Symmetric Searchable Encryption with Efficient Conjunctive Keyword Search

Nam-Su Jho[1], and Dowon Hong[2]
[1] Electronics and Telecommunications Research Institute
138 Gajeongno, Yuseong-gu, Daejeon, 305-700, Korea
[e-mail: nsjho@etri.re.kr]
[2] Department of Applied Mathematics, Kongju National University
56 Gongjudaehak-ro, Gongju-si, Chungcheongnam-do, 314-701, Korea
[e-mail: dwhong@kongju.ac.kr]
*Corresponding author: Dowon Hong

Received December 11, 2012; revised March 7, 2013; revised April 16, 2013; accepted May 6, 2013; published May 31, 2013

Abstract

Searchable encryption is a cryptographic protocol for searching a document in encrypted databases. A simple searchable encryption protocol, which is capable of using only one keyword at one time, is very limited and cannot satisfy demands of various applications. Thus, designing a searchable encryption with useful additional functions, for example, conjunctive keyword search, is one of the most important goals. There have been many attempts to construct a searchable encryption with conjunctive keyword search. However, most of the previously proposed protocols are based on public-key cryptosystems which require a large amount of computational cost. Moreover, the amount of computation in search procedure depends on the number of documents stored in the database. These previously proposed protocols are not suitable for extremely large data sets.

In this paper, we propose a new searchable encryption protocol with a conjunctive keyword search based on a linked tree structure instead of public-key based techniques. The protocol requires a remarkably small computational cost, particularly when applied to extremely large databases. Actually, the amount of computation in search procedure depends on the number of documents matched to the query, instead of the size of the entire database.

Keywords: Searchable encryption, database encryption, data privacy, conjunctive keyword search

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (Grant No. 2011-0029925). This work was also supported by the K-SCARF project, the ICT R&D program of ETRI(Research on Key Leakage Analysis and Response Technologies).

http://dx.doi.org/10.3837/tiis.2013.05.022

1. Introduction

As the amount of data is greatly increasing, methods for storing and managing data efficiently are attracting more interest and many solutions have been developed. The use of a remote database service is the most common and convenient method to manage huge data sets. However, storing sensitive data in a remote database in which the database manager is different from the data owner can cause many side effects. In particular, a privacy breach of the stored data is the most important issue. Remote database service providers have adopted various means such as user authentication or access control to maintain the privacy of the stored data. However, these are not fundamental solutions, as a database service provider can easily access the stored data. Another way to maintain the privacy of the stored data is to apply an encryption system. Well developed and designed encryption systems guarantee the secrecy of encrypted data theoretically. Adopting encryption system creates another problem, that is, encryption systems conceal too much information, thereby preventing useful database operations such as searching or sorting. Searchable encryption is proposed for solving this problem, allowing an efficient search of encrypted documents and maintaining the secrecy like encryption systems.

A method to search data while maintaining privacy was first researched by Ostrovsky and Goldreigh [1], [2]. The result of their work, oblivious RAMs, has different characteristics from searchable encryption systems. However, oblivious RAMs became the basis of searchable encryption systems in later research. In 2000, Song *et al*. introduced a formal definition of the searchable encryption and proposed several solutions [3]. Through many researches such as [4]-[6] the basic structure of searchable encryption was developed and formalized so that additional information, called an *index*, is stored in the server along with encrypted documents. Goh also introduced notions of security for searchable encryption focusing on an index: non-adaptive indistinguishability security against a chosen keyword attack (IND-CKA) and adaptive indistinguishability security against a chosen keyword attack (IND2-CKA) [4]. In 2006, Curtmola *et al*. revised the IND2-CKA security along with the first searchable encryption using a linked chain structure in which the searching time is independent of the number of total encrypted documents [7]. This formal structure and these security notions have been widely adopted in recent studies. There have also been attempts at constructing searchable encryption that supports public-key encryption, i.e., anyone can provide encrypted documents and indexes using a public-key, where searching and decryption is only available for the owner of a secret key [8], [9].

Basic searchable encryption provides a search method that finds documents corresponding to a single keyword. However, this basic search method is very limited and cannot satisfy the various demands that naturally arise. Therefore, designing a searchable encryption with additional search functions is one of the most important goals. Additional search functions that are frequently remarked upon include conjunctive keyword search, range query, ordering, and size comparison. In this paper, we concentrate on the conjunctive keyword search problem.

In conjunctive keyword search a query consists of multiple keywords for search, and a search procedure results list of documents which are matched to all of the keywords without revealing any other information. A conjunctive keyword search is frequently used for searching documents in everyday life. By using a basic keyword search algorithm repetitively, one can obtain similar results for conjunctive keyword search, i.e. the list of documents that contain all keywords, for a queried keywords $\{w_1, w_2, \ldots, w_\delta\}$. However, this method is

inefficient, and the secrecy of searched data is also breached. If a server performs a basic search method for each single keyword in $\{w_1, w_2, \ldots, w_\delta\}$, then the server obtains $\mathcal{D}(w_1)$, $\mathcal{D}(w_2)$, …, $\mathcal{D}(w_\delta)$ separately, where $\mathcal{D}(w_i)$ means the set of documents including the keyword w_i, and obtains the final result by computing $\mathcal{D}(w_1) \cap \mathcal{D}(w_2) \cap \ldots \cap \mathcal{D}(w_\delta)$. In this process, the server has to perform δ independent processes separately, which is very inefficient. Moreover, the server can obtain information for intermediate results, $\mathcal{D}(w_1)$, $\mathcal{D}(w_2)$, …, $\mathcal{D}(w_\delta)$.

Searchable encryption providing a conjunctive keyword search has been mainly constructed using techniques based on public-key cryptosystems, especially a bilinear map over elliptic curves [10]-[13]. In addition, the amount of computation which is required in the search procedure is proportional to the number of stored documents. This means that millions of public-key based decryptions or bilinear map computations can be required for just one query. Therefore, we focus on constructing a searchable encryption system with conjunctive keyword search using symmetric encryption system only. Our protocol provides remarkably efficient search algorithm.

2. Preliminaries

2.1 Searchable Symmetric Encryption Using a Linked Chain

In 2006, Curtmola *et al*. proposed a searchable symmetric encryption (SSE) using a linked chain structure along with security definitions for SSE [7]. We present brief introduction of the protocol proposed by Curtmola. The protocol consists of four algorithms, **KeyGeneration**, **BuildIndex**, **Trapdoor**, and **Search**.

