Toward Practical Searchable Symmetric Encryption

Wakaha Ogata[1], Keita Koiwa[2], Akira Kanaoka[3], and Shin'ichiro Matsuo[4]

[1] Tokyo Institute of Technology, Japan
ogata.w.aa@m.titech.ac.jp
[2] University of Tsukuba, Japan
koiwa@cipher.risk.tsukuba.ac.jp
[3] Toho University, Japan
akira.kanaoka@is.sci.toho-u.ac.jp
[4] National Institute of Information and Communications Technology, Japan
smatsuo@nict.go.jp

Abstract. Searchable symmetric encryption is a good building block toward ensuring privacy preserving keyword searches in a cloud computing environment. This area has recently attracted a great deal of attention and a large quantity of research has been conducted. A security protocol generally faces a trade-off between security/privacy requirements and efficiency. Existing works aim to achieve the highest levels of security requirements, so they also come with high overhead. In this paper, we reconsider the security/privacy requirements for searchable symmetric encryption and relax the requirements for practical use. Then, we propose schemes suitable for the new requirements. We also show experimental results of our schemes and comparison to existing schemes. The results show that the index sizes of our proposals are only a few times of that of a Lucene (without encryption). In document update, our proposal requests additional index which depends only on the size of new document.

1 Introduction

1.1 Background

In the last several years, the progress of network technology and computers, including broadband network and virtualization techniques, has made information technology (IT) environments more usable. The proliferation of cloud computing is a good example of this. Though cloud computing provides such a usable environment, its characteristics pose security issues since valuable information is stored and processed in uncontrollable locations for users, and this could be lead to information leakage by cloud operators.

Encrypting data stored in the cloud is considered to be a countermeasure for such threats, and a large number of studies have been conducted on this subject. In this research, data is encrypted in a manner that it can proceed in its encrypted form. Examples of such research are counting by using a homomorphic encryption scheme, processing any calculation using a homomorphic encryption scheme, and searchable encryption schemes, and these fall within the scope of this paper.

In each scheme, security requirements are defined and a scheme with provable security is proposed. These security requirements include privacy of user requests for the cloud server as well as confidentiality of information. Since privacy requirements vary among entities, we have not provided effective and general security requirements for the scheme. However, security researchers generally tend to aim at stronger security requirements. Yet security requirements and efficiency have trade-off relationship in our hopes for strong security, and processing performance and communication efficiency decrease. We therefore have to find a good balance among usability in terms of cloud computing, security, privacy, and efficiency.

1.2 Related Works

In this research, we focus on security requirements for a searchable encryption scheme, which is a demanded service for cloud computing and has been extensively researched to date. In searchable encryption schemes, the data and keyword for the search are first encrypted. Ciphertext is stored in the server. Only a party possessing access right can produce valid information (trapdoor) for a keyword search. The server cannot know the keyword from the trapdoor. This characteristic protects the privacy of the keyword.

Searchable encryption based on a symmetric cipher was firstly proposed in [17], and then schemes with improved security definitions were proposed in [8,7]. In, [7] Curtmola et al. proposes two searchable symmetric encryption schemes. Then, Kamara and Roeder showed that the scheme can convert to secure against adaptive adversary in CCS 2012 [14]. To enhance efficiency for keyword search, researches on reducing cost for document update are conducted recently. In [10,11], new indexes are reconstructed based on a single private key. On the other hand, the scheme proposed in [14] does not construct additional indexes.

Boneh et al. first proposed searchable encryption based on a symmetric cipher [5] as an application of an identity-based encryption scheme. Following that, many schemes [1,4] have been proposed including those based on anonymous hierarchical identity-based encryption. There is also research on operation when a searchable encryption scheme is applied to cloud computing [12].

1.3 Our Contributions

In this paper, we focus on searchable symmetric encryption. We refine the security definitions that offer a good balance between privacy and efficiency.

