pseudorandom permutation and a random string. IT^* is also indistinguishable from *IT*. The reason for this is that *IT* is constructed based on a semantically secure encryption scheme, and is the output of a pseudorandom permutation with seed k_{user}. If IT* and *IT* are distinguishable, someone can distinguish between the output of pseudorandom permutation and a random string, or between the output of a semantically secure ciphertext and a random string.

For $1 \le t \le q$, simulator *S* generates V_K^{t*}. Simulator *S* is adaptive, so it can use V_0^* and V_K^{t-1*}. Let simulator *S* generate a trapdoor for w_t. Then, simulator *S* checks if the trapdoor for w_t has been used before through Π_{t-1}. If the trapdoor has not been used before, simulator *S* generates a random address $addr_\tau$ and sets the $IT^*[addr_\tau] = \langle \gamma_i, \delta_i \rangle$ and the $T_t^* = addr_\tau$. Otherwise, simulator *S* obtains information regarding about how many queries corresponding to w_t have been asked through Γ_t. Let us call this *c*. Now, simulator *S* executes step 3 of the BuildIndex algorithm using *c* instead of $WCC[w_t]$. However, simulator *S* does not know about k_{user}, k_{share}, and p_{share}, and thus there is no choice except to use a random string. This is the same for DT^*. Therefore, we can say that V_q^{t*} and $V_K^t(H_q)$ are indistinguishable by definition 5. □

Theorem 2. The proposed scheme hides access pattern/hiding frequency.

Proof. For every w_i in *IT*, let the number of queries be t_i and the document index returned is then $E_{ks}^{t-1}(\pi_{kc}(id(D(w_i))))$. The size of the output of the encryption scheme $E{:}\{0,1\}^* \times \{0,1\}^k \rightarrow \{0,1\}^*$ is a fixed number, say *r*. Thus, the range of the domain is 2^r. Therefore, two outputs from one word would be distinguishable with probability $1/2^r$. □

For each trapdoor and *document_id* used, the proposed scheme permutes them by using a shared key between *U* and server *S*. The access pattern is not revealed with the history of trapdoors and *document_id*s because every trapdoor and *document_id* is unique. According to the definition of the random permutation, every trapdoor and *document_id* is used at most once and is indistinguishable from one another. In addition, for each session, the different seed value of the permutation is allocated.

Even for the same *w*, generated trapdoors $T_{w,1}, \ldots, T_{w,|D(w)|}$ differ from each other. Therefore, like a coin toss, an adaptive adversary cannot distinguish if two trapdoors are generated from the same word.

2. Performance

We quote the performance of the previous works from [1]. Table 4 shows the performance of the proposed scheme. We deduce each result from the procedure for generating trapdoors and searching documents for a word. *n* is the number of documents in document collection *D* and is the standard of the time complex. The server computation is the cost per returned document. The number of rounds only considers the interaction rounds between a server and user. The communication cost only considers a message's overhead, which is transmitted from the user to the server. That is, we omit the size of

Table 4. Properties and performances (per query) of various SSE schemes (n: the number documents stored in *DS*).

Properties	Proposed scheme	[7]	[2]	SSE-1 [1]	SSE-2 [1]	[3]
Access pattern	Y	Y	N	N	N	N
Adaptive adversaries	Y	Y	N	N	Y	N
Server computation	O(1)	$O(\log^3 n)$	$O(n)$	O(1)	O(1)	O(1)
Server storage	$O(n)$	$O(n*\log^2 n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Number of rounds	O(1)	O(log$n$)	2	1	1	O(1)
Communication	O(1)	$O(\log^3 n)$	O(1)	O(1)	O(1)	O(1)

extracted documents that the server returns. The server storage includes the stored database, table, and keys.

A. Server Storage

DS stores n encrypted documents in the *DT* for the proposed scheme. In *IT*, for distinct words Δ' that contain n documents, the server allocates $|D(w)|$-th tuples for each $w \in \Delta'$. That is, *IT* has $\sum_{i=1}^{|\Delta'|} |D(w_i)|$ tuples. In addition, when the servers establish a session for a user, the storage requirement is increased two-fold. They replicate the user's *DT* and *IT*. Therefore, the storage overhead is $2 \times (n + \sum_{i=1}^{|\Delta'|} |D(W_i)|) = O(n)$, which is the same as in [1].