KeyGeneration

1. The user prepares a symmetric encryption system, E, which is proven to be secure. From now on, $E_k(m)$ represents a cipher-text of a message m using a secret key k.
2. The user also selects two cryptographically secure one-way (pseudorandom) functions, f and h, which are kept secret.

BuildIndex ($\mathcal{D}$, E, k, f, h)

1. Initialization
   - A. Scan $\mathcal{D}$, the set of all documents, to build Δ', the set of distinct keywords in $\mathcal{D}$.
   - B. For each keyword $w \in \Delta'$, build $\mathcal{D}(w)$, the set of documents that contains the keyword w.
2. Build array A
   - A. For each $w_i \in \Delta'$, build a linked list, L_i
     - i. compute $v_{i,0} = f(w_i)$ and $k_{i,0} = h(w_i)$
     - ii. for $1 \le j \le |\mathcal{D}(w_i)|$
       1. randomly generate $v_{i,j}$ and $k_{i,j}$
       2. set a node $N_{i,j} = \langle\ \mathrm{id}(D_{i,j}) \parallel v_{i,j} \parallel k_{i,j}\ \rangle$, where $\mathrm{id}(D_{i,j})$ is the identifier of the j-th document in $\mathcal{D}(w_i)$
       3. compute $E_{k_{i,j-1}}(N_{i,j})$, and store it in $A[v_{i,j-1}]$

iii. for the last node of L_i (i.e., $N_{i,|\mathcal{D}(w_i)|}$), before encryption, set the address of the next node to 'empty'

B. set the remaining entries of A to random values of the same size as the existing entries of A

3. Output the index A

Trapdoor(w) : Output $T_w = (h(w), f(w))$

Search(A, $T_w = (h(w), f(w))$)

1. Decrypt the linked list L starting with the node at address v_w encrypted under key k_w
2. Output the list of document identifiers contained in L

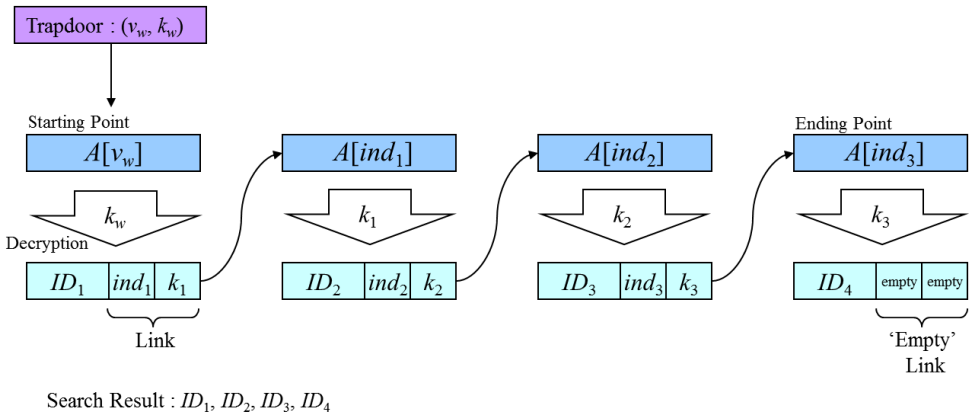

Fig. 1. Linked chain structure

3.2 Security Definition

We also present the security notions proposed by Curtmola [7].

A *history* is an interaction between a user and a server. If we assume that a user makes q queries, then the history, $H^{(q)}$, can be uniquely determined by the collection of documents, which is stored at the server, and q queries. Thus, we can define a history as $H^{(q)} = (\mathcal{D}, w_1, \ldots, w_q)$, where $\mathcal{D}$ is a collection of all documents, and each w_i is a queried keyword.

Intuitively, the history is what the user wants to hide from the server. However, the server knows identifiers of documents, encrypted documents, and the index. During interaction, the server obtains more information such as trapdoors, the number of documents matched with the given trapdoors. A set of this information, i.e., what an adversary actually gets to "see", is called a *view* of the given history. Formally, we can define a view of history $H^{(q)}$, as $V(H^{(q)}) = (\mathrm{id}(D_1), \ldots, \mathrm{id}(D_N), E(D_1), \ldots, E(D_N), I_{\mathcal{D}}, T_1, \ldots, T_q)$, where $E(D)$ indicates a ciphertext of D, $I_{\mathcal{D}}$ is an index, and T_i is a trapdoor.

A *trace* means the information about a history, which is leaked. The outcome of each search procedure, a collection of identifiers of documents that contain each queried keywords,

is leaked to the server. The pattern of searches, the information of correspondence between two trapdoors, is also revealed to the server. Moreover, since the encrypted documents are stored on the server, we can assume that the information about these encrypted documents and their identifiers are also leaked. Formally, the trace, $Tr(H^{(q)})$, can be defined as $Tr(H^{(q)}) = (\mathrm{id}(D_1), \ldots, \mathrm{id}(D_N), |D_1|, \ldots, |D_N|, \mathcal{D}(w_1), \ldots, \mathcal{D}(w_q), \Pi^{(q)})$, where $\Pi^{(q)}$ indicates the user's search pattern, $\Pi^{(q)}[i, j] = 1$ if $w_i = w_j$; otherwise, $\Pi^{(q)}[i, j] = 0$ for $1 \le i,j \le q$. Substituting a single keyword, w_i, into a keyword set, s_i, we can adopt these definitions for a conjunctive keyword search. The pattern of search $\Pi^{(q)}$ is also changed to $\Pi^{(q)}[i, j] = 1$ if $s_i \subseteq s_j$; otherwise, $\Pi^{(q)}[i, j] = 0$ for $1 \le i, j \le q$.

Definition (Non-Adaptive Indistinguishability Security for SSE) An SSE scheme is secure in the sense of non-adaptive indistinguishability if for all q, all (non-uniform) probabilistic polynomial-time adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, all polynomials p, and all sufficiently large k,

$$\Pr[b' = b \mid (H_0^{(q)}, H_1^{(q)}, \mathrm{state}) \leftarrow \mathcal{A}_1; b \leftarrow_R \{0,1\}; b' \leftarrow \mathcal{A}_2(V(H_b^{(q)}), \mathrm{state})] < \frac{1}{2} + \frac{1}{p(k)},$$

where $(H_0^{(q)}, H_1^{(q)})$ are histories over q queries such that $Tr(H_0^{(q)}) = Tr(H_1^{(q)})$, the 'state' is a polynomially bounded string that captures $\mathcal{A}_1$'s state, and the probability is taken over the internal coins of $\mathcal{A}$ and the underlying *BuildIndex* algorithm.

