SSS-V2: Secure Similarity Search

Hyun-A Park

Abstract—Encrypting information has been regarded as one of the most substantial approaches to protect users' sensitive information in radically changing internet technology era. In prior research, researchers have considered similarity search over encrypted documents infeasible, because the single-bit difference of a plaintext would result in an enormous bits difference in the corresponding ciphertext. However, we propose a novel idea of Security Similarity Search (SSS) over encrypted documents by applying character-wise encryption with approximate string matching to keyword index search systems. In order to do this, we define the security requirements of similarity search over encrypted data, propose two similarity search schemes, and formally prove the security of the schemes. The first scheme is more efficient, while the second scheme achieves perfect similarity search privacy. Surprisingly, the second scheme turns out to be faster than other keyword index search schemes with keyword-wise encryption, while enjoying the same level of security. The schemes of SSS support "like query('ab%')" and a query with misprints in that the character-wise encryption preserves the degree of similarity between two plaintexts, and renders approximate string matching between the corresponding ciphertexts possible without decryption.

Index Terms—keyword index search, encrypted data, similarity search, character-wise encryption, approximate string matching, hamming distance

1 Introduction

In the rapidly developing internet technology era, the amount of sensitive information radically increases and the information is required to be stored and managed in networked servers. In order to protect sensitive information from the inside and outside attackers, encryption is generally regarded as the last line of database defense. In prior research, keyword index search has been conducted over encrypted documents with various scenarios; however, in the field of the keyword index search over encrypted documents, there remains four problems in DBMS (DataBase Management System) as follows: (1) decline of performance, (2) difficulty for arithmetic and aggregated SQL queries, (3) challenge of similarity search, (4) difficulty in dynamic group search. This is because encryption transforms data into indistinguishable random strings, which can be read only by the holder of the decryption key. Park et al. [34] tried to solve the decline of performance, which is one of the most important and serious problems in this research area. [12],[16],[24],[39],[25] dealt with the difficulty of arithmetic and aggregated SQL queries and [34],[44] focused on the difficulty of dynamic group search. Yet, the challenge in similarity search has not been studied adequately. Regarding the reason for the challenge in similarity search, a single-bit change in a plaintext would result in the difference of enormous bits in the ciphertext; therefore, a similarity search such as a "like query" or queries with some misprints has been considered infeasible on encrypted documents. However, we believe that a similarity search is never infeasible on encrypted documents. As a result, in this paper, we propose two similarity search schemes over encrypted documents.

1.1 Key Idea and Contribution

The key idea of our schemes is to do with character-wise encryption with approximate string matching test, which enables similarity search over encrypted documents. In order to do this, we encrypt each character of the keyword separately, instead of encrypting a keyword. The value of this character-wise encryption preserves the degree of similarity between plaintexts and makes an approximate string matching test applicable to ciphertexts as well. Here, we utilize Hamming Distance as the simplest method to measure approximate string matching test. However, simple character-wise encryption is vulnerable to a dictionary attack because the domain is too small. That is, the total number of input alphabetical characters is only 26 and so is the total number of encryption outputs. Through trial and error, a malicious attacker can decrypt everything. To resolve this problem in character-wise encryption, we encrypt a character together with many secret values and identifiers (the user's secret key; the document's identifier to which the keyword belongs; the positional number of the character in the keyword; and the field's identifier of the cell in which the keyword is going to be stored). By this method of encryption, all characters are encrypted into different ciphers on the cells; we call this 'Index Privacy'.

In this paper, we propose two similarity search schemes (SSS-I and SSS-II) on encrypted documents, and define the privacy requirements of the similarity search. Our schemes can guarantee 'Index Privacy' – one of the privacy requirements we define. The second scheme (SSS-II) can satisfy 'Perfect Similarity Search Privacy' – the best security requirement. The implementation of the prototypes shows that our first scheme SSS-I is more efficient by accepting a slightly weaker security guarantee. We find that our second scheme SSS-II is faster than other keyword index search schemes with keyword-wise encryption (Golle *et al.*'s). At the same time, SSS-II can guarantee the same level of security as Golle *et al.*'s. Although our schemes cannot achieve good performance, they are the first attempt to enable similarity search over encrypted documents. Therefore, one of the most important findings of this paper is to support 'like queries'. In the encrypted database systems that do not support similarity search, to search 'ab%', one has to run 26 queries from 'aba' through to 'abz'. On the other hand, in our schemes, only one trial is sufficient. Additionally, our schemes can lower the rate of search failures that arise from misprints.

1.2 Related Works

So far search systems on encrypted data have been an active research area. At first, we look over the whole previous papers in this area, and then we look carefully the schemes related to similarity search over encrypted data.

Song *et al.* [40] firstly proposed a sequential scanning search algorithm over entire documents by using stream and block ciphers. Following this idea, most researches have been focused on the keyword index search. Boneh *et al.* [6] developed a keyword

• *Hyun-A Park is with HCI Lab, Konkuk University, 120 Neungdong-ro, Gwangjin-gu, Seoul, Korea, 143-701. E-mail: kokokzi@naver.com*

search with a public key system. Chang and Mitzenmacher [14] also proposed two index search schemes by the idea of pre-built dictionaries. Goh [19] formulated a security model of indexes known as semantic security (or indistinguishability) against an adaptive chosen keyword attack (IND-CKA) and also proposed an secure index scheme in the model. Waters *et al.* [42] published the building of an encrypted and searchable audit log which enables searches on the encrypted log with extracted keywords. Some of the proposed schemes extend the types of queries on encrypted data. In [12], Boneh *et al.* proposed a public key system that supports queries for testing any predicate on encrypted data by using tokens produced by a secret key. The authors constructed the systems for comparisons and subset queries, as well as conjunctive versions of these predicates. Hacigum*ü*s *et al.* [23] proposed a method for range queries on encrypted data in the DAS (Database As a Service) model by using privacy homomorphism which allows basic arithmetic $(+,-,\times)$ on encrypted data. Golle *et al.* [21] opened an efficient conjunctive keyword search over encrypted data in which their scheme constructs a keyword field. Shi *et al.* [39] and Katz *et al.* [25] developed some methods in public key predicate encryption system, where Shi *et al.*'s scheme supported delegation of capabilities and conjunctive queries. Katz *et al.*'s scheme supports inner-product queries and it is known as the most expressive scheme to date. They showed any predicate encryption scheme, supporting "inner product" predicates, can be used as a building block to construct more general types of predicates such as disjunctions, polynomials, CNF or DNF. Hwang *et al.* [24] constructed a conjunctive keyword search scheme for group users, based on a public key. Wang *et al.* [43] developed threshold privacy preserving keyword search scheme. However, the schemes of both [24] and [43] cannot support dynamic groups. Park *et al.* [33] firstly proposed search schemes for dynamic groups, which can deal with membership changes without re-encrypting documents for each change of membership. Moreover, in [34], they tried to resolve the inefficient problem caused by encryption and developed very efficient search scheme for practical use. This scheme can make a relational database over encrypted data by using a different database structure and applying an efficient database schema to encrypted database. Later, in [44], Wang *et al.* designed conjunctive keyword searches on encrypted data without keyword fields and extended the scheme to the setting of dynamic groups. In [11], Byun *et al.* raised a serious vulnerability of public key-based keyword search schemes, which are susceptible to an off-line keyword guessing attack caused by much smaller space than passwords.

Considering the prior research, most keyword index search schemes do not support similarity search, while [18] and [26] deal with approximate matching tests supporting finite or infinite alphabets over secure communication. Yet, these schemes are fundamentally different from an application, and they are not an encrypted-database-search-system point of view. Namely, they are much closer to Secure Multi-party (or 2-party) Computation Protocol. As to approximate matching test, although their approximate pattern matching test works for a finite or infinite alphabet, their data are only for DNA sequences, fingerprint images, voice patterns, and so on, not for documentary records or literatures.This case is easy because it does not require considering 'dictionary attack' in character-wise encryption. Rather, these schemes are similar to Oblivious Transfer Protocol and related to the Private Information Retrieval (PIR) Problem [13]. Therefore, communication overheads are very heavy: $O(n^2)$, $O(n \times N)$, and so on, where n is the total number of users' secret input and N is the total number of database strings T. Especially, [18] was proved insecure against ciphertext-only attacks by [26].

Recently, Pang *et al*'s paper, 'Privacy-Preserving Similarity-Based Text Retrieval' was published [31]. Their text retrieval scheme is designed for the vector space model and they apply a clustering algorithm on the document vectors, and then they search kNN(k nearest neighbors). Although they handle text retrieval, they do not consider an approximate matching test for finite or infinite alphabets. Namely, their scheme cannot provide similarity search such as "like query" or queries with misprints on encrypted documents. The scheme is more appropriate for biometric information such as DNA sequences, or imaging files rather than text or document retrieval, as other vector space models have bee designed.

Consequently, we propose novel schemes to enable a similarity search over encrypted documents.

2 Preliminaries

2.1 Database Model

Our database model is based on the Database As a Service (DAS) model [23], which is an instantiation of the computing model where a client is trusted and its data are stored in and managed by an untrustworthy server. The client has restricted computational power and storage, and relies on the server for mass computational power and storage.

Our schemes consist of setup and searching processes with three parties: users; a client; and a server. Initially, the client is given encryption keys. In the setup process, the client encrypts the data, generates the indexes of the keyword lists of the data, and uploads the encrypted data along with the corresponding indexes. The server can be an inside attacker in our scheme and is not allowed to read the data. Hence, the encryption key should not be known to the server (or the database administrator). In the searching process, a user queries the client with keywords in plaintexts. Upon receiving the query, the client generates trapdoors by encrypting the keywords and sends them to the server. The server then performs keyword search on indexes using the trapdoor and returns the search results to the client. Finally, the client decrypts the results and sends them back to the user. The messages on the communication channel between users and the client might be plaintexts, while the messages on the channel between the client and the server should be ciphertexts. Data privacy is assured under the condition that the client does not share the encryption keys, the metadata, or the unencrypted data with other parties.

We use the same Keyword Field as in Golle *et al.*'s scheme [21], which is made up of m keyword fields(columns) and each field(column) represents purpose-built attribute. We assume that each field has an assigned attribute or property. The relationship between each field and each attribute must be stored into the client for executing queries on behalf of users. The information of a document is stored in a single row.

2.2 Notations

In this paper, we will use the notations in TABLE 1.

2.3 Similarity Search Based on Approximate String Matching

The similarity search is based on approximate string matching in this paper. We use the 'Hamming Distance' as the approximate string matching method that defines the degree of similarity in our proposed scheme.

Definition 1. Hamming Distance and Degree of Similarity κ For two strings s and t, the Hamming Distance $H(s,t)$ is defined

TABLE 1
Notations

n	the number of documents
m	the number of fields.
s	the number of characters in the keyword
D_i	the i-th document
W_i	the document D_i's keyword list
W_{ij}	the keyword of the j-th field in the i-th document, where $1 \leq i \leq n$ and $1 \leq j \leq m$. i.e. $W_i = \{W_{i1},\ W_{i2}, ..., W_{im}\}$
w^s_{ij}	s-th character of the keyword W_{ij}. i.e. $W_{ij} = w^1_{ij}\|w^2_{ij}\|...\|w^s_{ij}$
d_i	the identifier of D_i which is randomly selected
j	the identifier of a field which is randomly selected.
I_i	the index list of W_i
I_{ij}	the index of W_{ij} i.e. $I_i = I_{i1}, I_{i2}, ...I_{ij}$
F	pseudo random function family
f	pseudo random function, $f \xleftarrow{\$} F$
$\mathfrak{g}$	random function
K	the set of users' secret keys
u	a user
k_u	a user u's secret key, $k_u \xleftarrow{\$} K$
f_{k_u}	the pseudorandom function with user's secret key k_u
T	trapdoor, an encrypted keyword which a user wants to search
P_{ij}	the pattern string of keyword W_{ij} for string matching test
HS	hash function whose range is $\{0,1\}^k$
$\|$	concatenation
g	a generator of a group G
id_{i1}	a kind of identifier of D_i in SSS-I. It is used for pattern generation.
id_{i2}	a kind of identifier of D_i in SSS-I. It is also used for pattern generation.
α	a random number generated every query time
h	hash function whose range is Z_q

as the number of places where the two strings differ. Accordingly, we define the degree of similarity as $H(s,t) \leq \kappa$, where $\kappa(\geq 0)$ is an integer [38].