We first show efficiency requirements for a practical searchable symmetric encryption scheme, and show that the existing scheme is not practical. After that, we reconsider the security requirement for searchable encryption. Next we propose new schemes that have smaller encrypted indexes and lower processing costs for adding documents. These schemes allow leakage of a part of privacy from search history, but this would not be a problem in most of practical usages. We also show experimental results of our schemes and existing schemes. The experimental results show that the original searchable symmetric encryption by Curtmola et al. has huge amount of index size against Lucene, and our proposal can reduce the index size to a few times of that of Lucene.

2 Definitions of Symmetric Searchable Encryption and Existing Schemes

2.1 System Model

We assume the following setting as in [7]: There is one user $\mathcal{U}$ and one server $\mathcal{S}$. $\mathcal{U}$ has a collection of documents $\mathcal{D} = \{D_1, \ldots, D_n\}$, each document D_j is stored on server $\mathcal{S}$ in an encrypted style. D_j is assigned a unique identifier $id(D_j)$ that does not reveal any confidential information, e.g., a sequential number $(id(D_j) = j)$ or a ciphertext of the document name. We assume that the set of searchable keywords, $\Delta = \{w_1, \ldots, w_d\}$, is predetermined and is called a *dictionary*. An outcome of a search for $w \in \Delta$ is denoted by $\mathcal{D}(w) = \{id(D_j) \mid w \in D_j\}$.

In an ordinary file system (with no security or privacy), a database called an *index* is generated in advance for quick keyword searching. For example, $\{(w_i, \mathcal{D}(w_i))\}_{i=1,\ldots,d}$ is stored. When a user issues a search query to the file system, the file system searches $\mathcal{D}(w_i)$ in the database and returns it to the user. A symmetric searchable encryption system (SSE) is a system in which an encrypted index is built to prevent information leakage.

An SSE consists of four algorithms as follows.

$\mathsf{Keygen}(1^k)$**:** User $\mathcal{U}$ uses this algorithm to generate private key K based on security parameter k.

$\mathsf{BuildIndex}(K, \mathcal{D}, \Delta)$**:** $\mathcal{U}$ uses this algorithm to build (encrypted) index $\mathcal{I}$ from document set $\mathcal{D}$. $\mathcal{I}$ is sent to server $\mathcal{S}$ along with encrypted documents $\zeta = (Enc(D_1), \ldots, Enc(D_n))$.

$\mathsf{Trapdoor}(K, w)$**:** $\mathcal{U}$ runs this algorithm when it searches in $\mathcal{D}$ for keyword w. The output $T_w = \mathsf{Trapdoor}(K, w)$, called a *trapdoor*, is sent to $\mathcal{S}$.

$\mathsf{Search}(\mathcal{I}, T)$**:** $\mathcal{S}$ uses this algorithm to search in encrypted documents. If $T = \mathsf{Trapdoor}(K, w)$, then it is necessary that $\mathsf{Search}(\mathcal{I}, T) = \mathcal{D}(w)$. $\mathcal{S}$ returns the result $\mathcal{D}(w)$ to $\mathcal{U}$.

Although the search process in this model is a one-round protocol, it can generally be a multi-round protocol.

2.2 Security Requirement

Let $(w_1, \ldots, w_q)$ be a sequence of q keywords. A *history* is defined as

$$H_q = (\mathcal{D}, w_1, \ldots, w_q),$$

which determines an instantiation of an interaction between $\mathcal{U}$ and $\mathcal{S}$. A *partial history* of H_q is $H_q^t(\mathcal{D}, w_1, \ldots, w_t)$, where $t \leq q$. An adversary's *view* of H_q under secret key K is defined as

$$V_K(H_q) = (id(D_1), \ldots, id(D_n), \zeta, \mathcal{I}, T_1, \ldots, T_q),$$

where $\mathcal{I} = \mathsf{BuildIndex}(K, \mathcal{D}, \Delta)$ and $T_i = \mathsf{Trapdoor}(K, w_i)$. A *partial view* is

$$V_K^t(H_q) = (id(D_1), \ldots, id(D_n), \zeta, \mathcal{I}, T_1, \ldots, T_t),$$

where $t \leq q$.

Oblivious RAMs introduced in [9] realize secure searching in which $V_K(H_q)$ does not leak any information of H_q. However, this scheme is highly inefficient. It is not practical to require perfect secrecy such as with oblivious RAMs. Thus, some weak security definitions that allow leakage of partial information of the history to the server were defined in the literature.