The searchable encryption protocol presented in the previous section satisfies non-adaptive indistinguishability security.

3. Construction

3.1 Basic Idea

Before describing the formal construction, we introduce the basic ideas utilized for constructing conjunctive keyword search from the searchable encryption of Curtmola.

Among the keyword sets (i.e., conjunctive keyword queries), there are inclusion relations. For example, if there are two keyword sets s_1 and s_2 satisfying $s_1 \subseteq s_2$, then $\mathcal{D}(s_1) \supseteq \mathcal{D}(s_2)$, where $\mathcal{D}(s_i)$ indicates the set of documents corresponding to the keyword set s_i. In this case, $\mathcal{D}(s_1)$ can be considered the union of two sub-sets: one sub-set is $\mathcal{D}(s_2)$, which is included in both results for queries s_1 and s_2, and the other sub-set is the differential '$\mathcal{D}(s_1) - \mathcal{D}(s_2)$', which is included in the result for query s_1 only. Therefore, all documents in $\mathcal{D}(s_2)$ should be included in two linked chains. Instead, we can construct two chains, one is for $\mathcal{D}(s_2)$ and the other is for $\mathcal{D}(s_1) - \mathcal{D}(s_2)$, and create a link between the chains (which we call an external link). For the query s_1, the searching procedure starts from the chain corresponding to $\mathcal{D}(s_1) - \mathcal{D}(s_2)$, and continues following the external link to the chain corresponding to $\mathcal{D}(s_2)$. Thus, the search algorithm can cover all documents of $\mathcal{D}(s_1)$. Note that searching for query s_2 is trivial. Similarly, we can deal with the keyword sets with the relation $s_1 \subset s_2 \subset s_3 \subset \ldots$ by connecting the corresponding linked chains sequentially.

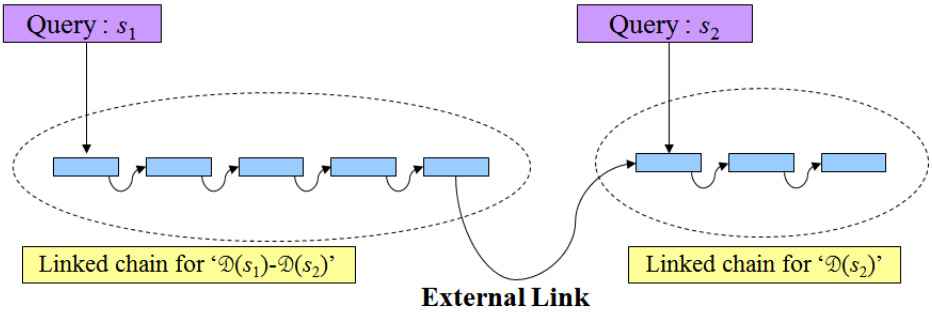

Fig. 2. External link

However, a real conjunctive keyword search is more complicated. In real document sets, incursion relations among keyword sets are not linear, but rather make up a kind of net. Thus, each linked chain requires many external links. In linked chain structure, each linked chain has only one empty end point, and it is too small to cover the required external links. To solve this problem, we expanded the linked chain structure into a linked tree structure. The expansion is easily accomplished by adding one additional link to each node.

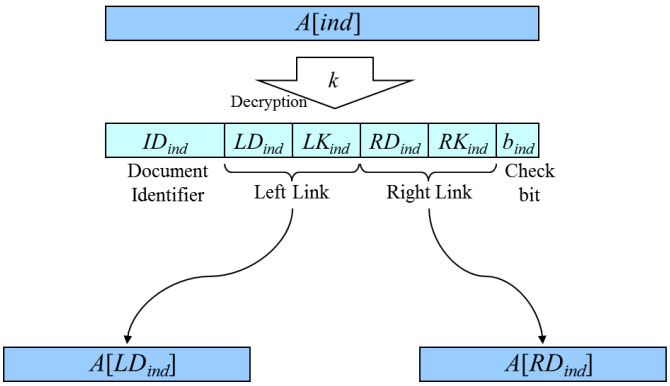

Fig. 3. Expansion into a linked tree structure

Note that when each linked chain is rearranged into a linked tree structure, each linked tree needs not be balanced. Instead, each node is added to the tree at a randomly chosen position. Finally, we can easily manage any conjunctive keyword search connecting external links between linked trees

3.2 Formal Description

Searchable symmetric encryption consists of four algorithms, *key generation*, *build index*, *trapdoor*, and *search*. Without loss of generality, we assume that data set $\mathcal{D}$ consists of N documents, and each document is an ordered tuple of ℓ keywords, $D_i = (w_{i,1}, w_{i,2}, \ldots, w_{i,\ell})$.