2.4 Secure Similarity Search Model

A secure similarity search (SSS) scheme consists of the following seven algorithms.

- **SysParam(1^k):** Parameter generation algorithm *SysParam* takes as input a security parameter k, and produces a system parameter λ.
- **KeyGen(λ):** Given λ, key generation algorithm *KeyGen* produces users' search key set K and document encryption key set R.
- **DocEnc(R,D):** Algorithm *DocEnc* takes as input the data encryption key R and the document D. Its output is an encrypted document.
- **IndGen(K,W,d,J,s):** Index generation algorithm *IndGen* takes as input: user's secret key K; keyword list W; document's identifier d; fields(column)'s identifiers J; and the positional number of the character s. It outputs the index list I of the keyword list W.
- **Trapdoor(K, W_{*j}, j, s):** Trapdoor generation algorithm *Trapdoor* takes as input: the keyword W_{*j}; user's secret key K; field's identifier j; and the positional number of the character s. It encrypts the keyword W_{*j} and returns the encryption value which is trapdoor $T_{W_{*j}}$ for keyword W_{*j}.
- **PattGen(T,d):** Pattern generation algorithm *PattGen* is executed by a server. It takes as input the trapdoor T and document's identifier d. It generates pattern P according to the given protocol to implement the approximate string matching test for similarity test.
- **SimMatch(P,I):** Similarity matching algorithm *SimMattch* takes pattern P and index I as input. It returns 'yes' if the similarity matching test satisfies the given similarity κ, or 'no' otherwise.

2.5 Security Building Blocks

We now define security building blocks that are used for the construction of our schemes.

Definition 2. PRF(Pseudo Random Function): We say that '$F : K_f \times X \longrightarrow Y$ is (t,q,e) -secure pseudorandom function' if every oracle algorithm A making at most q oracle queries and with running time at most t has advantage $Adv_A < e$. The advantage is defined as $Adv_A = \mid Pr[A^{F_k} = 1] - Pr[A^{\mathfrak{g}} = 1] \mid$ where $\mathfrak{g}$ represents a random function selected uniformly from the set of all maps from X to Y, and where the probabilities are taken over the choice of k and $\mathfrak{g}$ [40].

Definition 3. PRG G_r(Pseudo Random Generator): We say that '$Gr : K_{Gr} \longrightarrow S$ is a (t,e)-secure pseudorandom generator' if every algorithm A with running time at most t has advantage $Adv_A < e$. The advantage is defined as $Adv_A = \mid Pr[A(Gr(U_{K_{Gr}})) = 1] - Pr[A(U_S) = 1] \mid$. Where $U_{K_{Gr}}$, U_S are random variables distributed uniformly on K_{Gr}, S [40].

Definition 4. DDH (Decisional Diffie-Hellman): Let G be a group of prime order q and g a generator of G. The DDH problem is to distinguish between triplets of the form (g^a, g^b, g^{ab}) and (g^a, g^b, g^c), where a,b,c are random elements of $\{1,...,q-1\}$.

Consider the following experiment with a polynomial time adversary A : Flip a coin δ to get 0 or 1, if $\delta = 1$, set $c = ab$, else choose c at random. The DDH problem is said to be hard if for any polynomial time adversary A, $|Pr(A(G, g^a, g^b, g^c) = \delta) - 1/2|$ is negligible.

2.6 Security Models

It is the security proof model that guarantees semantic security(Indistinguishability) against *CKA*(Chosen Keyword Attack) and is based on the security model of Goh [19] and Golle *et al.* [21].

Definition 5. Security Game ICC (Indistinguishability of Ciphertext from Ciphertext)

- **Setup.** The challenger C creates a set WS of q words $\in \{D_i\}$ $(1 \leq i \leq n)$ and gives this to the adversary A. A chooses a polynomial number of subsets from WS. This collection of subsets is called WS^* and is returned to C. Upon receiving WS^*, C runs algorithm SysParam and KeyGen and encrypts each subset running algorithm IndGen. Finally, C sends all indexes with their associated subsets to A.
- **Queries.** A is allowed to query C on a word W_{ij} and receives the trapdoor $T_{W_{ij}}$ for W_{ij}. With $T_{W_{ij}}$, A can invoke algorithm PattGen and SimMatch on an index I to determine if $H(P_{W_{ij}}, I) \leq \kappa$. ($P_{W_{ij}}$ is the pattern of the word 'W_{ij}' for string matching test.)
- **Challenge.** After making some Trapdoor queries, A decides on a challenge by picking two keywords sets W_0, W_1, a querying field j (i.e., $W_{0j} \in W_0$ and $W_{1j} \in W_1$), and **two t-th characters $\mathbf{w}^t_{0j} \in \mathbf{W}_{0j}$, $\mathbf{w}^t_{1j} \in \mathbf{W}_{1j}$**. A must not have queried the trapdoors for the keywords belonged to W_{0j} and W_{1j}. Next, A gives W_0 and W_1 to C and then C chooses $b \xleftarrow{\$} \{0,1\}$,

invokes algorithm IndGen to obtain I_b for W_b, and returns the encrypted value I^t_{bj} for a character w^t_{bj} to A. The challenge for A is to determine b, i.e. **distinguish between W0=$\mathbf{I}^t_{0j}$ for a character $\mathbf{w}^t_{0j}$ and W1=$\mathbf{I}^t_{1j}$ for a character $\mathbf{w}^t_{1j}$**. After the challenge is issued, A is not allowed to query the trapdoors for the keywords belonged to W_{0j} and W_{1j} to C.

- **Response.** A eventually outputs a bit b', representing its guess for b. The advantage of A in winning this game is defined as $Adv_A = |Pr[b = b'] - 1/2|$. Adversary $A(t, \varepsilon, q)$ is said to have an $\varepsilon - advantage$ if $Adv_A > \varepsilon$ after A takes at most t times and makes q trapdoor queries to the challenger.

For simplicity of our proofs, we define other variants of the security game ICC. We call them, ICR(Indistinguishability of Ciphertexts from Random) and ICLR(Indistinguishability of Ciphertexts from Limited Random).

Security Game ICR(Indistinguishability of Ciphertexts from Random).
In the first variant, the adversary **chooses one querying keyword** W_{*j} of the j-th field from keyword list W_*. W_{*j} is set to $W0$. The challenger creates a keyword $W1 = Rand(W_{*j}, w^t_{*j})$, where $Rand(W_{*j}, w^t_{*j})$ means that $W1$ from $W0$ is formed by replacing the character $w^t_{*j} \in W_{*j}$ as random values. The goal of A is to **distinguish between W0 and W1 for a keyword $\mathbf{W}_{*j}$**. The detailed process is similar to ICC.

According to [21], the existence of an adversary that wins the game ICC with non-negligible probability implies the existence of an adversary that wins the game ICR with non-negligible probability.

Security Game ICLR(Indistinguishability of Ciphertexts from Limited Random).
As the final security game, we consider an adversary who is able to **distinguish between W0=Rand($\mathbf{W}_*$, $\mathbf{W}_{*j} - \{\mathbf{w}^t_{*j}\}$) and W1=Rand($\mathbf{W}_*$, $\mathbf{W}_{*j}$), for one keywords set $\mathbf{W}_*$**, a querying keyword W_{*j}, and a character $w^t_{*j} \in W_{*j}$. $Rand(W_*, W_{*j} - \{w^t_{*j}\})$ means that W_* is formed by replacing $(W_{*j} - \{w^t_{*j}\}) \in W_{*j}$ as random values. $Rand(W_*, W_{*j})$ means that W_{*j} is replaced by as random values. The detailed processes are similar to ICC and ICR. With the same reason as the game ICR, an adversary in the game ICC implies the existence of an adversary that wins the game ICLR with non-negligible probability. [1]

Next, we introduce the privacy building blocks.

2.7 Privacy Building Blocks

Definition 6. Trapdoor Privacy

For a querying keyword W_{*j}, two words $W0$ and $W1$ are generated according to the security game $ICC(ICR)$, where if the given protocol except for random value follows trapdoor generation algorithm, we call $W0$ and $W1$ as T_0 and T_1.

For all polynomial time adversary A, we define that the search scheme provides 'Trapdoor Privacy' if an adversary A cannot distinguish the trapdoors T_0 from T_1 with non-negligible advantage. The advantage is defined as $Adv_A = |Pr[Exp^{ind-cka-trp-1}_A = 1] - Pr[Exp^{ind-cka-trp-0}_A = 1]|$.[2]

Definition 7. Index Privacy

For a keywords-list W_*, two keywords-lists $W0$ and $W1$ are generated according to the security game $ICC(ICLR)$, where if the given protocol except for random values follows index generation algorithm, $W0$ and $W1$ are said as I_0 and I_1.

We define that the search scheme provides 'index privacy' if all polynomial time adversary A cannot distinguish the indexes I_0 from I_1 with non-negligible advantage, which is defined as $Adv_A = |Pr[Exp^{ind-cka-idx-1}_A = 1] - Pr[Exp^{ind-cka-idx-0}_A = 1]|$. [3]

Definition 8. Perfect Similarity Search Privacy

It is defined that the search scheme provides 'Perfect Similarity Search Privacy' if both trapdoor privacy and index privacy are achieved.

3 Construction of Secure Similarity Search (SSS)

In this section, we construct our first scheme, Secure Similarity Search-I (SSS-I). Our scheme SSS-I is constructed by using the seven algorithms described before and this scheme is also divided largely into Setup and Searching processes.

3.1 SetUp Process

SetUp Process is divided again into three subprocesses: System Setting; Encryption; and Uploading.

3.1.1 System Setting

This process constructs the environment for the encrypted database searching system by using the algorithms $SysParam(1^k)$ and $KeyGen(\lambda)$.

1. $SysParam(1^k)$ *Construction.* A client chooses the security parameter k and generates system parameter $\lambda = (f(\cdot),\ HS(\cdot),\ s,\ m,\ n)$. λ determines the elements which are required to set the encrypted database system such as the size of system. $f : \{0,1\}^k \times \{0,1\}^* \rightarrow Z_q$ is a pseudo random function, $HS : \{0,1\}^* \rightarrow \{0,1\}^k$ is an one-way hash function. s is the total number of characters in the keyword, m is the total number of keyword fields, and n is the total number of documents.

2. $KeyGen(\lambda)$ *Construction.* Under the system parameter λ, two search key sets $K \in \{0,1\}^k$ and $R \in \{0,1\}^k$ are generated by a client. $K \in \{0,1\}^k$ is for index encryption and $R \in \{0,1\}^k$ is for document encryption.

3.1.2 Encryption

In this process, the algorithms $DocEnc(R,D)$ and $IndGen(K,W,d,J,s)$ are constructed to set up the encrypted database system.

1. $DocEnc(R,\ D)$ *Construction.* If a user wants to store sensitive documents, he selects representative keywords from each document and sends documents and their selected keyword lists $\{D,\ W\}$ to a client, where $D = \{D_1,\ D_2, ..\}$, $W = \{W_1,\ W_2, ..\}$, W_i=$\{W_{i1}, W_{i2}, ..., W_{im}\}$. After receiving the data $\{D,\ W\}$, a client encrypts documents $\{D\}$ with the document encryption key $k_r \in R$. $DocEnc(R,D)$ is completed.

2. $IndGen(K,\ W,\ d,\ J,\ s)$ *Construction.* A client selects a unique document identifier d_i at random for each document D_i. For each keyword list W_i=$\{W_{i1}, W_{i2} ..., W_{im}\}$, a client encrypts each keyword

1. In brief, Security Game ICC is an indistinguishability for a character, ICR is for a keyword, and ICLR is for a keyword list.
2. 'ind-cka-trp' means indistinguishability under CKA(chosen keyword attack) for trapdoor
3. 'ind-cka-idx' means indistinguishability under CKA for index

list W_i character by character with the user's secret key $k_u \in_R K$, each document identifier d_i, and fields(columns)' identifier set $J = \{j_1, j_2, ..., j_m\}$ where, $1 \leq i \leq n$, W_{ij}=$w^1_{ij} w^2_{ij} .. w^s_{ij}$, and s is a positional number of a character.