Chang and Mitzenmacher [6] defined the security of SSEs. In [7], a vulnerability of the definition was pointed out, and the authors gave four new security definitions: semantic security against non-adaptive attack, semantic security against adaptive attack, indistinguishability against non-adaptive attack, and indistinguishability against adaptive attack. Since equivalence of semantic security and indistinguishability was shown [7], here we give only the definition of semantic security.

Definition 1 (Trace). *For a given history $H_q = (\mathcal{D}, w_1, \ldots, w_q)$, let Π_q be a $q \times q$ binary matrix where $\Pi_q[i,j] = 1$ if $w_i = w_j$, $\Pi_q[i,j] = 0$ otherwise. The* trace *of H_q is the sequence*

$$Tr(H_q) = (id(D_1), \ldots, id(D_n), |D_1|, \ldots, |D_n|, \mathcal{D}(w_1), \ldots, \mathcal{D}(w_q), \Pi_q).$$

Trace indicates information that we allow to leak to the server.

Definition 2 (Semantic security against non-adaptive attack). *We say that an SSE is non-adaptively semantically secure if all $q \in N$ and for any ppt adversary $\mathcal{A}$, there exists a ppt simulator Sim such that for all traces Tr_q, all polynomial samplable distributions $\mathcal{H}_q$ over $\{H_q \in 2^{2^\Delta} \times \Delta^q : Tr(H_q) = Tr_q\}$, all functions f,*

$$|\Pr[\mathcal{A}(V_K(H_q)) = f(H_q)] - \Pr[Sim(Tr(H_q)) = f(H_q)]|$$

is negligibly small, where $H_q \leftarrow \mathcal{H}_q, K \leftarrow \mathsf{Keygen}(1^k)$, and the probabilities are taken over $\mathcal{H}_q$ and the internal coins of Keygen*, $\mathcal{A}$, Sim and the underlying* BuildIndex *algorithm.*

Definition 3 (Semantic security against adaptive attack). *We say that SSE is adaptively semantically secure if all $q \in N$ and for all ppt adversaries $\mathcal{A}$, there exists a ppt simulator Sim such that for all traces Tr_q, all polynomial* *samplable distributions* $\mathcal{H}_q$ *over* $\{H_q \in 2^{2^{\Delta}} \times \Delta^q : Tr(H_q) = Tr_q\}$*, all functions* f*, all* $0 \leq t \leq q$*:*

$$|\Pr[\mathcal{A}(V_K^t(H_q)) = f(H_q^t)] - \Pr[Sim(Tr(H_q^t)) = f(H_q^t)]|$$

is negligibly small, where $H_q \leftarrow \mathcal{H}_q, K \leftarrow \mathsf{Keygen}(1^k)$*, and the probabilities are taken over* $\mathcal{H}_q$ *and the internal coins of* Keygen*,* $\mathcal{A}$*, Sim and the underlying* BuildIndex *algorithm.*

2.3 Curtmola et al. Scheme (SSE-1)

Existing SSE schemes can be classified into two types. The first type uses a Boolean $n \times d$ matrix as an index (such as [6]), and the second type uses a list of $(w_i, \mathcal{D}(w_i))$ (such as [7]). We focus on the second type of scheme in this paper because computational complexity of searching in such schemes is $O(\log n)$, while it is $O(n)$ in the first type of scheme.

Curtmola et al. proposed two SSE schemes [7]. The first, called SSE-1, is secure against non-adaptive attacks, and is more efficient. The second, called SSE-2, is secure against adaptive attacks, but less efficient. On the other hand, in [14] it is shown that simple modification of SSE-1 can make it adaptively secure in the random oracle model. We review the SSE-1 scheme here.

Let k be a security parameter, p be the bit length of the longest keyword in Δ, *unit* be the bit length of the shortest keyword, and m be the total size of the plaintext documents $\mathcal{D}$ expressed in *unit*. Let $\mathcal{E}$ be a symmetric encryption function with key length ℓ. SSE-1 uses the following pseudo-random function f and pseudo-random permutations π, ψ.