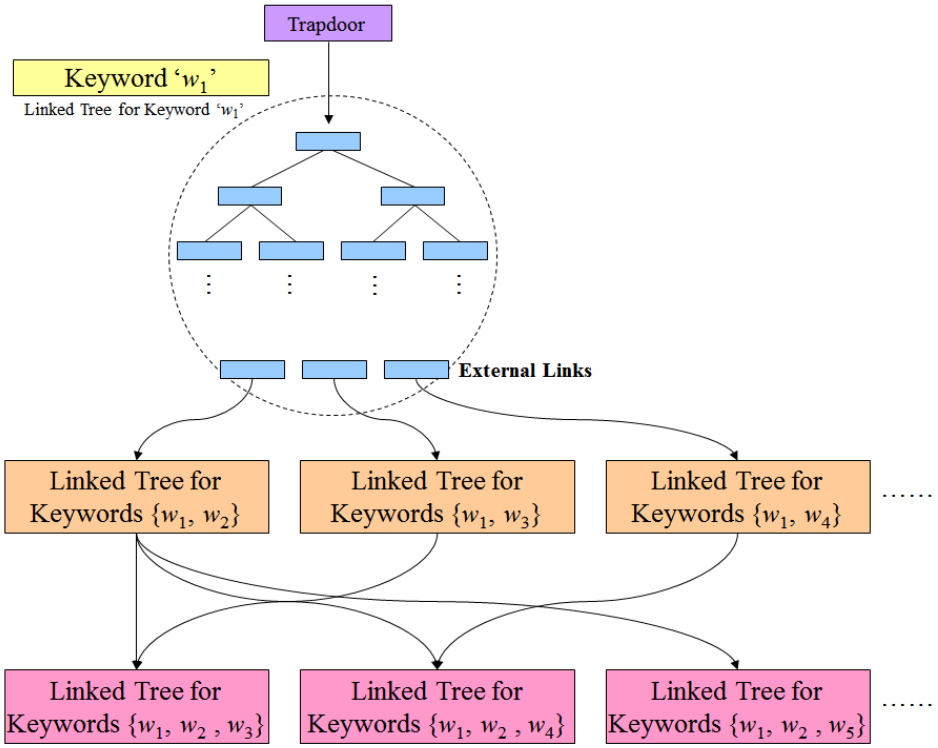

Fig. 4. Conjunctive search using a linked tree structure

Key Generation

There are two security parameters, λ and μ, where λ represents the size of the secret key, and μ is a constant determined from the size of the data set to be encrypted. The user prepares a symmetric encryption system E which uses λ-bit encryption keys. The user also selects a λ-bit secret key k and two one-way (pseudorandom) functions $f:\{0,1\}^* \rightarrow [1,\mu]$ and $h : [1,\mu] \rightarrow \{0,1\}^\lambda$. k, f, and h are kept secret.

Build Index

Build index ($\mathcal{D}$, E, k, f, h) outputs an array A which consists of μ elements. Each element of A is the ciphertext of a tuple,

$$A[ind] = E(ID_{ind} \,||\, \langle\, LD_{ind} ; LK_{ind} \,\rangle \,||\, \langle\, RD_{ind} ; RK_{ind} \,\rangle || \, b_{ind}).$$

ID_{ind} is a document identifier. $\langle\, LD_{ind} ; LK_{ind} \,\rangle$ and $\langle\, RD_{ind} ; RK_{ind} \,\rangle$ are left and right links, respectively. Thus, each element can perform the role of a parent node with two children. LD_{ind} and RD_{ind} represent addresses in array A, LK_{ind} and RK_{ind} are λ-bit keys used in performing encryption E. Finally, b_{ind} is one-bit information.

Build index ($\mathcal{D}$, E, k, f, h)

1. Preparation
   A. For each document $D_i \in \mathcal{D}$
      i. Build Δ_i, the set of all distinct keywords in D_i

ii. Define $\mathcal{P}_i := \left\{ \alpha \mid \alpha \in 2^{\Delta_i}, |\alpha| \le \ell \right\}$, i.e., $\mathcal{P}_i$ is the power set of Δ_i, or the set of all conjunctive keywords in D_i

B. Define $\mathcal{P} = \cup_{1 \le i \le N} \mathcal{P}_i$, so that $\mathcal{P}$ is the set of all possible (conjunctive) queries which has at least one corresponding document in $\mathcal{D}$

2. Build Index A

A. Initialization

i. Make an array A with μ elements.

ii. Each element of A is a tuple of following form

$$A[ind] = (ID_{ind}, \langle LD_{ind} ; LK_{ind} \rangle, \langle RD_{ind} ; RK_{ind} \rangle, b_{ind}).$$

iii. Initially, each ID_{ind}, LD_{ind}, and RD_{ind} are set to '*EMPTY*' and b_{ind} is set to 0, where *EMPTY* indicates a special pre-assigned character.

B. Marking a starting point for each $s_i \in \mathcal{P}$. Note that s_i is a set of keywords, i.e., $s_i = \{w'_{i_1}, w'_{i_2}, ...\}$. We can re-arrange s_i as an ordered tuple $\bar{s}_i = (w_{i,1}, w_{i,2}, \ldots, w_{i,\ell})$, where $w_{i,j}$ may represent a special keyword '*NONE*'.

i. Compute $ind(\bar{s}_i) = f(w_{i,1} \| w_{i,2} \| \ldots \| w_{i,\ell})$.

ii. Find the node $A[ind(i)]$ and set $b_{ind(i)}$ to be 1.

C. Separate $\mathcal{D}$ into disjoint subsets

i. Define $\overline{\mathcal{D}}(s_i)$ be $\mathcal{D}(s_i) - \bigcup_{s_i \subset s_j, s_j \in \mathcal{P}} \mathcal{D}(s_j)$, i.e. the set of all documents those are exactly related to s_i.

ii. So that $\mathcal{D} = \cup \overline{\mathcal{D}}(s_i)$, and $\overline{\mathcal{D}}(s_i) \cap \overline{\mathcal{D}}(s_j) = \emptyset$ for any $i \neq j$.

D. Storing documents (for each $s_i \in \mathcal{P}$)

i. Choose $|\overline{\mathcal{D}}(s_i)|$-1 empty nodes (the node satisfying the condition of $b_{ind} = 0$) in A randomly.

ii. Construct a linked tree over $|\overline{\mathcal{D}}(s_i)|$ nodes. Note that the root node should be the starting point, $A[ind(\bar{s}_i)]$, selected at the step (B). And the other $|\overline{\mathcal{D}}(s_i)|$-1 nodes selected at the previous step are randomly arranged.

- Constructing tree structure is simply storing address of child node at LD_{ind} or RD_{ind} of the parent node.
- If $A[ind_1]$ and $A[ind_2]$ are two children of $A[ind_3]$, then the value of ind_1 and ind_2 are stored at LD_{ind_3} and RD_{ind_3}, respectively. And LK_{ind_3} and RK_{ind_3} are filled with λ-bit encryption keys, $h(ind_1)$ and $h(ind_2)$.
- If $A[ind_3]$ has only one child node, then one of $\langle LD_{ind_3}; LK_{ind_3} \rangle$ and $\langle RD_{ind_3}; RK_{ind_3} \rangle$ is remained *EMPTY*, i.e. empty link.
- If $A[ind_3]$ is a leaf node then it has two empty links.

iii. Store identifiers of documents in $\overline{\mathcal{D}}(s_i)$ at nodes in the tree, Each node stores one identifier.

iv. We denote the tree as $\mathcal{T}(s_i)$.