The encryption process is divided into two stages. First, each character is encrypted with the user's secret key k_u, the character's positional number in the word, and the field's identifier. Second, each character is encrypted with the document's identifier as the encryption key. We added the document's identifier in the second encryption so that even if two keyword lists, W_0 and W_1, have a common keyword, the common keyword has different encryption values for W_0 and W_1 because of the different document's identifier, d_0 and d_1. In addition, we encrypt the field's identifier j with the user's secret key k_u, so that even if two keywords W_{i0} and W_{i1} of a keyword list W_i either have a common character or are the same, the values are different because of differing field identifiers, viz., 0 and 1. Furthermore, a character's positional number s makes it impossible to distinguish w^o_{ij} from w^1_{ij} even if they are the same character of a keyword within a cell. Due to this style of encryption method, our scheme can achieve **"Index Privacy"**. The detailed encryption processes are as follows.

(1) The first encryption

For a simple illustration, we consider the case of only one keyword, $W_{ij} = w^1_{ij} w^2_{ij} ... w^s_{ij}$. A client computes the following:
$f_{k_u}(W_{ij} = w^1_{ij} w^2_{ij} ... w^s_{ij})$
$= f_{k_u}(1_{st}\ character|character's\ positional\ number|field's\ identifier)$
$f_{k_u}(2_{nd}\ character|character's\ positional\ number|field's\ identifier)|..$
$..|f_{k_u}(s_{th}\ character|character's\ positional\ number|field's\ identifier)$
$= f_{k_u}(w^1_{ij}|1|j)|f_{k_u}(w^2_{i,j}|2|j)|......|f_{k_u}(w^s_{ij}|s|j)$

For more simple expression, we denote $f_{k_u}(w^s_{i,j}|s|j)$ as f^s_{ij}. Therefore, $f_{k_u}(W_{ij}) = f^1_{ij}|f^2_{ij}|......|f^s_{ij}$. Where, '|' denotes concatenation. In this process, each character is concatenated with the character's positional number and the field's identifier and then encrypted with user's key k_u.

Example 1. We use the fifth document D_5 as an example to illustrate.

- Keyword list $W_5 = \{god,\ thank,\ love,\ evil,\ song,\ heart,\ pray\}$,where m = 7
- The identifier d_5 of a document D_5 = '55555'
- A user u's secret key k_u = '12345'
  Among this keyword list W_5, we address only this keyword 'god'. Therefore,
- $W_{51} = god$: the first keyword of D_5
- w^1_{51}=g : the first character of W_{51}
- w^2_{51}=o : the second character of W_{51}
- w^3_{51}=d : the third character of W_{51}
  where, s is 3.

$$f_{12345}(god) = f_{12345}(g|1|1)|f_{12345}(o|2|1)|f_{12345}(d|3|1)$$

For simple expression, we denote $f_{k_u}(w^s_{ij}|s|j)$ as $f_{ij}(w^s_{ij})$. Therefore, $f_{12345}(god) = f_{51}(g)|f_{51}(o)|f_{51}(d)$.

(2) The second encryption

For the first encryption $f_{k_u}(W_{ij}) = f^1_{ij}|f^2_{ij}|...|f^s_{ij}$, a client encrypts each character with the document's identifier d_i and then hashes that as well, character by character. This value gets to the index I_{ij} for W_{ij}.
$HS(f_{d_i}(f^1_{ij}))|HS(f_{d_i}(f^2_{ij}))|..|HS(f_{d_i}(f^s_{ij})) = I_{ij}$

Finally, a client produces an index list I_i for W_i=$\{W_{i1}, W_{i2}, ..., W_{im}\}$ by collecting each index I_{ij} for W_{ij}. An index list I_i is as follows.
$I_i = d_i, I_{i1}, I_{i2}, ...I_{im}$
$= d_i,\ HS(f_{d_i}(f^1_{i1}))|...|HS(f_{d_i}(f^s_{i1})),\ HS(f_{d_i}(f^1_{i2}))|...|HS(f_{d_i}(f^s_{i2})), ...$
$......,\ HS(f_{d_i}(f^1_{im}))|...|HS(f_{d_i}(f^s_{im}))$

Example 2. For the first encryption $f_{12345}(god) = f_{51}(g)|f_{51}(o)|$ $f_{51}(d)$,
$HS(f_{55555}(f_{51}(g)))|HS(f_{55555}(f_{51}(o)))|HS(f_{55555}(f_{51}(d))) = I_{51}$

$I_5 = d_5, I_{51}, I_{52}, ...I_{57}$
$= 55555,\ \ HS(f_{55555}(f_{51}(g)))|HS(f_{55555}(f_{51}(o)))|HS(f_{55555}(f_{51}(d))),$
$HS(f_{55555}(f_{52}(t)))|HS(f_{55555}(f_{52}(h)))|HS(f_{55555}(f_{52}(a)))|HS(f_{55555}($
$f_{52}(n)))|HS(f_{55555}(f_{52}(k))), ..., f_{55555}(f_{57}(p))|f_{55555}(f_{57}(r))|HS($
$f_{55555}(f_{57}(a)))|HS(f_{55555}(f_{57}(y)))$

Where, I_{52} denotes the second keyword which is 'thank', $\cdots$, and I_{57} denotes the last keyword in the list which is 'pray'.

3.1.3 Uploading

After producing the encrypted documents and their index lists, a client sends them to a server. The server stores the index lists in a database table. The encrypted documents can be stored in a file storage system or database table if the database condition is allowed.

3.2 Searching Process

Given the setup of the encrypted search system, from this section onwards, we explain the Searching Process. The Searching Process is also again divided into three subprocesses: Trapdoor Generation and Querying; Verification; and Returning and Decryption.

3.2.1 Trapdoor Generation and Querying

In this process, an algorithm $Trapdoor(K, W_{*j}, j, s)$ is constructed by a client. If a user wants to search a keyword $W_{*j} = w^1_{*j} w^2_{*j} ... w^s_{*j}$, he queries a client with it. The client makes a trapdoor. We call an encrypted keyword query as a trapdoor. The trapdoor generation method is similar to the first encryption of $IndGen(K, W, d, J, s)$.

Let $T_{W_{*j}}$ be the trapdoor for one keyword $W_{*j} = w^1_{*j} w^2_{*j} ... w^s_{*j}$. Then, $T_{W_{*j}}$ can be described as: $T = (T_1,\ T_2)$. Where, T_1 indicates j-th field to search and T_2 is an encrypted keyword for $W_{*j} = w^1_{*j} w^2_{*j} ... w^s_{*j}$. T_2 is computed as follows:
$T_2 = f_{k_u}(1_{st}\ character|character's\ positional\ number|T_1)|$
$f_{k_u}(2_{nd}\ character|character's\ positional\ number|T_1)|....$
$....|f_{k_u}(s_{th}\ character|character's\ positional\ number|T_1)$
$= f_{k_u}(w^1_{*j}|1|j)|f_{k_u}(w^2_{*j}|2|j)|...|f_{k_u}(w^s_{*j}|s|j)$
For simple expression, we denote $f_{k_u}(w^s_{*j}|s|j)$ as t^s_j.
Therefore, $T_2 = t^1_j|t^2_j|...|t^s_j$

In more detail,
$T_1 = j:\ the\ field\ identifier$
$T_2 = f_{k_u}(w^1_{*j}|1|j)|f_{k_u}(w^2_{*j}|2|j)|.....|f_{k_u}(w^s_{*j}|s|j) = t^1_j|t^2_j|...|t^s_j$

A client queries this trapdoor $T = (T_1,\ T_2)$ for a keyword $W_{*j} = w^1_{*j} w^2_{*j} ... w^s_{*j}$ to a server.

Example 3. For a keyword 'god',
$T = (T_1,\ T_2)$,

$T_1 = 1_{st}\ field$
$T_2 = f_{12345}(g|1|1)|f_{12345}(o|2|1)|f_{12345}(d|3|1)$
$\quad = t_1^1|t_1^2|t_1^3$

3.2.2 Verification

In this process, a server constructs an algorithms $PattGen(T,d)$ and $SimMatch(P,I)$ to search what a user wants in an encrypted database with the queried trapdoor.

1. $PattGen(T,d)$ *Construction.* For approximate string matching over encrypted data, we need the encrypted patterns for a keyword $W_{*j} = w_{*j}^1 w_{*j}^2 ... w_{*j}^s$. A server generates each pattern P_{ij} for each document D_i, where $1 \leq i \leq n$ and n is the total number of documents.
$P_{ij} = HS(f_{d_i}(t_j^1))|HS(f_{d_i}(t_j^2))|...|HS(f_{d_i}(t_j^s))$.

That is, after receiving the trapdoor $T = (T_1,\ T_2)$, a server encrypts it with each document's identifier d_i, and then hashes it, character by character. A server produces all patterns, P_{1j} to P_{nj}, for all documents from D_1 through to D_n.

2. $SimMatch(P,I)$ *Construction.* A server implements Hamming Distance test between each document's index string I_{ij} and the produced pattern string P_{ij}. The computed Hamming Distance between a pattern string P_{ij} and an index string I_{ij} for a document D_i can be denoted as follows: $H(P_{ij},\ I_{ij})$, where $1 \leq i \leq n$. For each I_i, if it satisfies the condition $H(P_{ij},\ I_{ij}) \leq \kappa$, I_i is the index list of the document D_i which a user wants.

Example 4. From the above Example 3, we can produce all pattern strings from P_{11} to P_{n1} as follows:
$P_{11} = HS(f_{d_1}(t_1^1))|HS(f_{d_1}(t_1^2))|HS(f_{d_1}(t_1^3))$
$= HS(f_{d_1}(f_{12345}(g|1|1)))|HS(f_{d_1}(f_{12345}(o|2|1)))|HS(f_{d_1}(f_{12345}(d|3|1)))$

$P_{21} = HS(f_{d_2}(t_1^1))|HS(f_{d_2}(t_1^2))|HS(f_{d_2}(t_1^3))$
$= HS(f_{d_2}(f_{12345}(g|1|1)))|HS(f_{d_2}(f_{12345}(o|2|1)))|HS(f_{d_2}(f_{12345}(d|3|1)))$
....

$P_{n1} = HS(f_{d_n}(t_1^1))|HS(f_{d_n}(t_1^2))|HS(f_{d_n}(t_1^3))$
$= HS(f_{d_n}(f_{12345}(g|1|1)))|HS(f_{d_n}(f_{12345}(o|2|1)))|HS(f_{d_n}(f_{12345}(d|3|1)))$

Next, we need to compute Hamming Distance from I_1 to I_n for similarity test. We show only I_5 of the document D_5.
$H(P_{51},\ I_{51}) = 0$.
We assume $\kappa = 1$. Therefore, $H(P_{51},\ I_{51})$ satisfies this condition $H(P_{ij},\ I_{ij}) \leq \kappa$ and the document D_5 is what a user wants to retrieve.

3.2.3 Returning and Decryption

In the *Verification* Process, if there are index lists that satisfy the similarity degree of hamming distance, the server returns the corresponding documents to the client. The client decrypts them with the user's document encryption key k_r and then sends the decrypted documents to a user.

4 Security Analysis for SSS-I

The goal of designing SSS-I is focused on more practical respects in a real world. SSS-I can provide 'Index Privacy' but not 'Trapdoor Privacy'. In the index formation, the encryption requirement for Index Privacy should include all values of the user's secret key, the identifier of the document, and the identifier of the field. In addition to these, the character's positional number ensures that even the same character in the same cell have different encryption values. However, because SSS-I does not use a random factor in the generation of trapdoor, it cannot provide 'Trapdoor Privacy'. We prove the index privacy of SSS-I by using the security game model ICLR. First, proof sketch is given and then a detailed, formal proof is addressed in APPENDIX A.

Theorem 1. SSS-I can guarantee 'Index Privacy' according to the game ICC if f is (t,q,e) -secure pseudorandom function.

Proof Sketch. We prove it by contraposition. We assume that SSS-I cannot provide 'Index Privacy' under the security game ICC. According to [21], the existence of an adversary that wins the game ICC with non-negligible probability implies the existence of an adversary that wins the game ICLR with non-negligible probability. Let A be an adversary that wins the game ICLR with advantage ε. We construct an adversary β which can solve the problem of whether f is pseudorandom function or random function. β can access an oracle $\mathscr{O}_f$ for the unknown function f. In every algorithm, β substitutes the values of f through the queries to the oracle $\mathscr{O}_f$. β uses an algorithm A as a subroutine and simulates algorithm A by using Security Game ICLR. By constructing an adversary β, under the assumption of the existence of an adversary $A(t,\varepsilon,q)$, we show that SSS-I can provide 'Index Privacy'.