B. Number of Rounds

In the proposed scheme, a user generates a trapdoor $T_{w,1}$ and query to the servers for a w. From the servers' response, the user obtains a *document_id* of the first document and the number of entire documents $|D(w)|$ that contain the w. If $|D(w)| > 1$, the user proceeds with the search further by an extra of $|D(w)| - 1$ rounds. We do not include the search of *DT* with *document_ids*. Namely, the proposed scheme requires $O(|D(w)|) = O(1)$ rounds.

C. Communication

The trapdoor of w, $T_{w,i}$, is $\pi_{kw}{}^{j}(\pi_{kuser}(w||i))$, where $1 \le i \le |D(w)|$, $0 \le j \le t$, and t is the number of searches for w. Due to the random permutation $\pi : \{0, 1\}^k \times \{0, 1\}^k \rightarrow \{0, 1\}^k$, the size of a message for the server is $O(k) = O(1)$ because k is less than a few hundred and is a fixed number.

D. Server Computation

For w, *IS* searches the *IT* with T_w and retrieves a corresponding tuple. It then permutes the tuple. A trapdoor and *document_id* in the *IT* are permuted with shared keys k_{share} and p_{share}. For *word_counter*, the servers encrypt the *word_counter* with the shared key k_{session}, which can be computed with k_{share} and p_{share}. The above process occurs for each query, and the amount of server computations is O(1).

VI. Conclusion

This paper proposed a privacy-preserving database encryption scheme. The proposed scheme shares a session key between the user and server for the documents encrypted by a user's private key. They use the key to permute an index of a database tuple once accessed. The user maintains the word call cache *WCC*, and references it to permute a trapdoor or to decrypt a *word_counter*. The server permutes a tuple's value with the session key for each search. A permuted tuple is matched to a trapdoor. The server deletes the original tuple and creates a new one. This process requires only O(1). The proposed scheme hides the access pattern for each access.

References

[1] R. Curtmola et al., "Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions," *13th ACM Conf. Comput. Commun. Security*, 2006.

[2] Y.C. Chang and M. Mitzenmacher, "Privacy Preserving Keyword Searches on Remote Encrypted Data," *Applied Cryptography Netw. Security Conf.*, 2005.

[3] Z. Yang, S. Zhong, and R. Wright, "Privacy-Preserving Queries on Encrypted Data," *11th European Symposium Research in Security*, 2006.

[4] M. Kantarcıoglu and C. Clifton, "Security Issues in Querying Encrypted Data," Purdue Computer Science Technical Report 04-013, 2004.

[5] M. Abdalla et al., "Searchable Encryption Revisited: Consistency Properties, Relation to Anonymous IBE, and Extensions," *Crypto*, 2005.

[6] D. Song, D. Wagner, and A. Perrig, "Practical Techniques for Searches on Encrypted Data," *Proc. IEEE Symp. Security Privacy*, 2000, pp. 44-55.

[7] O. Goldreich and R. Ostrovsky, "Software Protection and

Simulation on Oblivious RAMs," *J. ACM*, vol. 43, no. 3, 1996, pp. 431-473.

[8] B. Chor et al., "Private Information Retrieval," *J. ACM*, vol. 45, no. 6, 1998, pp. 965-982.

[9] S.W. Smith and D. Safford, "Practical Private Information Retrieval with Secure Coprocessors," IBM Research Report, RC 21806, 2000.

[10] R. Ostrovsky and W.E. Skeith, "A Survey of Single-Database Private Information Retrieval: Techniques and Applications," *LNCS,* vol. 4450, 2007, pp. 393-411.

[11] Q. Chai and G. Gondm, "Verifiable Symmetric Searchable Encryption for Semi-Honest-but-Curious Cloud Servers." http://www.cacr.math.uwaterloo.ca/techreports/2011/cacr2011-22.pdf

[12] E. Shi et al., "Multi-Dimensional Range Query over Encrypted Data," *Proc. IEEE Symp. Security Privacy*, 2007, pp. 350-364.