E. Connecting linked trees (for each keyword set $s_i \in \mathcal{P}$)
   i. For each $s_j \in \mathcal{P}$ satisfying the conditions, $s_i \subset s_j$ and $|s_i| + 1 = |s_j|$, construct an external link from $\mathcal{T}(s_i)$ to $\mathcal{T}(s_j)$ as follows.
   ii. Randomly select a node $A[x]$ from $\mathcal{T}(s_i)$ which has at least one empty link, i.e., $LD_x = EMPTY$ or $RD_x = EMPTY$ (here, we assume that the left link is empty). Then store $ind(\bar{s}_j)$ and $h(ind(\bar{s}_j))$ in LD_x and LK_x, respectively.
   iii. If there is no node with an empty link in $\mathcal{T}(s_i)$, then extend the linked tree as in step (F).

F. Extending a linked tree
   i. Randomly choose a node from $\mathcal{T}(s_i)$ and an empty node from A, say $A[x]$ and $A[y]$, respectively.
   ii. Move one link of $A[x]$ to $A[y]$, i.e., the values of LD_x and LK_x are stored at LD_y and LK_y, respectively (here, we may select the right link instead).
   iii. y, the address of $A[y]$ is stored at LD_x and $h(y)$ is stored at LK_x.
   iv. ID_y is filled with a dummy identifier that cannot be distinguished by anyone except the data owner.
   v. Insert $A[y]$ into $\mathcal{T}(s_i)$, then $\mathcal{T}(s_i)$ has a node with an empty link for an external link.

G. Fill remaining nodes
   i. Fill remaining nodes of A, which are still empty, with a random bit string while keeping the value of b_{ind} as 0.
   ii. For every i, if ID_i is *empty* then fill ID_i with a random bit string.

H. Encryption
   i. Encrypt each node $A[i]$ using the encryption key $h(i)$.

3. Output A

Trapdoor

To make a query about keywords $w_1, w_2, \ldots, w_\ell$ conjunctively, the user computes the following:

$$\text{Trapdoor}(w_1, w_2, \ldots, w_\ell) = \langle f(w_1 \| w_2 \| \ldots \| w_\ell), h(f(w_1 \| w_2 \| \ldots \| w_\ell)) \rangle.$$

Here, w_i may have a value '*NONE*'.

Search

The search process is simply collecting document identifiers in the linked tree starting from the node corresponding to the given trapdoor.

Search(A, $\langle v;k \rangle$)

1. Initialization
   A. Generate set of links $L = \{ \langle v;k \rangle \}$, and set of result $R = \emptyset$.
2. Search
   A. Randomly select a link $\langle \alpha;\beta \rangle$ from L and remove it from L.

B. Decrypt the node $A[a]$,

$$D_\beta(A[\alpha]) = (ID_\alpha, \langle LD_\alpha ; LK_\alpha \rangle, \langle RD_\alpha ; RK_\alpha \rangle, b_\alpha).$$

C. If $b_\alpha = 0$, then terminate the search process and return '*EMPTY*'.

D. Otherwise, two links $\langle LD_\alpha ; LK_\alpha \rangle$ and $\langle RD_\alpha ; RK_\alpha \rangle$ are inserted into L, and ID_α is inserted into R, respectively

E. Iterate the above procedures until L is empty.

3. Output R

4. Analysis

4.1 Correctness

For a keyword set $\bar{s}_1 = \{w_{1,1}, w_{1,2}, \ldots, w_{1,\ell}\}$, let $\mathcal{T}(s_1)$ be the linked tree previously defined. It is clear that for any node in $\mathcal{T}(s_1)$, there is a linked chain from the root node, $A[f(w_{1,1} \| w_{1,2} \| \ldots \| w_{1,\ell})]$. Therefore, with a trapdoor, $\langle f(w_{1,1} \| w_{1,2} \| \ldots \| w_{1,\ell}), h(f(w_{1,1} \| w_{1,2} \| \ldots \| w_{1,\ell})) \rangle$, the user can receive all documents contained in $\mathcal{T}(s_1)$. However, there are more documents corresponding to the query, s_1.

For any keyword set s_2 satisfying $s_1 \subset s_2$, there exists at least one increasing chain, $s_1 = s'_1 \subset s'_2 \subset \ldots \subset s'_\delta = s_2$, with $|s'_i| + 1 = |s'_{i+1}|$. This means that there is an external link from $\mathcal{T}(s'_i)$ to the starting point of $\mathcal{T}(s'_{i+1})$. Therefore, with the query s_1, the result contains all documents corresponding to all keyword sets, $s'_1, s'_2, \ldots, s'_\delta$, i.e., all documents containing the keywords set, s_1.

Note that the user may generate a trapdoor for a keyword set corresponding to no document, i.e., an empty query. Therefore, a method to distinguish a proper query from an empty one is needed. Recall that each node has a check bit b_{ind} and the reply is made only if every check bit of the decrypted nodes has the proper value.

We claim that the probability of a server making a wrong reply for an empty query is negligible. The reason for this is easy to understand. Note that if a trapdoor, $\langle v;k \rangle$, is for an empty query, then k is not a proper secret key for $A[v]$. The server can compute $D_k(A[v])$ properly, but the server obtains a random string after decryption. This means that the server receives random links again. The process is terminated with a wrong result if all random links have an *EMPTY* value and all check bits have a value of 1. The probability that one link has the value of *EMPTY* is almost $1/\mu$. Recall that μ indicates the number of elements in A. The probability that the server will terminate the process properly after searching m nodes is $(1/2)^m \times 1/\mu^{m+1}$. Finally, the probability that the server will generate a wrong reply is $\sum_{m=1}^{\infty} 1/(2^m \mu^{m+1})$. For a large μ, this can be summarized as $1/(2\mu^2 - \mu)$.

Up to this point, we have not considered a collision of f, an event in which f outputs the same value for different inputs. A collision disturbs the construction of a linked tree structure. The more that array A is filled, the more serious collision effect becomes. In the proposed protocol, however, only the starting point of a linked tree is influenced by a collision. The other elements are randomly chosen among empty elements instead of the output of f. Compared to the size of A, the number of starting points is very small, and thus the starting points are very sparsely positioned in A. Since there are many well-developed methods used to

manage a sparse hash table efficiently, we can solve the collision problem easily (usually, at the cost of slightly increased storage). In this paper, we omit the details of such a solution.

4.2 Efficiency

Size of array *A*: We claim that $v(\ell)\cdot N$ nodes are sufficient to store all linked trees. Here, we will provide a practical bound for $v(\ell)$.