5 Construction of Secure Similarity Search-II (SSS-II)

The first scheme SSS-I can achieve high efficiency but it cannot guarantee 'Trapdoor Privacy'. Thus it cannot guarantee 'Perfect Similarity Search Privacy'. It motivates us to develop a secure similarity search scheme that can provide 'Perfect Similarity Search Privacy'. To the best of our knowledge, most of the previous schemes have not been able to guarantee 'Trapdoor Privacy' except for that of Golle *et al.* [21]. However, in [21], to provide 'Trapdoor Privacy', the authors have to update some index parts of all documents during each query. Our objective in SSS-II is to design a scheme that can guarantee 'Perfect Similarity Search Privacy', and at the same time, not require that all the documents be updated every query time. The details related to this problem are addressed in Section 8.

In this section, we construct our second scheme SSS-II. We use the Keyword Field again as we did in SSS-I. SSS-II is also constructed by using seven algorithms.

5.1 SetUp Process

5.1.1 System Setting

Like SSS-I, this process constructs the algorithms $SysParam(1^k)$ and $KeyGen(\lambda)$. The constructions are very similar to SSS-I. The only difference is $\lambda = (G,\ g,\ f(\cdot),\ G_r,\ h(\cdot),\ HS(\cdot),\ s,\ m,\ n)$. G is a group of order q which is a large prime and g is a generator of a group G. $f : \{0,1\}^k \times \{0,1\}^* \rightarrow Z_q$ is a pseudo random function and G_r is a pseudo random generator. $h : \{0,1\}^* \rightarrow Z_q$ and $HS : \{0,1\}^* \rightarrow \{0,1\}^k$ are one way hash functions. Other processes are the same as SSS-I.

5.1.2 Encryption

1. *$DocEnc(R,D)$ and the first encryption of $IndGen(K,W,d,J,s)$ Construction.* In this process, the construction of algorithms $DocEnc(R,D)$ and the first encryption of $IndGen(K,W,d,J,s)$

are the same as SSS-I but the second encryption of $IndGen(K,W,d,J,s)$ is different. After constructing $DocEnc(R,D)$ and the first encryption of $IndGen(K,W,d,J,s)$, a client implements the second encryption.

2. *The second encryption of* $IndGen(K,\ W,\ d,\ J,\ s)$ *Construction***.**

For the first encryption $f_{k_u}(W_{ij}) = f_{k_u}(w^1_{ij}|1|j)|f_{k_u}(w^2_{ij}|2|j)|......|$ $f_{k_u}(w^s_{ij}|s|j)$, a client multiplies these values character by character: d_i; $h(k_u)$; and the first encryption value $f_{k_u}(w^s_{ij}|s|j)$. This multiplied value gets to the power of g: $g^{d_ih(k_u)f_{k_u}(w^1_{ij}|1|j)}|g^{d_ih(k_u)f_{k_u}(w^2_{ij}|2|j)}|...|g^{d_ih(k_u)f_{k_u}(w^s_{ij}|s|j)}$.
After that, it is also hashed character by character. This value is the index I_{ij} for one keyword W_{ij}.

$$I_{ij} = HS(g^{d_ih(k_u)f_{k_u}(w^1_{ij}|1|j)})|HS(g^{d_ih(k_u)f_{k_u}(w^2_{ij}|2|j)})|.......$$
$$.......|HS(g^{d_ih(k_u)f_{k_u}(w^s_{ij}|s|j)})$$

Finally, a client produces an index list I_i for keyword list W_i=$\{W_{i1},W_{i2},...,W_{im}\}$ by collecting each index I_{ij} for each keyword W_{ij}. An index list I_i is as follow.

$$I_i = id_{i1}, id_{i2}, I_{i1}, I_{i2}, ...I_{im}$$

Where, $id_{i1} = g^{d_ih(k_u)}$, $id_{i2} = g^{-d_i}$. These are a kind of identifiers of document D_i.

Example 5. For a keyword 'god',

$$I_{51} = HS(g^{d_5h(k_u)f_{k_u}(w^1_{51}|1|1)})|HS(g^{d_5h(k_u)f_{k_u}(w^2_{51}|2|1)})|.......$$
$$........|HS(g^{d_5h(k_u)f_{k_u}(w^s_{51}|s|1)})$$

Where, we assume $h(k_u) = 35$, $id_{51} = g^{(55555)(35)}$, and $id_{52} = g^{-(55555)}$. Therefore,

$$I_{51} = HS(g^{(55555)(35)(f_{12345}(g|1|1))})|HS(g^{(55555)(35)(f_{12345}(o|2|1))})|$$
$$HS(g^{(55555)(35)(f_{12345}(d|3|1))})$$

$$I_5 = id_{51},\ id_{52},\ I_{51},\ I_{52}, ...I_{57}$$
$$= \ g^{(55555)(35)},\ g^{-(55555)},\ HS(g^{(55555)(35)(f_{12345}(g|1|1))})|$$
$$HS(g^{(55555)(35)(f_{12345}(o|2|1))})|HS(g^{(55555)(35)(f_{12345}(d|3|1))}),..$$
$$..,HS(g^{(55555)(35)(f_{12345}(p|1|7))})|HS(g^{(55555)(35)(f_{12345}(r|2|7))})|$$
$$HS(g^{(55555)(35)(f_{12345}(a|3|7))})|HS(g^{(55555)(35)(f_{12345}(y|4|7))})$$

5.1.3 Uploading

This process is the same as SSS-I.

5.2 Searching Process

5.2.1 Trapdoor Generation and Querying

If a user wants to search a keyword $W_{*j} = w^1_{*j}w^2_{*j}...w^s_{*j}$, s/he queries a client with it. In particular, in the construction of algorithm $Trapdoor(K,W_{*j},j,s)$ of SSS-II, we add a random number α to the trapdoor in each query to prohibit a server from deducing some information from cumulative results. The formation of trapdoor in SSS-II is as follows:

$T = (T_1,T_2,T_3)$
$T_1 = j : the\ field\ identifier$
$T_2 = h(k_u)\alpha$
$T_3 = (f_{k_u}(w^1_{*j}|1|j)+\alpha)|(f_{k_u}(w^2_{*j}|2|j)+\alpha)|...|(f_{k_u}(w^s_{*j}|s|j)+\alpha)$

We put $(f_{k_u}(w^s_{*j}|s|j)+\alpha)$ as t^s for simplicity. Therefore,
$T_3 = t^1|t^2|...|t^s$
t^1=$(f_{k_u}(w^1_{*j}|1|j)+\alpha)$
t^2=$(f_{k_u}(w^2_{*j}|2|j)+\alpha)$
....
t^s=$(f_{k_u}(w^s_{*j}|s|j)+\alpha)$

Where, T_1 is the field number of the keyword that a user wants to search. T_2 is the multiplication of the random number α by the hashed value of the user's secret key k_u. T_3 is the masked value with α for the first encryption of each character $f_{k_u}(w^s_{*j}|s|j)$ to provide 'trapdoor privacy'.

α is the random value which is newly generated during each query by the pseudo random generator, and the trapdoor is computed by modulus. Hence, a server cannot learn the following facts : 1) Whether a queried keyword is related to the previously queried keywords, except for the search results. 2) Whether the user's key k_u for querying is related to the previous queriers' keys except for the search results, even if the same user queries for the same keyword. Therefore, a server cannot deduce information from accumulated queries, so that SSS-II can provide **'Trapdoor Privacy'**.

Example 6. We assume $\alpha = 9$. For a keyword 'god',

$$T = (T_1,T_2,T_3) = (1,\ (35)(9),\ T_3)$$
$$T_3 = (f_{12345}(g|1|1)+9)|(f_{12345}(o|2|1)+9)|(f_{12345}(d|3|1)+9)$$

5.2.2 Verification

In this process, a server constructs the algorithms $PattGen(T,d)$ and $SimMatch(P,I)$ for approximate string matching over encrypted data.

For a keyword $W_{*j} = w^1_{*j}w^2_{*j}...w^s_{*j}$, a server generates each pattern string P_{ij} for each document D_i, receiving the trapdoor $T = (T_1,\ T_2,\ T_3(= t^1|t^2|...|t^s))$. Then, the server implements the Hamming Distance test in j-th field if the T_1 is j. Where, $1 \le i \le n$ and n is the total number of documents. P_{ij} is computed by using id_{i1} and id_{i2} at j-th field in each document D_i's index list as follows:

$$P_{ij} = HS((id_{i1})^{t^1}\cdot(id_{i2})^{T_2})|...|HS((id_{i1})^{t^s}\cdot(id_{i2})^{T_2})$$
$$= HS((g^{d_ih(k_u)})^{(f_{k_u}(w^1_{*j}|1|j)+\alpha)}\cdot(g^{-d_i})^{(h(k_u)\alpha)})|...$$
$$...|HS((g^{d_ih(k_u)})^{(f_{k_u}(w^s_{*j}|s|j)+\alpha)}\cdot(g^{-d_i})^{(h(k_u)\alpha)})$$
$$= HS((g^{d_ih(k_u)f_{k_u}(w^1_{*j}|1|j)+d_ih(k_u)\alpha-d_ih(k_u)\alpha})|...$$
$$...|HS((g^{d_ih(k_u)f_{k_u}(w^s_{*j}|s|j)+d_ih(k_u)\alpha-d_ih(k_u)\alpha})$$
$$= HS(g^{d_ih(k_u)f_{k_u}(w^1_{*j}|1|j)})|......|HS(g^{d_ih(k_u)f_{k_u}(w^s_{*j}|s|j)})$$

A server produces all pattern strings from P_{1j} to P_{nj} at the j-th field for all documents from D_1 through to D_n.

After that, a server implements Hamming Distance test between each document's index I_{ij} and the generated pattern string $P_{i,j}$ for all the documents. If the computed Hamming Distance satisfies the condition $H(P_{ij},\ I_{ij}) \le \kappa$, this index list I_i is the index list of the document D_i that a user wants.

Example 7. For the trapdoor $T = (1,\ (35)(9),\ T_3)$ of the above Example 6, a server generates from P_{11} through to P_{n1}. For simplicity, we only show P_{51}:

$$P_{51} = HS((id_{51})^{t^1}\cdot(id_{52})^{(35)(9)})|HS((id_{51})^{t^2}\cdot(id_{52})^{(35)(9)})|$$
$$HS((id_{51})^{t^3}\cdot(id_{52})^{(35)(9)})$$

$$= HS((g^{d_5(35)})^{(f_{12345}(g|1|1)+9)}\cdot(g^{-d_5})^{(35)(9)})|$$
$$HS((g^{d_5(35)})^{(f_{12345}(o|2|1)+9)}\cdot(g^{-d_5})^{(35)(9)})|$$
$$HS((g^{d_5(35)})^{(f_{12345}(d|3|1)+9)}\cdot(g^{-d_5})^{(35)(9)})$$

$$= HS((g^{(55555)(35)})^{(f_{12345}(g|1|1)+9)} \cdot (g^{-(55555)})^{(35)(9)})|$$
$$HS((g^{(55555)(35)})^{(f_{12345}(o|2|1)+9)} \cdot (g^{-(55555)})^{(35)(9)})|$$
$$HS((g^{(55555)(35)})^{(f_{12345}(d|3|1)+9)} \cdot (g^{-(55555)})^{(35)(9)})$$

$$= HS(g^{(55555)(35)f_{12345}(g|1|1)})|HS(g^{(55555)(35)f_{12345}(o|2|1)})|$$
$$HS(g^{(55555)(35)f_{12345}(d|3|1)})$$

Next, the Hamming Distance Test from I_1 through to I_n is implemented. It is also shown for only I_5 of the document D_5. $H(P_{51},\ I_{51}) = 0$. This satisfies the condition $H(P_{ij},\ I_{ij}) \leq \kappa$ if κ is 1. Hence, the document D_5 is what a user wants to retrieve.

5.2.3 Returning and Decryption

This process is the same as in SSS-I, too.

6 Security Analysis for SSS-II

The emphasis of SSS-II is on the privacy protection. It guarantees 'Perfect Similarity Search Privacy'. As with SSS-I, we prove the privacy of SSS-II by using the security game model.