- $f : \{0,1\}^k \times \{0,1\}^p \to \{0,1\}^{\ell+\log m}$
- $\pi : \{0,1\}^k \times \{0,1\}^p \to \{0,1\}^p$
- $\psi : \{0,1\}^k \times \{0,1\}^{\log m} \to \{0,1\}^{\log m}$

We give a list of parameters in Table 1 for convenience.

Keygen(1^k)**:** Generate random keys $s, y, z \xleftarrow{R} \{0,1\}^k$ and output $K = (s, y, z, 1^\ell)$.
BuildIndex$(K, \mathcal{D}, \Delta)$**:**

Table 1. Parameters used in SSE-1

k	security parameter, key length of pseudo-random function/permutations
ℓ	key length of symmetric encryption
n	the number of documents in document collection $\mathcal{D}$
d	the number of keywords in dictionary Δ
p	bit length of the longest keyword
unit	bit length of the shortest keyword
m	total length of $\mathcal{D}$ expressed by *unit*

1. Scan $\mathcal{D}$ and build $\Delta'(\subseteq \Delta)$, which is the set of all keywords in $\mathcal{D}$. Build $\mathcal{D}(w)$ for each word $w \in \Delta'$.
2. Set up array A with m entries as follows. First, global counter ctr is set to 1. For each $w_i \in \Delta'$ choose a random ℓ-bit string $\kappa_{i,0}$, and for each $id_{i,j} \in \mathcal{D}(w_i)$ $(1 \le j \le |\mathcal{D}(w_i)|)$, set node $N_{i,j} = \langle id_{i,j} || \kappa_{i,j} || \psi_s(ctr+1)\rangle$, where $\kappa_{i,j}$ is a random ℓ-bit string.
   Compute $\mathcal{E}_{\kappa_{i,j-1}}(N_{i,j})$ and store it in $\mathsf{A}[\psi_s(ctr)] = \mathcal{E}_{\kappa_{i,j-1}}(N_{i,j})$.
   Store a random string in all entries that are not used to store an encrypted node.
3. Build lookup table T with d entries as follows.
   For each $w_i \in \Delta'$, set $\mathsf{T}[\pi_z(w_i)] = \langle \mathrm{addr}(\mathsf{A}(N_{i,1}))||\kappa_{i,0}\rangle \oplus f_y(w_i)$, where $\mathrm{addr}(\mathsf{A}(N_{i,1}))$ is the address add where $\mathsf{A}[add] = \mathcal{E}_{\kappa_{i,0}}(N_{i,1})$.
   Store a random string in all $\mathsf{T}[\pi_z(w_i)]$ s.t. $w_i \in \Delta \setminus \Delta'$.

Output $\mathcal{I} = (\mathsf{A}, \mathsf{T})$.

$\mathsf{Trapdoor}(K, w)$**:** Output $T_w = (\pi_z(w), f_y(w))$.

$\mathsf{Search}(\mathcal{I}, T)$**:** Let $T = (\gamma, \eta)$. Retrieve $\theta = \mathsf{T}[\gamma]$. Let $\langle \alpha || \kappa\rangle = \theta \oplus \eta$. Decrypt $\mathsf{A}[\alpha]$ with κ to obtain $N_{i,1}$, which includes identifier $id_{i,1}$, the next random key $\kappa_{i,1}$, and the next address $\mathrm{addr}(\mathsf{A}(N_{i,2}))$. Then decrypt $\mathsf{A}[\mathrm{addr}(\mathsf{A}(N_{i,2}))]$ with $\kappa_{i,1}$ to obtain $N_{i,2}$, which includes $id_{i,2}$. Iterating the same process to recover all $id_{i,j}$. Output all identifiers $\{id_{i,j}\}$.

It is shown that SSE-1 is semantically secure against non-adaptive attacks, if $\mathcal{E}$ is a secure symmetric encryption function, f is a pseudo-random function, and π, ψ are pseudo-random permutations.