Recall that $v(\ell)$ means the number of all possible conjunctive keywords consist of at most ℓ keywords and the bound for $v(\ell)$ is $v(\ell) \le {}_nC_1 + {}_nC_2 + \ldots + {}_nC_\ell$, where n means the number of all possible keywords. However, this bound is not reasonable when it is applied to the storage size. Remind that in the BuildIndex algorithm only queries which correspond to at least one document are stored, i.e. no empty query is stored. Since each document has at most ℓ keywords, in the viewpoint of storage $v(\ell)$ is bounded as $v(\ell) \le {}_\ell C_1 + {}_\ell C_2 + \ldots + {}_\ell C_\ell = 2^\ell - 1$. This is also theoretical bound for $v(\ell)$. In a typical case, many common keywords exist in a database, and thus the actual number of nodes required greatly decreases. To measure the storage size in a typical case, we select a real data set, 1990 US census data [14], and simulate a build index process. The data set consists of 68 attributes and 48,536 rows. We generated simulations using $\ell = 5$ and $\ell = 10$. Table 1 shows the results of the simulations, where $\bar{v}(\ell)$ is the measured ratio of the total storage size over the number of documents, i.e., $|A|/N$.

Table 1. **Storage size in usual cases**

rows	10,000	20,000	30,000	40,000
$\bar{v}(5)$	1.283	1.164	1.119	1.095
$\bar{v}(10)$	41.649	28.050	22.139	18.566

We can easily deduce that this ratio decreases more as the size of the database becomes larger.

Trapdoor Size: The trapdoor is only one link, independent of the number of keywords, which are queried conjunctively. More precisely, $\log \mu = \log 2^\ell \cdot N = \ell + \log N$ bits are sufficient for an address in the array A, and the encryption key is λ bits. Therefore, the size of a trapdoor is $\ell + \log N + \lambda$ bits.

Computational Cost: First, consider the computational cost for a query. Assume that s_i's are sets of keywords satisfying $s_1 \subset s_2 \subset s_3 \subset \ldots \subset s_\ell$. If s_1 is queried then the result of the query is $\overline{\mathcal{D}}(s_1) \cup \overline{\mathcal{D}}(s_2) \cup \cdots \cup \overline{\mathcal{D}}(s_\ell)$. Therefore, the number of documents in the result is $|\overline{\mathcal{D}}(s_1)| + |\overline{\mathcal{D}}(s_2)| + \ldots + |\overline{\mathcal{D}}(s_\ell)|$, say m.

Note that the amount of computation for search is decrypting all nodes in $\mathcal{T}(s_1)$, $\mathcal{T}(s_2)$, …, $\mathcal{T}(s_\ell)$, where $\mathcal{T}(s_i)$ is the linked tree corresponding to s_i. If $|\overline{\mathcal{D}}(s_i)| > 0$ then the number of nodes in $\mathcal{T}(s_i)$ is equal to $|\overline{\mathcal{D}}(s_i)|$. On the other hand, $|\overline{\mathcal{D}}(s_i)| = 0$ implies $\mathcal{T}(s_i)$ has just one node, the root node. Therefore, if $|\overline{\mathcal{D}}(s_i)| > 0$ for all i, then the computation cost is $|\overline{\mathcal{D}}(s_1)| + |\overline{\mathcal{D}}(s_2)| + \ldots + |\overline{\mathcal{D}}(s_\ell)|$ decryptions, i.e. m decryptions. If $|\overline{\mathcal{D}}(s_i)| = 0$ for some i, then the cost increases. In the worst case, $|\overline{\mathcal{D}}(s_1)| = |\overline{\mathcal{D}}(s_2)| = \ldots = |\overline{\mathcal{D}}(s_{\ell-1})| = 0$ and $|\overline{\mathcal{D}}(s_\ell)| = m$ implies that the computation cost is $1 + 1 + \ldots + 1 + m = \ell - 1 + m$ decryptions. This means that the actual amount of work for the server is strictly smaller than $\ell + m$ decryptions and table searches.

For an empty query, the server terminates the process if a check bit which equals to 0 is found. Remind that if a trapdoor, $\langle v;k \rangle$, is for an empty query, then k is not a proper secret key

for $A[v]$. The probability of $D_k(A[v])$ provides 0 for the check bit is 1/2. It means that the probability of the server terminates search procedure for empty query within 1 decryption is also 1/2. Similarly, the server terminates the search within 2 decryption if the first check bit is 1 and the second check bit is 0, i.e. probability = 1/2 × 1/2. Therefore, the expectation value can be summarized as

$$1\cdot(1/2) + 2\cdot(1/2)^2 + 3\cdot(1/2)^3 + \ldots = \sum_{m=1}^{\infty} m\cdot(1/2)^m$$

For any |x| < 1, $\sum_{m=1}^{\infty} mx^m = x/(1-x)^2 = (1/2)/(1/2)^2 = 2$. This means that in average 2 nodes are checked for an empty query.

Recall that array A consists of at most $v(\ell)\cdot N$ nodes. In the *buildIndex* procedure, the amount of work conducted by the user is also $O(v(\ell)\cdot N)$ for the following reasons: In Step 1, for each document, there are at most $v(\ell)$ keyword sets, and thus $N\times O(v(\ell))$ is sufficient. In Step 2(a), the initialization of $v(\ell)\cdot N$ nodes requires $O(v(\ell)\cdot N)$. In Step 2(b), the amount of work is $O(|\mathcal{P}|)$, where $|\mathcal{P}| \le N\times v(\ell)$. In Step 2(d), only N nodes are modified. In Steps 2(e) and 2(f), the number of total external links are bounded by $v(\ell)\cdot N$, and thus the amount of work is also $O(v(\ell)\cdot N)$.

4.3 Security

Theorem 1: The proposed scheme satisfies non-adaptive indistinguishability security.

(Proof) For the proof, we define a simulator which is a probabilistic polynomial-time algorithm. We assume that with the input $Tr(H^{(q)})$, $\mathcal{S}$ can generate $\tilde{V}(H^{(q)})$ which is indistinguishable from $V(H^{(q)})$. We claim that constructing such a simulator is sufficient to prove the above theorem. Remind that the proposed scheme is secure in the sense of non-adaptive indistinguishability, if for all positive integers q, all (non-uniform) probabilistic polynomial-time adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, all polynomials p, and all sufficiently large k,

$$\Pr[b' = b \mid (H_0^{(q)}, H_1^{(q)}, \text{state}) \leftarrow \mathcal{A}_1; b \leftarrow_R \{0,1\}; b' \leftarrow \mathcal{A}_2(V(H_b^{(q)}), \text{state})] < \frac{1}{2} + \frac{1}{p(k)},$$

where $(H_0^{(q)}, H_1^{(q)})$ are histories over q queries such that $Tr(H_0^{(q)}) = Tr(H_1^{(q)})$, the 'state' is a polynomially bounded string that captures $\mathcal{A}_1$'s state, and the probability is taken over the internal coins of $\mathcal{A}$, and the underlying *BuildIndex* algorithm.