[13] H. Pang and K.L. Tan, "Verifying Completeness of Relational Query Answers from Online Servers," *ACM Trans. Inf. Syst. Security*, vol. 11, no. 2, article 9, May 2008.

[14] M. Abdalla et al., "Searchable Encryption Revisited: Consistency Properties, Relation to Anonymous IBE, End Extensions," *Proc. Adv. Cryptology, LNCS,* vol. 3621, 2005, pp. 205-222.

[15] J. Bethencourt, D. Song, and B. Waters, "New Techniques for Private Stream Searching," *ACM Trans. Inf. Syst. Security*, vol. 12, no. 3, article 16, Jan. 2009.

[16] P. Paillier, "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes," *Proc. Adv. Cryptology: EUROCRYPT, LNCS*, vol. 1592, 1999, pp. 232-238.

[17] H. Pang, J. Shen, and R. Krishnan, "Privacy-Preserving Similarity-Based Text Retrieval," *ACM Trans. Internet Technol.*, vol. 10, no. 1, article 4, Feb. 2010.

[18] C. Wang et al., "Secure Ranked Keyword Search over Encrypted Cloud Data," *Proc. Int. Conf. Distrib. Comput. Syst.*, 2010, pp. 253-262.

[19] N. Cao et al., "Privacy-Preserving Multi-keyword Ranked Search over Encrypted Cloud Data," *Proc. IEEE INFOCOM*, 2011, pp. 829-837.

[20] A. Boldyreva et al., "Order-Preserving Symmetric Encryption," *Proc. Eurocrypt, LNCS*, vol. 5479, 2009, pp. 224-241.

[21] H. Pang, J. Zhang, and K. Mouratidis, "Scalable Verification for Outsourced Dynamic Databases," *Proc. 35th VLDB Conf.*, Aug. 2009, pp. 802-813.

[22] S. Papadopoulos, W. Cheng, and K.L. Tan, "Separating Authentication from Query Execution in Outsourced Databases," *Proc. 25th Int. Conf. Data Eng.*, Apr. 2009, pp. 1148-1151.

[23] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers," *Proc. Adv. Cryptology: CRYPTO*, 2010, pp. 465-482.

[24] C. Gentry, "Fully Homomorphic Encryption Using Ideal Lattices," *Proc. 41st Annual ACM Symp. Theory Comput.*, 2009, pp. 169-178.

[25] M. Dijk et al., "Fully Homomorphic Encryption over the Integers," *Proc. Adv. Cryptology: EUROCRYPT*, 2010, pp. 24-43.

[26] A. Joux, "A One Round Protocol for Tripartite Diffie-Hellman," *Proc. Algorithmic Number Theory, LNCS*, vol. 1838, 2000, pp. 385-393.

Daeseon Choi received the BS in computer engineering from Dongguk University, Rep. of Korea, in 1995, the MS in computer engineering from POSTECH, Rep. of Korea in 1997, and the PhD in computer science from KAIST, Rep. of Korea in 2009. Since 1999, he has been a member of research staff with ETRI, Daejeon, Rep. of Korea. His main research areas are ID management, mobile security, and personalization service.

Seung-Hyun Kim received the BS in computer engineering from Kumoh National University of Technology, Rep. of Korea, in 2002, and the MS in computer engineering from POSTECH, Rep. of Korea, in 2004. Since 2004, he has been a member of research staff with ETRI, Daejeon, Rep. of Korea. His main research areas are ID management, mobile security, and personalization service.

Younho Lee received the BE, MS, and PhD in computer science from KAIST, Rep. of Korea, in 2000, 2002, and 2006, respectively. He worked as a visiting postdoctoral researcher and as a member of research staff at the Georgia Tech Information Security Center from 2007 to 2009. He is currently an assistant professor in the Department of Information and Communication Engineering, Yeungnam University, Rep. of Korea. His research interests include network security, applied cryptography, and multimedia security.