Theorem 2. SSS-II can provide 'Index Privacy' according to the game ICC in a random oracle model if the DDH problem is hard.

Proof Sketch. According to [21], this theorem can be proved using the security game ICLR. We prove it by contraposition. Let A be an adversary that wins the game ICLR with advantage ε. We construct an adversary Δ, which uses A as a subroutine and breaks the DDH with non-negligible advantage. The details are shown in APPENDIX B.

Theorem 3. SSS-II can provide 'Trapdoor Privacy' according to the game ICC if G_r is (t, e)-secure pseudo random generator.

Proof Sketch. According to [21], this theorem can be also proved using the security game ICR. We prove it as well by contraposition. Assume that SSS-II cannot provide 'Trapdoor Privacy' according to the security game ICR. Then, there exists an algorithm $A(t, \varepsilon, q)$ which wins the game ICR. We construct an algorithm β' which can solve the problem of whether G_r is a pseudo random or random generator with non-negligible probability. β' can access an oracle $\mathcal{O}_\alpha$ for the unknown generator G_r. In the algorithm $Trapdoor(K, W_{*j}, j, s)$, β' substitutes the value of α through the queries to the oracle $\mathcal{O}_\alpha$. β' uses the algorithm A as a subroutine and simulates algorithm A by using the Security Game ICR.

Under the existence of an algorithm $A(t, \varepsilon, q)$, we can prove Theorem 3 by constructing an algorithm β'. Refer to APPENDIX C for a detailed proof.

Theorem 4. SSS-II can provide 'Perfect Similarity Search Privacy' if it guarantees both 'Trapdoor Privacy' and 'Index Privacy'.

By Theorem 2 and 3, it is clear that SSS-II can provide 'Perfect Similarity Search Privacy'.

7 Prototype Implementation

In most of the existing schemes, indexes for each document are stored by a row and each document needs some random factors for a high level of security. Hence, the searching process requires at least one computation for each document in every row to verify whether or not a document contains a querying keyword. There is no decryption at a server, because the server is regarded as an inside attacker. This makes it difficult for database engineers to apply database schemas into an encrypted database search system. It is one of the major reasons why efficiency is lowered. Consequently, it is true that there are hardly efficient schemes for practical business use in this 'keyword index search' area. For example, in the experiment of [5], the search of 10,000 indexes requires approximately 720 seconds (720,000 ms). The pairing function used in the paper is another reason to lower the performance. However, actually, a very small number of studies such as [34] do try to solve the problem of inefficiency. The search process of [34] is similar to a general plaintext search system because it can directly access data without verification for every row. However, it cannot guarantee even 'index privacy' and cannot provide similarity search over encrypted documents.

Since SSS is the first result that enables similarity search, a performance comparison with other schemes is not meaningful. We hence implement a prototype. In this section, we describe the experiments with regard to our proposed schemes SSS-I and SSS-II.

TABLE 2
Implementation Environment and Paramenters

Agent	Processor	Intel Core 2 Duo 2.13 GHz
	RAM	2 GB
	Language	C++
	Crypto. Eng.	OpenSSL 0.9.8e
Database	Product	Oracle 10g
	Interface	OCCI (Oracle C++ Call Interface)
Cryptographic Parameter	Curve	WTLS Curve 3
	Size	163 bits
	Hash Function	SHA-1 (160 bits)
	PRF	AES (128 bits)
Data Set	keywords (=m)	10, 20, 30
	# of MAX characters (=s)	15
	# of documents	1,000/5,000/10,000 15,000 / 20,000 / 25,000
	# Hamming distance $H(P,I) \leq \kappa$	$\kappa = 1$

7.1 Setting

Our system processes the transactions on an Intel Core 2 Duo 2.13 GHz processor with 2 GB RAM. We use Oracle 10g as the database system and OCCI (Oracle C++ Call Interface) as the interface between Oracle DBMS and the SSS client to reduce the interface latency. We use OpenSSL cryptography modules for cryptographic operations such as SHA-1, AES, and Elliptic Curve operation. Since an exponentiation calculation is very heavy and the size of a group element is generally long, we use a group over 'Elliptic Curve' to solve that problem. We use 'Koblitz curve', where the underlying field $GF(2^{163})$ is defined by generating the polynomial $x^{163} + x^7 + x^6 + x^3 + x + 1$. This curve has been used in many standards and identified in WAP WTLS standard as WTLS Curve 3. As for the degree of similarity, Hamming Distance $H(P,I) \leq \kappa$, we set κ as 1. The detailed implementation parameters are presented in TABLE 2.

7.2 Experimental Categories and Results

To evaluate the performance of SSS-I and SSS-II, we construct a data set, then estimate the time that is required for the uploading and search processes of our two schemes. First, we build two tables

TABLE 3
Uploading Time

	SSS-I			SSS-II		
m	10	20	30	10	20	30
Total Uploading Time	35.083	35.143	35.219	59.045	82.077	107.114
Document Encryption	35	35	35	35	35	35
1_{st} encryption	0.035	0.068	0.103	0.035	0.067	0.104
2_{nd} encryption	0.038	0.065	0.106	24	47	72
Other Processing Time	0.01	0.01	0.01	0.01	0.01	0.01

m-the number of keyword fields, time unit- ms

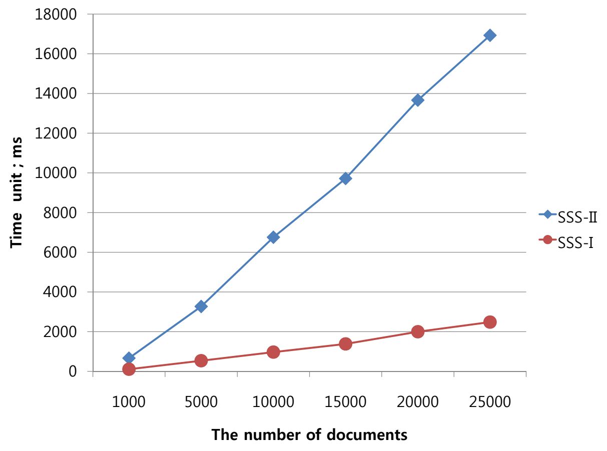

Fig. 1. Performance of the Search Process

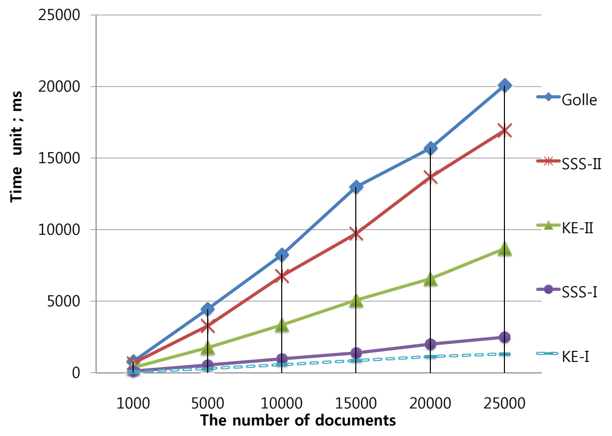

Fig. 2. Comparison with keyword-wise encryption schemes

for SSS-I and SSS-II and select random keywords by using the standard C language random function rand() where s = 15, m=10, 20, 30, and the size of a document is 1MB. Next, SSS generates the index strings $I = \{I_{ij}\}$ that are related to the keywords and then inserts the document into the database. We repeat this procedure 25,000 times, i.e., 25,000 documents are inserted into the database. We consider the case that different documents contain common keywords, so that we set for the occurrence of a common keyword and similar($\kappa \leq 1$) keywords in at least every 2,500 documents in the total of 25,000 documents. Finally, if a user requests a document with the specific keyword, the SSS client performs the search process with a server. We repeat the experiment with respect to the keyword field size and the size of data set.

In addition, we experiment on Golle *et al.*'s scheme and the modified version, SSS-IM, KE-I, and KE-II. SSS-IM is the SSS-I that uses the non-purposebuilt keyword-field different from Golle *et al.*'s [21], i.e., keyword-field free database table. KE-I and KE-II are keyword-wise – not character-wise – encrypted version of SSS-I and SSS-II. Further details are addressed below.

7.2.1 Uploading Time

First, we evaluate the uploading time for producing an index list and encrypting the document. The total uploading time consists of document encryption, the first encryption of index generation, the second encryption of index generation, and other processing times. The required time for a document(of size 1MB) is shown in TABLE 3, where m=10, 20, and 30. We find that document encryption takes up most of the total uploading time. Since SSS-II needs relatively more time in the second encryption due to exponential calculation, total uploading time is also longer.

7.2.2 Searching Time

Fig.1 shows the searching times of SSS-I and SSS-II. We evaluate each scheme by the size of data set(the number of documents). The evaluated time includes the index search time in a server and the decryption time for some returned documents because we set common and similar($\kappa \leq 1$) keywords for ten documents after every 2,500 documents. In results, for 1,000 documents, the search time is checked as 108 ms(0.108 seconds) and 665 ms(0.665 seconds) for SSS-I and SSS-II, respectively. For 25,000 documents, the serch times are 2,482 ms and 16,928 ms, respectively.

7.2.3 Comparison with keyword-wise encryption of scheme

Our schemes, SSS-I and SSS-II, encrypt character by character. We may guess that these schemes take up more time than keyword-wise encryption. We again encrypt SSS-I and SSS-II, keyword by keyword; then, all documents are verified by an equality test, not a similarity test. We call them KE-I and KE-II, i.e., keyword-wise encryption for SSS-I and SSS-II.

Originally, the encryption mechanism of SSS-I is character-wise encryption applied on pseudo random function. In this paper, we use pseudo random function as 'AES(128 bits)', which is known to have the best performance in a real world. Hence, we just compare SSS-I with KE-I. SSS-II is the character-wise encryption version of the improvement of Golle *et al.*'s scheme. To the best of our knowledge, Golle *et al.*'s scheme is the first and one of the very rare schemes in this area that can achieve 'Trapdoor Privacy', so that we additionally evaluate Golle *et al.*'s scheme as well. We compare SSS-II with KE-II and Golle *et al.*'s scheme.

Fig.2 shows the results. One of the most important aspects is that **our proposed schemes are faster than Golle *et al.*'s scheme even if our scheme's encryption method is character-wise encryption.** It is natural that KE-II is faster than Golle *et al.*'s scheme because

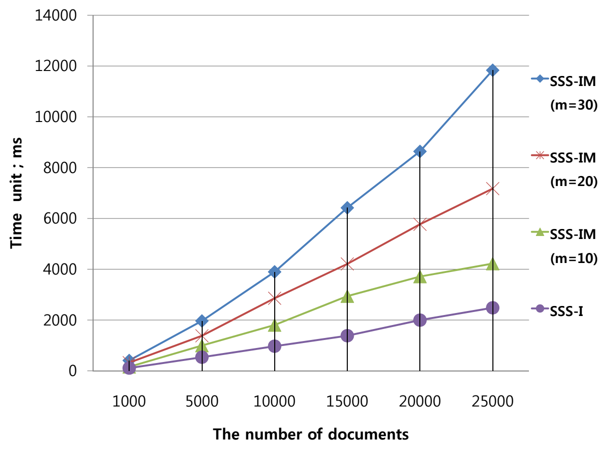

Fig. 3. Comparison with non-purposebuilt keyword field schemes

KE-II is the improved scheme of Golle *et al.*[4] However, even SSS-II, which uses character-wise encryption, is faster than Golle *et al.*'s scheme. For 10,000 documents, SSS-II took 6,756 ms (6.756 seconds), while Golle *et al.*'s scheme took 8229ms (8.229 seconds). This is because Golle *et al.*'s scheme requires too much time for updating the last fields for all documents in a server every query time. It is the reason why we improved Golle *et al.*'s scheme. The more detail reasons are addressed in Section 8.1.

Since s(the total number of characters of a keyword) is set as 15, our concern was that our proposed schemes with character-wise encryption would be 15 times as slow as keyword-wise encryption schemes, KE-I, KE-II, and Golle *et al.*'s scheme. However, as shown in Fig.2, KE-I and KE-II are only about *twice* as fast as SSS-I and SSS-II, respectively. For 10,000 documents, SSS-I and SSS-II took 969 ms (0.969 seconds) and 6,756 ms (6.756 seconds), respectively, while KE-I and KE-II take 557 ms (0.557 seconds) and 3,336 ms (3.336 seconds), respectively. It is because we set κ as 1. In the verification process for each document's Hamming Distance test, we coded that the process moves into the next document, once the number of different places exceeds one, i.e., $\kappa > 1$. This way of coding can relieve our schemes from the ramification of the Hamming Distance if the server is not malicious. It will be discussed in the Section 8.