Assume that if there exists $\mathcal{A}_2$ that can find b' with a probability of significantly greater than 1/2, then it means that $\mathcal{A}_2$ can distinguish $V(H_0^{(q)})$ from $V(H_1^{(q)})$ with non-negligible probability. From the existence of $\mathcal{S}$ which can simulate the view from the trace, and the fact of $Tr(H_0^{(q)}) = Tr(H_1^{(q)})$, we have $\mathcal{S}(Tr(H_0^{(q)})) = \mathcal{S}(Tr(H_1^{(q)}))$. By the definition of $\mathcal{S}$, $V(H_0^{(q)})$ is indistinguishable from $\mathcal{S}(Tr(H_0^{(q)}))$, and that $V(H_1^{(q)})$ is also indistinguishable from $\mathcal{S}(Tr(H_1^{(q)}))$. Therefore, there are no polynomial time distinguishers that can distinguish $V(H_0^{(q)})$ from $V(H_1^{(q)})$, which is contradictory to the assumption that $\mathcal{A}_2$ can find b' with a probability significantly greater than 1/2. This ends the proof.

To prove the claim, we start with the construction of $\mathcal{S}(Tr(H^{(0)}))$. For $q = 0$, $\mathcal{S}$ generates $\tilde{V}^{(0)} = (id(D_1),\ldots,id(D_N), \tilde{E}(D_1),\ldots,\tilde{E}(D_N), \tilde{A})$ such that each $id(D_i)$ is copied from $Tr(H^{(0)})$ and $\tilde{E}(D_i) \leftarrow_R \{0,1\}^{|D_i|}$ for $1 \le i \le N$. Finally, $\mathcal{S}$ constructs $\tilde{A}$ as follows: $\mathcal{S}$ makes $\tilde{A}$ into an array with μ elements, where each element is a random string with the same length as an element in A.

We claim that no probabilistic polynomial-time adversary $\mathcal{A}$ can distinguish $\tilde{V}^{(0)}$ from $V(H^{(0)}) = (\mathrm{id}(D_1), \ldots, \mathrm{id}(D_N), E(D_1), \ldots, E(D_N), A)$ for any $H^{(0)}$. By a standard hybrid argument, if there exists an adversary $\mathcal{A}$ that can distinguish $\tilde{V}^{(0)}$ from $V(H^{(0)})$, then we can easily deduce that $\mathcal{A}$ can distinguish at least one of the elements of $\tilde{V}^{(0)}$ from its corresponding element in $V(H^{(0)})$. Note that the view of history consists of three types of elements, the document identifiers, encrypted documents, and the array A. Since the document identifier in $\tilde{V}^{(0)}$ is exactly the same as that in $V(H^{(0)})$, it is not possible to distinguish the identifiers. From the security of the embedded encryption algorithm, a ciphertext $E(D_i)$ is indistinguishable from a random string $\tilde{E}(D_i)$. Finally, note that each element of array A is also encrypted using a semantically secure encryption scheme, and therefore cannot be distinguishable from a random string with the same length.

For $q > 0$, $\mathcal{S}$ constructs $\tilde{V}^{(q)} = (id(D_1), \ldots, id(D_N), \tilde{E}_1, \ldots, \tilde{E}_N, \tilde{T}_1, \ldots, \tilde{T}_q, \tilde{A})$, such that $\mathrm{id}(D_i)$ is copied from $Tr(H^{(q)})$ and $\tilde{E}(D_i) \leftarrow_R \{0,1\}^{|D_i|}$ for $1 \le i \le N$. Recall that $Tr(H^{(q)}) = (\mathrm{id}(D_1), \ldots, \mathrm{id}(D_N), |D_1|, \ldots, |D_N|, \mathcal{D}(s_1), \ldots, \mathcal{D}(s_q), \Pi_q)$. To build $\tilde{A}$, $\mathcal{S}$ chooses a symmetric key encryption algorithm $\tilde{E}$, random secret key $\tilde{k}$, and two pseudorandom functions $\tilde{f}$ and $\tilde{h}$, randomly. $\mathcal{S}$ also sets $\tilde{\mathcal{D}} = \mathcal{D}(s_1) \cup \mathcal{D}(s_2) \cup \cdots \cup \mathcal{D}(s_q) \cup \tilde{\mathcal{D}}_0$, where $\tilde{\mathcal{D}}_0$ is a set of dummy documents to make $|\tilde{\mathcal{D}}| = N$. Assume that $\tilde{\mathcal{D}}_0$ contains no documents matched to any s_i. $\mathcal{S}$ runs *Build index*$(\tilde{\mathcal{D}}, \tilde{E}, \tilde{k}, \tilde{f}, \tilde{h})$ and can obtain an array $\tilde{A}$. Finally, $\mathcal{S}$ generates trapdoors $\tilde{T}_i$ as a link for the root node of the linked tree $\mathcal{T}(s_1)$.

We again claim that no probabilistic polynomial-time adversary $\mathcal{A}$ can distinguish $\tilde{V}^{(q)}$ from $V(H^{(q)}) = (\mathrm{id}(D_1), \ldots, \mathrm{id}(D_N), E(D_1), \ldots, E(D_N), T_1, \ldots, T_q, A)$. By a standard hybrid argument, if there exists an adversary $\mathcal{A}$ that can distinguish $\tilde{V}^{(q)}$ from $V(H^{(q)})$, then we can easily deduce that $\mathcal{A}$ can distinguish at least one of the elements of $\tilde{V}^{(q)}$ from its corresponding element in $V(H^{(q)})$. Identifiers and encrypted documents are also indistinguishable by the same reason for the case of $q = 0$.