2.4 Other Schemes Supporting Document Update

Consider the case that the user $\mathcal{U}$ keeps a set of documents $\mathcal{D}_1$ on a server $\mathcal{S}$ with an index $\mathcal{I} = \mathsf{BuildIndex}(K, \mathcal{D}_1, \Delta)$, and now $\mathcal{U}$ is going to store an additional set of documents $\mathcal{D}_2$. A simple way to add documents is that $\mathcal{U}$ builds a new index $\mathcal{I}' = \mathsf{BuildIndex}(K, \mathcal{D}_2, \Delta)$ and $\mathcal{S}$ replaces an old index $\mathcal{I}$ with $(\mathcal{I}, \mathcal{I}')$. In this case, however, $\mathcal{S}$ learns $\mathcal{D}_2(w)$ if $\mathcal{U}$ already made a search query for w in $\mathcal{D}_1$ (but not in $\mathcal{D}_2$) since $\mathcal{S}$ knows $T = \mathsf{Trapdoor}(K, w)$.

Accordingly, the following process is adopted in [6] and [7]. To add $\mathcal{D}_2$, $\mathcal{U}$ runs Keygen to generate a new key K', and builds a new index $\mathcal{I}' = \mathsf{BuildIndex}(K', \mathcal{D}_2, \Delta)$, which is sent to $\mathcal{S}$ with (encrypted) $\mathcal{D}_2$. When $\mathcal{U}$ wants to search for a keyword w, it sends two trapdoors $T = \mathsf{Trapdoor}(K, w)$ and $T' = \mathsf{Trapdoor}(K', w)$ to $\mathcal{S}$. $\mathcal{S}$ runs $\mathsf{Search}(\mathcal{I}, T)$ and $\mathsf{Search}(\mathcal{I}', T')$.

This process does not leak unnecessary information. However, if a few documents are added frequently, $\mathcal{U}$ has to keep many private keys and a set of many trapdoors has to be sent to search for a keyword.

Recently, some researchers have proposed SSE schemes in which the user can add document sets freely without increasing the size of the private key. SSE schemes proposed in [10] and [11] construct new index $\mathcal{I}'$ based on a single private key. On the other hand, the scheme proposed in [14] does not construct additional indexes but utilizes unused memory space of the original index.

3 What Is Practical SSE?

SSE schemes with high security are needed in special purposes. In most cases, however, we require practicality – reasonable index size and small communication/computational cost – rather than security, since we use storage services as a tool for improving convenience.

In this section, we first introduce requirements for practical SSE schemes. We then claim that existing schemes such as SSE-1 do not satisfy the requirements.

3.1 Requirements for Practicality

Here we introduce three requirements for practicality.

1. **Efficient search.** We perform keyword search repeatedly, so real-time response is required. We require that a much longer time than in an ordinary (unencrypted) system is not needed to search for a keyword.
2. **Reasonable index size.** In general, the size of an index depends on the total size of $\mathcal{D}$. We require that the size of index for $\mathcal{D}$ is not much larger than $\mathcal{D}$ itself.
3. **Scalability.** In most cases, new documents are added in storage one after another. On such occasions, the user must renew the index by performing an update protocol with the server. For scalability, it is desirable for an SSE scheme to have the following two properties.
   (R1) The size of secret key K and computation/communication cost for searching do not depend on the number of updates of the index.
   (R2) The computational cost to update the index depends on the additional document size, but not on the total document size.

3.2 Inefficiency of SSE-1

In SSE-1, index $\mathcal{I}$ consists of an array A and a lookup table T. A has m entries, where m is the total size of the plaintext documents $\mathcal{D}$ expressed in the shortest keyword length. Each entry consists of three parts: a document identifier, a random key, and the next address of A. Since the lengths of these parts are $\lceil \log n \rceil$, ℓ, $\lceil \log m \rceil$, respectively, the total size of A is $m(\lceil \log n \rceil + \ell + \lceil \log m \rceil)$ bits. T has d entries, each consisting of an address and a value, which are p bits and $(\ell + \lceil \log m \rceil)$ bits, respectively. Therefore, the size of T is $d(\ell + \lceil \log m \rceil + p)$. In total, the bit length of the index is $|\mathcal{I}| = m(\lceil \log n \rceil + \ell + \lceil \log m \rceil) + d(\ell + \lceil \log m \rceil + p)$.