7.2.4 Comparison with non-purposebuilt keyword field of scheme

Our schemes' index table is a purposebuilt keyword field, as in [21]. This keyword field makes it possible for a client, on behalf of a user, to query a target field and for a server to search the queried field. We experiment on another version of SSS-I, SSS-IM, whose keyword field is non-purposebuilt, i.e. keyword-field free.

SSS-IM requires all fields have to be searched for every document; hence, we evaluate it by the different numbers of keyword fields: 10, 20, and 30. As we would expect, the longest search time arises when the number of keyword fields, m, is 30. However, we did not experiment on the modified SSS-II with a non-purposebuilt keyword field because original SSS-II itself takes up much time due to exponential calculation.

7.2.5 Correctness

To check the correctness of our scheme, we make four test files. Each of them includes the keywords 'complement', 'compliment', 'complement and compliment', and 'complemente', respectively. These files are inserted into the database that is generated with a random function. First, we query 'complement' and then 'compliment'. The first result yields all the four test files and the second result yields the first three files including 'complement', 'compliment', and 'complement and compliment'. This shows the correctness of our scheme.

4. It is because SSS-II is the character-wise encryption version of the improvement of the Golle *et al.*'s

7.3 Experimental Analysis

SSS-I scheme is more efficient than SSS-II since the exponentiation calculation of SSS-II requires heavy computational overhead.

Our schemes with character-wise encryption, SSS-I and SSS-II, take only about twice as much time as the keyword-wise encryption schemes, KE-I and KE-II(which are modified versions of our schemes SSS-I and SSS-II). In particular, our schemes are faster than Golle *et al.*'s although we use character-wise encryption. In the experimental setting, if we control s and κ, the performance may be changed.

In another respect, under the search scheme with keyword-wise encryption, if we want *to search 'ab%', we have to query 26 times from 'aba' to 'abz' on general keyword-wise search system. Therefore, our scheme will be the preferred choice.*

We expect that our SSS-I and SSS-II schemes can be internally supported in the DBMS through ongoing works so that the performance of the search process can be improved for practical business use.

8 Discussion

8.1 Trapdoor Privacy

In SSS-II, α, which is a randomly generated number in the trapdoor-generation equation during each query, is removed from the pattern generation process. It has some significance. These equations enable SSS-II to avoid having to update all documents at each query, differently from in the scheme of Golle *et al.* [21]. In [21], a random number is generated during each query and all documents in a server have to be updated during each query to prohibit the inference of any information from the accumulated results. This is because a random number that is generated in a query time cannot be removed during verification by a server. However, our proposed scheme SSS-II can eliminate such an inefficient process of updating because we designed trapdoor and pattern-generation equations very pertinently, as we mentioned earlier. At the same time, SSS-II can maintain the same level of security as Golle *et al.*'s scheme. Owing to α (newly generated every query time), even if a user queries the same keyword repeatedly, the trapdoors' values are different; hence, no one knows whether this trapdoor has been queried before or not. In short, SSS-II has only to generate α at each query and does not need to update all the documents in a server for providing 'Trapdoor Privacy'. The performance of our improvement is already shown in Section 7.2.3.

8.2 The Ramification of Hamming Distance

8.2.1 Index List Table

The server can try to evaluate the Hamming Distance between the index strings in the database table, in which the server might be expected to be able to correlate some documents by similarities through the Hamming Distance. This is because the smaller value of Hamming Distance on the index list table implies more similarity; however, it is not true because of 'Index Privacy'. In our index list, the same characters have different encryption values even in the same cell (refer to the Theorem 1 and 2). Hence, the Hamming

Distance on our Index Lists does not have any meaning. In other words, the small value of the Hamming Distance on our index lists cannot guarantee the real similarity. As a result, the server's efforts are in vain.

8.2.2 Verification Process

In the verification process for the Hamming Distance test, a server can try other attacks. A malicious server may not follow the given matching test code that the process checks only until a similarity degree $\kappa \leq 1$. The server may try the test for all the characters of a targeted keyword index even if $\kappa > 1$.

This attack requires valid trapdoors, which can be considered largely in two cases as follows:

At first, we consider the case that an attacker pretends to be a valid user. An attacker tries to masquerade as a valid user or a server to make valid trapdoors, since s/he does not know other valid users' secret keys. In this case, an attacker can learn the targeted user's index information by querying all possible keywords. But, this problem falls under the user authentication and the limitation of the DAS model. Authentication is out of our research scope, so that we do not deal it with any more.

The next case is that there is neither masquerade nor conspiracy. A server implements the matching test for the queried trapdoors by valid users, not by the given coding rule(until $\kappa \leq 1$) but for all the characters. All of the results are stored, and based on them, the server may try several attacks such as a dictionary attack or a guessing attack. Although a server does not know which character would be matched to the value, he can learn 'matching' or 'not' between an index value and the querying keyword.

For example, we assume that a server receives a trapdoor $T = t_1|t_2|t_3$ for the j-th field and implements matching tests for every row. The maximal fact that the server can learn from the matching test except for the results is like this;
· the number of matching rows to t_1: x
· the number of matching rows to t_2: y
· the number of matching rows to t_3: z

We anticipate a dictionary attack based on statistical data of relative frequency of characters in English. Let n be the total number of stored documents, i.e. the total number of rows, then we can obtain the frequency of t_1, $\frac{x}{n}$. If this value is 0.12 and we know the relative frequency of an alphabet 'e' is 12% in English, can we estimate t_1 as 'e'? The answer is 'no'. The relative frequency 12% of a certain character in SSS means this; the probability is 12% in that a certain character will be matched to t_1 and the character is encrypted with these factors: the same user's secret key; the targeted field identifier; the same positional character number, together. The right answer for 'e' should be $\frac{x}{the\ number\ of\ documents\ that\ the\ user\ stores} = 0.12$.

Generally, the Caesar Cipher and Vigenere Cipher are known as easily attacked based on statistical information of 'relative frequency in English'. In these kinds of Ciphers, an attacker can classify the sentences by identical patterns through analyzing a whole ciphertext, since same plaintexts or characters have the same ciphers. For each identical pattern, an attacker obtains the relative appearance frequency, which is adaptively mapped to the well known statistical data of the relative appearance frequency in English as in TABLE 4. The process is repeated until a meaningful plaintext is encountered.

In our scheme, obtaining available relative appearance frequencies of characters is almost impossible because a character has all different encryption values over whole indexes and there is no identical pattern. If a character is to be estimated through a malicious matching test based on statistical data, the sample space of one querying field for matching test results is too small and inappropriate, whereas SSS requires many things to be matched at the same time such as a user, a character, the position of a character, a field, and a document identifier. Especially, how many documents are stored by a user is a very important factor for the more correct frequency. It is also as hard as to learn the user's secret key.

TABLE 4
The ranking of relative appearance frequency in English

Character	relative frequency(%)	relative frequency	Character	relative frequency(%)	relative frequency
E	12.702	over 12%	M	2.406	1.5% - 3%
T	9.056	6% - 9%	W	2.360	
A	8.167		F	2.228	
O	7.607		G	2.015	
I	6.966		Y	1.974	
N	6.749		P	1.929	
S	6.327		B	1.492	
H	6.094		V	0.978	below 1%
R	5.987		K	0.772	
D	4.253	4%	J	0.153	
L	4.025		X	0.150	
C	2.782		Q	0.095	
U	2.758		Z	0.074	

Beker and Piper(1982) examined the frequency distribution of the letters in English novels and newspapers.

In detail, if the matching character to t_1 is really 'e' in the user's documents, the relative frequency of the first character in j-th field of the user's documents might be 12%. In order to estimate more correctly, an adversary should know the frequencies for all alphabets in the first character's position, j-th field of the user's documents in advance. In reality, the probability is low in that all characters should be in one position within a certain field for a certain user's documents and the all characters contain all of 26 characters, a through z. A probability is much lower in that a user stores documents that should be as many as approximating to the correct frequency data for all characters statistically. Valid trapdoors for matching tests should be accumulated enough to provide correct statistical frequency data, but this probability is much lower. For example, we assume that Alice and Bob query "exam" for the jth-field; $T_A = t_a^1|t_a^2|t_a^3|t_a^4$, $T_B = t_b^1|t_b^2|t_b^3|t_b^4$. A malicious server S implements matching tests for all characters in j-th field and then S obtains the number of matching characters for the each querying character like this:
$t_a^1 : x,\ \ t_a^2 : y,\ \ t_a^3 : z,\ \ t_a^4 : u$
$t_b^1 : x',\ \ t_b^2 : y',\ \ t_b^3 : z',\ \ t_b^4 : u'$

We assume that Alice and Bob store l and m documents. If t_a^1 and t_b^1 are "e", then $\frac{x}{l} \approx \frac{x'}{m}$ would approximate 0.12.

In English documents, since "e" has a particularly high rate 12%, we may guess t_a^1 and t_b^1 as "e". However, a server cannot know the numbers, l and m, of Alice's and Bob's documents and S knows only the matching numbers x and x'. It means that S cannot compute the relative frequency of the matching character, so that S cannot know whether the frequency is high or not. If the difference of frequencies is high like between t_a^1 for "e" and t_a^2 for "x", S only knows that t_a^1 is not for z(the lowest frequency) and t_a^2 is not for e(the highest frequency), for the characters within one querying keyword. Especially, the characters with similar frequency such as a, o, i, s, h, are also not proper for the statistical data based guessing attack because the sample space is too small and there are too many factors that have to be considered.

Furthermore, statistical estimation is also meaningless in SSS, since S cannot know that T_A and T_B are the queries for different keywords by the same user or the queries by different users. In this case, S at first may assume they are the queries from the same user Alice. Then S chooses l adaptively to compute ($\frac{x}{l}$, $\frac{x'}{l}$, $\frac{y}{l}$, $\frac{y'}{l}$, $\frac{z}{l}$, $\frac{z'}{l}$, $\frac{u}{l}$, $\frac{u'}{l}$). Based on TABLE 4 and the computed results, S will try to find out some meaningful messages. In another case, if S assumes the trapdoors are the queries from different users, S also does not know who queries each of them and how many documents the queriers store. With the same method as above, S may choose l and m adaptively and will try to find out some meaningful messages again. Even if S gets some meaningful messages, it is hard for S to be confident that. Especially, this kind of estimation may be impossible in the case of the characters having similar frequency such as t,a,o,i,n,s,h,r.

However, we assume that S succeeds in guessing correctly. Although the guess is right, it cannot make any contribution to decrypt other indexes such as the Vigenere Cipher or Caesar Cipher. As we mentioned before, every character is encrypted differently and the index list has no identical pattern, because we encrypt a character with all of these factors: a user's secret key; positional number of a character; field identifier; document identifier. This is the same meaning as that all different keys are applied to each character, which plays a similar role of *one-time encryption*. Therefore, it is impossible to decrypt another index by correlating with correctly guessed characters.

In addition, if the indexes of SSS are to be decrypted, some secret values should be required. The indexes for one character of SSS-I and SSS-II are formed like this; $HS(f_{d_i}(f_{k_u}(w^s_{i,j}|s|j))$, $HS(g^{d_i h(k_u) f_{k_u}(w^s_{ij}|s|j)})$. At first, we cannot know the input values of them, $(f_{d_i}(f_{k_u}(w^s_{i,j}|s|j))$ and $(g^{d_i h(k_u) f_{k_u}(w^s_{ij}|s|j)})$ because of the one-wayness of the hash function, where we cannot know the user's secret key k_u and document identifier d_i because of the security of pseudo random function and the hardness of the DDH(Decisional Diffie-Hellman) problem. Consequently, a server S cannot know secret values to decrypt and this is so irrespective of correctly guessed results.

By these reasons, we used together all these factors: a user's secret key, the position number of a character, a field identifier, and a document identifier, for the encryption of one character. In our scheme SSS, a statistical data based attack as well as a dictionary attack are not likely to be successful.