Every node in $\tilde{A}$ is encrypted by a semantically secure encryption algorithm. Therefore, from the security of the encryption algorithm, we can deduce that no polynomial-time adversary can distinguish these nodes from the nodes in A. For the trapdoor, we should check whether the given trapdoor makes correct output. In other words, using the given trapdoor $\tilde{T}_i$, we must be able to obtain the list of documents $\mathcal{D}(s_i)$. Note that $\tilde{A}$ is generated from $\tilde{\mathcal{D}} = \mathcal{D}(s_1) \cup \mathcal{D}(s_2) \cup \cdots \cup \mathcal{D}(s_q) \cup \tilde{\mathcal{D}}_0$. Therefore $\tilde{A}$ contains a linked tree for $\mathcal{D}(s_i)$ and $\tilde{T}_i$ represents the link to the root node of corresponding linked tree. Thus, we can obtain the same results though the regular search process using the given trapdoors. This means that no adversary can distinguish $\tilde{A}$ from A.

4.4 Comparison

Previously proposed searchable encryption protocols with a conjunctive keyword search mainly utilize public-key cryptosystem based techniques, particularly a bilinear map over elliptic curves. Thus, these protocols have different properties and advantages (or disadvantages) from those of our scheme. In our construction, we focused on optimizing efficiency of searching process. As a result, we provide the first conjunctive keyword search method of which the search time is independent from the number of all stored documents.

Note that the other protocols require $O(N)$ computation for search. It means that for searching only one document millions (or billions) of pairing (or secret sharing) computations are required. However, our protocol has a weakness in the storage size (multiplied by $v(\ell)$ rather than ℓ). The value of $v(\ell)$ is changed by the stored data type. For the data-set in which same keywords are frequently repeated (for example zip-code, age, etc.), $v(\ell)$ has a very small value and the weakness can be mitigated.

We provide a comparison in the table below. In the table, N is the number of all documents, ℓ is the maximum number of keywords which is queried conjunctively, $v(\ell)$ is a constant determined by the characteristic of a given data set, m is the number of matched documents, m' is the number of documents which are matched to one of conjunctively queried keywords (usually $m' > m$), 'pairing' means computation of bilinear map over EC, and 'SD' means computation of symmetric key decryption.

Table 2. Comparison

		Encryption type	Index size	Trapdoor size	Computation Cost for Search
GSW04[10]		Public key	$O(\ell \cdot N)$	$O(1)$	$(2\ell+1)\cdot N \times$ (pairing)
BW07[12]		Public key	$O(\ell \cdot n \cdot N)$	$O(\ell)$	$(2\ell+1)\cdot N \times$ (pairing)
BKM05[11]	Pairing	Symmetric key	$O(\ell \cdot N)$	$O(1)$	$2N \times$ (pairing)
	Secret Sharing		$O(\ell \cdot N)$	$O(N)$	$N \times$ (secret sharing)
CJJ13[13]		Symmetric key	$O(\ell \cdot N)$	$O(m' \cdot \ell)$	$m' \cdot \ell \times$ (exponentiation)
Proposed scheme		Symmetric key	$O(v(\ell) \cdot N)$	$O(1)$	$(\ell + m) \times$ (SD)

5. Conclusion

We proposed a new protocol for a conjunctive keyword search on encrypted databases. The protocol is based on a linked tree structure instead of public-key cryptosystem based techniques, which were usually adopted in previous conjunctive keyword search protocols. Since the new protocol utilizes only a symmetric-key cryptosystem, it requires a remarkably small computation cost for managing an extremely large database. Moreover, this is the first conjunctive keyword search protocol in which the search time is independent of the size of the entire database. It was proven to be non-adaptive indistinguishability secure. We expect that the proposed scheme can be improved to satisfy the adaptive indistinguishability security with the further works.

References

[1] R. Ostrovsky, "Efficient computation on oblivious RAMs," in *Proc. of 22nd Annual ACM Symposium on Theory of Computing*, pp.514-523, May 13-17, 1990. Article (CrossRefLink).

[2] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *Journal of the ACM, 43(3)*, pp.431-473, May, 1996. Article (CrossRefLink).

[3] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. of IEEE Symposium on Security and Privacy 2000,* pp. 44-55, May 14-17, 2000. Article (CrossRefLink).

[4] E.-J. Goh, "Secure indexes," *Cryptology ePrint Archive 2003/216*, 2003. Article (CrossRefLink) .
[5] S. Bellovin and W. Cheswick, "Privacy-enhanced searches using encrypted Bloom filters," *Cryptology ePrint Archive 2004/022*, 2004. Article (CrossRefLink) .
[6] Y. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Proc. of Applied Cryptography and Network Security Conference (ACNS)*, pp. 442-455 June 7-10, 2005. Article (CrossRefLink).
[7] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. of ACM CCS 06*, pp. 79-88, October 30 – November 3, 2006. Article (CrossRefLink).
[8] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. of Eurocrypt 2004*, pp. 506-522, May 2-6, 2004. Article (CrossRefLink).
[9] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi, "Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions," in *Proc. of Crypto 2005*, pp. 205-222, August 14-18, 2005. Article (CrossRefLink).
[10] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *Proc. Applied Cryptography and Network Security Conference (ACNS)*, pp. 31-45, June 8-11, 2004. Article (CrossRefLink).
[11] L. Ballard, S. Kamara, and F. Monrose, "Achieving efficient conjunctive keyword searches over encrypted data," in *Proc. of ICICS 2005*, pp. 414-426, December 6-9, 2005. Article (CrossRefLink).
[12] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proc. of TCC 2007*, pp. 535-554, February 21-24, 2007. Article (CrossRefLink).
[13] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for Boolean queries," *IACR ePrint Cryptography Archive* 2013/169, 2013.
[14] 1990 US census data, http://kdd.ics.uci.edu .

Nam-Su Jho received his B.S. degree in mathematics from Korea Advanced Institute of Science and Technology, Daejeon, Korea, in 1999 and the Ph. D. degree in mathematics from Seoul National University, Seoul, Korea, in 2007. Since 2007, He has been with Electronics and Telecommunications Research Institute as a senior member of engineering staff. His research interests include cryptography, data privacy, and information theory.

Dowon Hong received his B.S., M.S. and Ph.D. degrees in mathematics from Korea University, Seoul, Korea on 1994, 1996, and 2000. He has been a principal member of engineering staff of Electronics and Telecommunications Research Institute, Korea from 2000 to 2012. Since March 2012, he has been an associate professor of the Department of Applied Mathematics at Kongju National University, Korea. His research interests include cryptography and information security, data privacy, and digital forensics.