Next, we estimate the sizes of $\mathcal{I}$ for concrete parameters. We assume that

– $\mathcal{D}$ consists of $n = 10^3$ documents, the size of each document is on average 10KB. The total size of $\mathcal{D}$ is 10MB.
– The dictionary includes 100,000 keywords, that is, $d = 10^5$. The length of the shortest keyword is 2 B (two letters) and that of the longest keyword is 20 B (20 letters, 10 units). Therefore, $p = 160$.

Table 2. Index size in SSE-1 and Lucene in a case$^{(*)}$

	size of document set: $\|\mathcal{D}\|$	index size: $\|\mathcal{I}\|$	Ratio: $\|\mathcal{I}\|/\|\mathcal{D}\|$
SSE-1	10MB	836MB	83.6
Lucene	67MB	83MB	1.28

$(*)$ $\mathcal{D}$ consists of $n = 10^3$ documents, the size of each document is on average 10KB.
The dictionary includes 100,000 keywords,
The shortest keyword is two letters, the longest keyword is 20 letters (10units).
The private key length is $\ell = 128$.

– The key length is $\ell = 128$ as in AES.

Under these parameters, $m = 5 \times 10^6$. We show the index size under these parameters in Table 2. For comparison, we also give the case of Lucene [2] as an example of systems that do not consider privacy at all.

From the table, we can see that the index is huge in SSE-1. Now consider the case in which we store documents using a free storage service. If the free space is 5 GB, we can store 2 GB (non-confidential) documents in total, along with a 2.6 BG index of Lucene. On the other hand, if we want to store them by using SSE-1, we cannot store 60 MB of documents in total since their index exceeds 5 GB.

3.3 Scalability of Existing SSE

As we mentioned before, SSE-1 does not have scalability.

In contrast, SSE schemes that support document updates have scalability. However, they require huge indexes as well as SSE-1.

4 Relaxation of Security

As we show in section 3.2, a serious disadvantage of SSE-1 (and its variations) is index size, especially the size of array A. A has m entries, but only $m'(= \sum_{w \in \Delta'} |\mathcal{D}(w)|)$ entries are used to store meaningful values. The remaining entries are prepared to hide the number m'. This means that if the user does not mind the server knowing the number m', the number of entries of A can be reduced to $m'(\ll m)$. Similarly, there is a possibility that a rather efficient SSE scheme can be constructed if the user does not mind leakage of some additional information.

In this section, we discuss the need for adaptive indistinguishability and define several levels of security.

4.1 Adaptive Attack

An adversary that mounts a chosen-keyword attack (cka) has the ability to obtain trapdoors corresponding to the keywords. We discuss the feasibility of cka.

The general attack scenario of active attacks such as chosen-keyword attacks and chosen-ciphertext attacks is a lunchtime attack. That is, an adversary illegitimately accesses a computer that is used to make trapdoors. Do we possess other means against the attack other than cryptographical control? Yes, we will be able to avoid such illegitimate use by adequately managing a private key K.

Another attack scenario of a chosen-keyword attack is a social attack, as follows.

- An adversary popularizes a target keyword w. Accordingly, the user would search for w in his documents by sending a trapdoor $T = \mathsf{Trapdoor}(K, w)$.
- A malicious administrator of the server tells the user a forged notification that word w is not allowed to be stored in storage (e.g., for certain political reasons). Accordingly, the user searches for w in his documents.

Although it is difficult to avoid such social attacks, we think that an adversary cannot frequently succeed in obtaining desirable trapdoors. It seems particularly hard to adaptively obtain desirable trapdoors.

From the above discussion, if adaptively indistinguishable SSE schemes are much more inefficient than non-adaptively indistinguishable ones, one practical choice is to use an efficient non-adaptively indistinguishable one together with appropriate key management and other controls against social attacks.

4.2 Relaxed Security Definitions

In [7], a trace of history H_q is defined as

$$Tr(H_q) = (id(D_1), \ldots, id(D_n), |D_1|, \ldots, |D_n|, \mathcal{D}(w_1), \ldots, \mathcal{D}(w_q), \Pi_q).$$

As mentioned before, $Tr(H_q)$ indicates partial information of H_q that we allow to leak to the server. Below, we define some variations of trace.