8.3 Limitation of Similarity and Like Query

Character-wise encryption of our main method may cause some problems such as a dictionary attack. The solution for these problems is that every character is differently encrypted together with its positional number, field identifier, document identifier, and a user's secret key. Because of the character's positional number, we chose Hamming Distance as approximate string matching test. However, the Hamming Distance Test cannot provide a high level of similarity such as Edit Distance because Hamming Distance is an exact positional string matching method. For example, for a keyword 'sunflower', Hamming Distance cannot search for 'flowersun'.

However, in another respect, due to character-wise encryption, our scheme can support 'like query' over encrypted documents as long as we set $\kappa = 0$. For the keyword 'abc%', our scheme for similarity search has the following results. If $\kappa = 0$, all the possible results are $abca \sim abcz$, i.e., abc%. When $\kappa = 1$, the results are xbc%, axc%, abx%, and $\kappa = 2$, the results are xxc%, axx%, xbx%. Here, if we want like query for 'abc%', we should set $\kappa = 0$.

If we take away the positional number from our scheme, we could support every approximate string matching measurement as well as Edit Distance, while the level of security would be lower. Therefore, we need to work more on the schemes which can resolve the problems caused by character-wise encryption and at the same time support a high level of similarity.

9 Conclusion

In this socio-computing era, the storage of sensitive data is one of the most important issues in database-management [35]. Recently, radical information-communication technologies encountered a turning point with cloud computing services [36]. Many people and enterprises have utilized the storage service with the concept of DaaS (datacenter as a service), where the DAS (database as a service) model is regarded as an appropriate database model to protect users' sensitive information by encryption. However, encryption methods have some negative effects in DBMS. Although many researchers have properly treated most of the given problems, the similarity searches over encrypted documents has remained infeasible. In this paper, we open another possibility, i.e., similarity search over encrypted documents. In order to do this, we designed two similarity search schemes for encrypted documents, implemented the prototype, and analyzed the security and efficiency of the schemes. SSS-II can meet the 'Perfect Similarity Search Privacy', which we define as the best security. The SSS-I is more efficient by operating under a slightly weaker security guarantee than SSS-II.

For future work, we suggest examining how our search algorithms could be incorporated efficiently into database management systems, and investigating a search algorithm with a higher level of similarity.

References

[1] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. a Shi, "Searchable Encryption Revisited: Consistency Properties, Relation to Anonymous IBE, and Extensions", *Crypto05, LNCS 3621*, pp.205-222, 2005

[2] M. Atallah, K. Frikken, M. Goodrich, and R. Tamassia, "Secure Biometric Authentication for Weak Computational Devices", *FC 2005, LNCS 3570*, pp. 357-371, 2005

[3] R. Baeza-Yates, "Text retrieval: Theory and practice", *In 12th IFIP World Computer Congress, Elsevier Science*, vol. I, pp. 465-476, 1992

[4] L. Ballard, M. Green, B. de Medeiros, F. Monrose, "Correlation-Resistant Storage via Keyword-Searchable Encryption", *SPAR Technical Report* TR-SP-BGMM-050705

[5] L. Ballad, S. Kamara, and F. Monrose, "Achieving efficient conjunctive keyword searches over encrypted data", *ICICS 2005, LNCS 3783*, pp.414-426, 2005

[6] D. Boneh, G. Crescenzo, R. Ostrovsky, and G. Persiano, "Public-key encryption with keyword search", *Eurocrypt04, LNCS 3027*, pp. 506-522, 2004

[7] J. Bethencourt, H. Chan, A. Perrig, E. Shi, and D. Song, "Anonymous multi-attribute encryption with range query and conditional decryption", *Technical report*, CMU-CS- 06-135, 2006.

[8] M. Burmester, Y. Desmedt, R. Wright, and A. Yasinsac, "Accountable Privacy", *Security Protocols 2004*, LNCS 3957, pp. 83-95, 2006.

[9] K. Bennett, C. Grothoff, T. Horozov, I. Patrascu, "Efficient sharing of encrypted data", *ACISP02, Springer-Verlag*, pp 107-120, 2002

[10] R. Baeza-Yates and G. Navarro, "Faster approximate string matching", *Algorithmica 23*, pp. 127-158, 1999

[11] J. Byun, H. Rhee, H. Park, D. Lee, "Off-Line Keyword Guessing Attacks on Recent Keyword Search Schemes over Encrypted Data", *SDM2006, LNCS 4165*, pp.75-83, 2006.

[12] D. Boneh, B. Waters, "Conjunctive, Subset, and Range Queries on Encrypted Data", *TCC 07,LNCS 4392*, pp. 535-554, 2007.

[13] B. Chor, N. Gilboa, M. Naor, "Private Information Retrieval by Keywords", *Technical Report TR CS0917*, 1997

[14] Y.C.Chang and M.Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data", *Cryptology ePrint Archive*, 2004

[15] G. Cornode and S. Muthukrishnan, "The String Edit Distance Matching Problem With Moves", *ACM Transactions on Algorithms*, Vol. 3, No. 1, Article 2, 2007

[16] S. Chung and G. Ozsoyoglu, "Processing Aggregation Queries over Encrypted Databases", *Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDEW'06)* pp. 98-112, 2006.

[17] J. Domingo-Ferrer, "A new privacy homomorphism and applications", *Information Processing Letters*, 1996.

[18] W. Du and M. J. Atallah, "Protocols for Secure Remote Database Access with Approximate Matching", ser. Advances in Information Security. Kluwer Academic Publishers, Boston, 2001, vol. 2, pp. 192, http://www.wkap.nl/prod/b/0-7923-7399-5.

[19] E. Goh, "Secure Indexes", *Cryptology ePrint Archive*, 2004

[20] T. Ge, S. Zdonik, "Fast, Secure Encryption for Indexing in a Column-Oriented DBMS", *Proceedings of the 23nd International Conference on Data Engineering* pp.676-685, 2007.

[21] P. Golle, J. Staddon, B. Waters, "Secure Conjunctive Keyword Search Over Encrypted Data", *ACNS04, LNCS 3089*, pp. 31-45, 2004

[22] H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra, "Executing SQL over Encrypted Data in the Database-Service-Provider Model", *In the Proceedings of ACM SIGMOD,* pp. 216-227, June, 2002.

[23] H. Hacigumus, B. Iyer, and S. MehrotraEfficient, "Execution of Aggregation Queries over Encrypted Relational Databases", *DASFAA2004, LNCS 2793,* pp.125-136, 2004

[24] Y. Hwang, P. Lee, "Public Key Encryption with Conjunctive Keyword Search and Its Extension to a Multi-user System", *Pairing 2007, LNCS 4575*, pp. 2-22, 2007

[25] J. Katz, A. Sahai, B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products", *EUROCRYPT 2008, LNCS 4965*, pp.146-162, 2008

[26] S.Laur and H. Lipmaa, "On Private Similarity Search Protocols", Proceedings of the Ninth Nordic Workshop on Secure IT Systems (NordSec 2004), pp. 73-77, 2004

[27] E. Mykletun and G. Tsudik, "Aggregation Queries in the Database-As-a-Service Model", *In the proceedings of DBSEC2006, LNCS 4127* pp.89-103, 2006.

[28] G. Navarro, "A guided tour to approximate string matching", *ACM Computing Surveys 33(1)*, pp. 31-88, 2001.

[29] G. Navarro and R. Baeza-Yates, "Matchsimile: A Flexible Approximate Matching Tool for Searching Proper Names", *Journal of the American society for Information Science and Technology*, Volume 54 , Issue 1, pp. 3- 15, 2003

[30] W. Ogata and K. Kurosawa, "Oblivious Keyword Search", *Journal of Complexity,* Volume 20, pp.356 - 371, 2004

[31] H.H. Pang and J. Shen, "Privacy-Preserving Similarity-Based Text Retrieval",*ACM Transaction on Internet Technology*, Vol 10, No 1, Article 4, 39pages, 2010

[32] D. Park, K. Kim, and P. Lee, "Public Key Encryption with Conjunctive Field Keyword Search", *WISA 2004, LNCS 3325*, pp.73-86, 2004

[33] H. Park, J. Byun, and D. Lee, "Secure Index Search for Groups", *TrustBus 2005, LNCS3592* pp.128-140, 2005

[34] H. Park, J. Zhan, G. Blosser and D. Lee, "Efficient Keyword Index Search over Encrypted Data of Groups", *ISI2008, IEEE Computer Society Press*, pp.225-229, 2008

[35] H. Park, J. Hong, J. Park, J. Zhan and D. Lee, "Combined Authentication based Multi-Level Access Control in Mobile Application for DailyLifeService", *IEEE Transactions on Mobile Computing, IEEE Computer Society Press*, 9(6), pp. 824-837, 2010

[36] H. Park, J. Park, J. Choi, and D. Lee, "Toward an Integrated System between Cloud Computing and Smartcard Application", *ICCIT 2010, IEEE Computer Society Press*, pp. 2010

[37] R. Sion and B. Carbunar, "Conjunctive keyword search on encrypted data with completeness and computational privacy", *Cryptology ePrint Archive,* 2005

[38] D. Sankoff, and J. Kruskal, "TimeWarps, String Edits, and Macromolecules", *The Theory and Practice of Sequence Comparison. Addison- Wesley*, 1983.

[39] E. Shi, B. Waters, "Delegating capabilities in predicate encryption systems", *ICALP 2008, LNCS 5126*, pp.560-578, 2008

[40] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data", *IEEE Symposium on Security and Privacy,* pp.44-55, 2000

[41] A. Takasu, "An Approximate Multi-word Matching Algorithm for Robust Document Retrieval", *CIKM06, Proceedings of the 15th ACM international CIKM* pp. 34-42, 2006

[42] B. Waters, D. Balfanz, G. Durfee, and D. Smetters, "Building an encrypted and searchable audit log", *NDSS04, The Internet Society* pp.205-214, 2004

[43] P. Wang, H. Wang, and J. Pieprzyk, "Threshold Privacy Preserving Keyword Searches", *SOFSEM 2008, LNCS 4910*, pp. 646-658, 2008

[44] P. Wang, H. Wang, and J. Pieprzyk, "Keyword Field-free Conjunctive Keyword Searches on Encrypted Data and Extension for Dynamic Groups", *CANS 2008, LNCS*, 2008

[45] Ontario, Office of the Information and Privacy Commissioner (IPC) and Netherlands Registratiekamer (1995), Privacy-Enhancing Technologies: The Path to Anonymity, Information and Privacy Commissioner and Registratiekamer, at: http://www.ipc.on.ca/english/pubpres/papers/anon-e.htm

APPENDIX A
THE PROOF OF THEOREM 1.

- **Setup.** Algorithm β creates a set WS of q words $\in \{D_i\}(1 \leq i \leq n)$ and gives this to the adversary A. A chooses a polynomial number of subsets $\{W_i\}$ from WS. We call this the collection of subsets $WS^* = \{W_i\}$, where $W_i = \{W_{i1}, W_{i2}, .., W_{im}\}$, $W_{ij} = w^1_{ij}|w^2_{ij}|...|w^s_{ij}$, and $1 \leq j \leq m$. A sends them to β again. Upon receiving WS^*, β runs algorithm SysParam and KeyGen and invokes algorithm IndGen for keywords set $W_i \in WS^*$. Where an index list I_i is produced, an unique identifier d_i for each document D_i is assigned. After producing all the index strings, β sends them to A.
- **Queries.** If A queries the trapdoor for a word $W_{ij} \in W_i$, β runs algorithm Trapdoor for a word W_{ij} and produces $T_{W_{ij}}$ and sends it to A.
- **Challenge.** After making some queries, A chooses a challenge keyword set $W_* = \{W_{*1}, W_{*2}, ..., W_{*m}\}$, a querying keyword $W_{*j} \in W_*$, and a character $w^t_{*j} \in W_{*j}$. A's challenge is to distinguish whether it is encrypted with $W0$ or $W1$ in the game ICLR, where the querying keyword's field is j. Then, β guesses a value w^z_{*j} for the character w^t_{*j} is encrypted by pseudorandom function or random function, by picking w^z_{*j} uniformly independently at random in $\{W_{*1}, W_{*2}, ..., W_{*m}\}$. If $w^z_{*j} \neq w^t_{*,j}$, β returns the value of random function in reply to the pseudorandom or random challenge.
  With the probability of about $\frac{1}{l}$(l is the total number of all characters in the keyword set W_*), we have $w^z_{*j} = w^t_{*j}$ and that case β proceeds as follows. Let $I^t_{*j} = HS(f_{d_*}(f^t_{*j}))$. For $W_{*j} = w^1_{*j}|w^2_{*j}|...|w^s_{*j}$, $w^c_{*j} \neq w^t_{*j}$, let $I^c_{*j} = \mathrm{g}^c_{*j}$, where g is a random function and $c \in \{1, 2, ..., s\}$. For $W_{*y} \in W_* = \{W_{*1}, W_{*2}, ..., W_{*m}\}$, $y \neq j$, let $I_{*y} = HS(f_{d_*}(f^1_{*y}))|...|HS(f_{d_*}(f^s_{*y}))$. This means that the keywords W_{*y} except for the querying keyword W_{*j} are encrypted according to the given protocol. β returns to A the following index list; $I_* = d_*, I_{*1}, I_{*2}, ..., I_{*m}$, where $I_{*j} = \mathrm{g}^1_{*j}|\mathrm{g}^2_{*j}|...|HS(f_{d_*}(f^t_{*y}))|...|\mathrm{g}^s_{*j}$.
  A checks that this index list is an encryption of W_* in every keyword W_{*y}, $y \neq j$. If f is a pseudorandom function, this is also an encryption of a character w^t_{*j} in W_*, otherwise it is not.
  A is again allowed to ask for more index of keyword sets with the restriction that A must not make queries that are distinguishing $W0 = Rand(W_*, W_{*j} - \{w^t_{*j}\})$, $W1 = Rand(W_*, W_{*j})$.
- **Response.** A finally outputs a bit b', representing its guess for b. $b' = 1$ means that the function for w^t_{*j} is random function and $b' = 0$ is pseudorandom function. If $b = b'$, β outputs 1 indicating that β guesses f is a pseudorandom function, otherwise 0. β can know the pseudorandom function challenge with the same advantage that A has in winning game ICLR.

We show that β can solve the problem about whether f is pseudo random or random function with non-negligible probability. Accordingly, the advantage of β in winning this experiment is;

$$Adv_\beta = Pr[Exp^{PR}_\beta = 1] = Pr[b' = b]$$

$$= Pr[b' = b|b = 1] \cdot Pr[b = 1] + Pr[b' = b|b = 0] \cdot Pr[b = 0]$$

$$= Pr[b' = b|b = 1] \cdot \frac{1}{2} + Pr[b' = b|b = 0] \cdot \frac{1}{2}$$

$$= Pr[b' = 1|b = 1] \cdot \frac{1}{2} + Pr[b' = 0|b = 0] \cdot \frac{1}{2}$$

$$= Pr[b' = 1|b = 1] \cdot \frac{1}{2} + (1 - Pr[b' = 1|b = 0]) \cdot \frac{1}{2}$$

$$= \frac{1}{2} + \frac{1}{2}(Pr[Exp_A^{ind-cka-1} = 1] - Pr[Exp_A^{ind-cka-0} = 1])$$
$$= \frac{1}{2} + \frac{1}{2}Adv_A^{ind-cka} = \frac{1}{2} + \frac{1}{2}\varepsilon$$

We showed the existence of β which can solve whether f is pseudorandom or random with the probability more than $\frac{1}{2}$. Accordingly, by contradiction, it can be said that SSS-I provide 'Index Privacy'.

APPENDIX B THE PROOF OF THEOREM 2.

- **Setup.** Algorithm Δ creates a set WS of q words $\in \{D_i\}(1 \leq i \leq n)$ and gives this to the adversary A. A chooses a polynomial number of subsets $\{W_i\}$ from WS. We call this the collection of subsets $WS^* = \{W_i\}$, where $W_i = \{W_{i1}, W_{i2}, .., W_{im}\}$, $W_{ij} = w_{ij}^1|w_{ij}^2|...|w_{ij}^s$, and $1 \leq j \leq m$. A sends them to Δ again. Δ invokes the algorithms SysParam and KeyGen. Let $g^\delta, g^\tau, g^\gamma$ be a Diffi-Hellman triplet, the challenge is to determine whether $\gamma = \delta\tau$. Δ guesses a value w_{ij}^z for the character w_{ij}^t that A will choose in the game ICLR, by picking w_{ij}^z uniformly independently at random in $\{W_{i1}, W_{i2}, ..., W_{im}\}$.
  Δ simulates the algorithm IndGen on W_i as follows: Δ maps every keyword $W_{ij} = w_{ij}^1 w_{ij}^2 ... w_{ij}^s$ to a random value $X_{ij} = x_{ij}^1 x_{ij}^2 ... x_{ij}^s$. For simplicity, Δ maps g^K to g'. For $W_i = \{W_{i1}, W_{i2}, ..., W_{im}\}$, Δ chooses a random value δ_i and outputs:

$$\begin{aligned} I_i = (&g^{\delta_i K},\ \ g^{-\delta_i},\ \ HS(g^{\delta_i x_{i1}^1 K})|..|HS(g^{\delta_i x_{i1}^s K}), ...\\ &..., HS(g^{\delta_i x_{ij}^1 K})|..|HS((g^\tau)^{\delta_i x_{ij}^t K})|..|HS(g^{\delta_i x_{ij}^s K}), ...\\ &..., HS(g^{\delta_i x_{im}^1 K})|..|HS(g^{\delta_i x_{im}^s K})\\ = (&(g')^{\delta_i},\ \ g^{-\delta_i},\ \ HS((g')^{\delta_i x_{i1}^1})|..|HS((g')^{\delta_i x_{i1}^s}), ...\\ &..., HS((g')^{\delta_i x_{ij}^1})|..|HS(((g')^\tau)^{\delta_i x_{ij}^t})|..|HS((g')^{\delta_i x_{ij}^1}), ...\\ &..., HS((g')^{\delta_i x_{im}^1})|..|HS((g')^{\delta_i x_{im}^s})) \end{aligned}$$

- **Queries.** If A queries for a keyword $W = w^1|w^2|..|w^s$, Δ outputs the trapdoor $T_w = (T_1,\ T_2,\ T_3 = (t^1,\ t^2, .., t^s))$ including random number α. For verification, A generates P_{ij}; $P_{ij} = ((g')^{\delta_i})^{t^1} \cdot (g^{-\delta_i})^{T_2}|...|((g')^{\delta_i})^{t^s} \cdot (g^{-\delta_i})^{T_2}$. Then, A asks Δ character-wise hash values for P_{ij} like this; $HS(((g')^{\delta_i})^{t^1} \cdot (g^{-\delta_i})^{T_2})|...|HS(((g')^{\delta_i})^{t^s} \cdot (g^{-\delta_i})^{T_2})$.
  Δ knows if P_{ij}(of D_i) satisfies SimMatch. If it is satisfied, Δ returns this; $P_{ij} = HS(((g')^{\delta_i})^{t^1} \cdot (g^{-\delta_i})^{T_2})|...|HS(((g')^{\delta_i})^{t^s} \cdot (g^{-\delta_i})^{T_2})$. If not, Δ returns character-wise random values for P_{ij}.
- **Challenge.** Finally, A selects a challenge keyword set $W_* = \{W_{*i}, .., W_{*m}\}$ for document D_* along with a querying keyword $W_{*j} \in W_*$ and a character $w_{*j}^t \in W_{*j}$, where $j \in \{1, 2, .., m\}$ and $t \in \{1, 2, .., s\}$ are selected randomly.
  If $w_{ij}^z \neq w_{ij}^t$, Δ returns a random value in reply to the DDH challenge. With the probability of about $1/l(l$ is the total number of all characters in the keyword set W_*), we have $w_{ij}^z = w_{ij}^t$ and that case Δ proceeds as follows. Let $I_{*j}^t = HS((g')^{\gamma \cdot x_{*j}^t}) = HS(((g')^\tau)^{\delta_i x_{*j}^t})$. It is the setting for the DDH triplet. For $W_{*j} = w_{*j}^1|..|w_{*j}^s$, $w_{*j}^c \neq w_{*j}^t$, let $I_{*j}^c = R_{*j}^c$(random value), where $c \in \{1, 2, .., s\}$. For $W_{*y} \in W_* = \{W_{*i}, .., W_{*m}\}$, $y \neq j$, let $I_{*y} = HS((g')^{\delta_* x_{*y}^1})|...|HS((g')^{\delta_* x_{*y}^s})$. This means that the keywords except for querying keyword W_{*j} are encrypted according to the given protocol. Δ returns to A the following index list; $I_* = ((g')^{\delta_*}, g^{-\delta_*}, I_{*1}, .., I_{*m})$. Where, $I_{*j} = (R_{*j}^1,\ R_{*j}^2,\ , .., HS((g')^{\gamma \cdot x_{*j}^t}), .., R_{*j}^s)$.
  A checks that this index is an encryption of W_* in every keyword W_{*y}, $y \neq j$. If $\gamma = \delta\tau$, this index is also an encryption of W_* for a character w_{*j}^t; otherwise it is not.
  A is again allowed to ask for index of keyword sets, with the restriction that A must not make a query that are distinguishing $W0 = Rand(W_*, W_{*j} - \{w_{*j}^t\})$ from $W1 = Rand(W_*, W_{*j})$. Where $W0$ means that it is a query with a DDH triplet and W_1 is a query without a DDH triplet.
- **Response.** Finally, A outputs a bit b'. If $b' = 0$, Δ guesses that $g^\delta, g^\tau, g^\gamma$ is a DDH triplet. If $b' = 1$, Δ guesses that $g^\delta, g^\tau, g^\gamma$ is not a DDH triplet. Since the encryption will be random for character w_{ij}^c if and only if the challenge is not a DDH tuple, Δ solves the DDH challenge with the same advantage that A has in winning game ICLR.

APPENDIX C THE PROOF OF THEOREM 3.

- **Setup.** Algorithm β' and adversary A do the same things as *Setup* process of Theorem 1.
- **Queries.** This process is also the same as Theorem 1.
- **Challenge.** After making some queries, A chooses a challenge keyword set $W_* = \{W_{*1}, W_{*2}, ...W_{*m}\}$ for D_*, a querying keyword $W_{*j}(= W0) \in W_*$, and a character $w_{*j}^t \in W_{*j} = w_{*j}^1|w_{*j}^2|...|w_{*j}^s$, where $1 \leq j \leq m$ and $1 \leq t \leq s$. A's challenge is to distinguish whether the trapdoor is generated with $W0 = W_{*j}$ or $W1 = Rand(W_{*j}, w_{*j}^t)$ in the game ICR. β' guesses a value $t_{*j}^z = f(w_{*j}^z) + \alpha$ for the character w_{*j}^z is produced by α through pseudo random generator or random generator, by picking w_{*j}^z uniformly independently at random in $W_{*j} = w_{*j}^1|w_{*j}^2|...|w_{*j}^s$.
  If $z \neq t$, β' returns $t_{*j}^z = f(w_{*j}^z) + G_r(\alpha)$, where $G_r(\alpha)$ is the value α of pseudo random generator in reply to the pseudo random or random challenge. With the probability of about $\frac{1}{s}$, we have $z = t$, and that case β' returns $t_{*j}^z = f(w_{*j}^z) + R(\alpha)$ of random generator. This means that the trapdoor T_{*j} (for the keyword W_{*j}) except for the w_{*j}^t is generated according to the given protocol. β' returns to A the trapdoor $T_{*j} = f(w_{*j}^1) + G_r(\alpha)|f(w_{*j}^2) + G_r(\alpha)|..|f(w_{*j}^t) + R(\alpha)|..|f(w_{*j}^s) + G_r(\alpha)$.
  A checks that this trapdoor is produced by pseudo random generator for the keyword W_{*j}.
  A is allowed to ask for more trapdoors with restriction that A must not make queries that are distinguishing $W0 = W_{*j}$, $W1 = Rand(W_{*j}, w_{*j}^t)$.
- **Response.** A finally outputs a bit b', representing its guess for b. $b' = 1$ means that the generator for w_{*j}^t is random generator and $b' = 0$ is pseudo random generator. If $b = b'$, β' outputs 1. Otherwise, β' outputs 0. 'β' outputs 1' means that G_r is a pseudo random generator.

β' can solve the problem about whether G_r is pseudo random or random generator with the probability non-negligibly different from $\frac{1}{2}$. The advantage of β' in winning this experiment can be obtained through the similar process to Theorem 1. By constructing β' like this, we can say that SSS-II can provide 'Trapdoor Privacy'.