For given dictionary $\Delta = \{w_1, \ldots, w_d\}$ with d words and document set $\mathcal{D} = \{D_1, \ldots, D_n\}$, we define *index matrix* P which is expressed by a binary matrix:

$$P = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,n} \\ \vdots & \vdots & \cdots & \vdots \\ p_{d,1} & p_{d,2} & \cdots & p_{d,n} \end{bmatrix},$$

$$p_{i,j} = \begin{cases} 1 & \text{if } w_i \in D_j, \\ 0 & \text{otherwise.} \end{cases}$$

Let

$$W_H(w_i) = \sum_{j=1}^{n} p_{i,j},$$

$$W_H(D_j) = \sum_{i=1}^{d} p_{i,j},$$

$$W_H(P) = \sum_{i=1}^{d} \sum_{j=1}^{n} p_{i,j}.$$

$W_H(w_i)$ is the number of documents in $\mathcal{D}$ that include keyword $w_i (\in \Delta)$, that is, $W_H(w_i) = |\mathcal{D}(w_i)|$. $W_H(D_j)$ is the number of keywords in document D_j. For randomly chosen permutations π_d (over $\{1, \ldots, d\}$), let $\hat{P}$ be a binary matrix such that rows of P are permuted by π_d. We call $\hat{P}$ a randomized index matrix.[1]

Our new security definitions are as follows.

Definition 4. *For a given history $H_q = (\mathcal{D}, w_1, \ldots, w_q)$, define*

$$\begin{aligned}
Tr^{(0)}(H_q) &= Tr(H_q),\\
Tr^{(1)}(H_q) &= (Tr(H_q), W_H(P)),\\
Tr^{(2)}(H_q) &= (Tr(H_q), W_H(D_1), \ldots, W_H(D_n)),\\
Tr^{(3)}(H_q) &= (Tr(H_q), W_H(w_1), \ldots, W_H(w_d)),\\
Tr^{(4)}(H_q) &= (Tr(H_q), \hat{P}).
\end{aligned}$$

(In this definition, we only consider non-adaptive semantic security.) For $k \in \{0, 1, 2, 3, 4\}$, $Tr^{(k)}$-security is defined the same way as in Def. 2 except that $Tr(H_q)$ is replaced with $Tr^{(k)}(H_q)$.

From the definition, $Tr^{(0)}$-security is equivalent to the original semantic security. Clearly,

$$Tr^{(0)}\text{-secure} \Rightarrow Tr^{(1)}\text{-secure} \Rightarrow Tr^{(2)}\text{-secure and } Tr^{(3)}\text{-secure},$$

$$Tr^{(2)}\text{-secure or } Tr^{(3)}\text{-secure} \Rightarrow Tr^{(4)}\text{-secure}$$

hold.

When a user searches for a keyword, the server learns a vector in the randomized index matrix $\hat{P}$ even if the scheme has $Tr^{(0)}$-semantic security. Therefore, after the user searches for all keywords in the dictionary, the server learns the entire $\hat{P}$. This means that $Tr^{(0)}$-semantic security gets closer to $Tr^{(4)}$-security the more keywords are searched.

In the next subsection, we further discuss the relation among the security notions focusing on document update.

4.3 Relations among the Security Notions

We assume that an SSE scheme has $Tr^{(3)}$-semantic security, that is, $\mathcal{I}$ leaks $W_H(w_i) = \sum_{j=1}^{n} p_{i,j}$ for all $w_i \in \Delta$. Consider the case that the user add a new document D_{n+1} and the index is replaced with $\mathcal{I}'$ that has information about D_{n+1}. At this moment, the server learns $(p_{1,n+1}, \ldots, p_{d,n+1})$ since

$$\sum_{j=1}^{n+1} p_{i,j} - \sum_{j=1}^{n} p_{i,j} = p_{i,n+1}$$

[1] Note that the order of documents is not randomized, since it does not have any confidential information.

holds. If documents are added one by one, the server learns the entire $\hat{P}$ (even if no keyword is queried). This situation happens independently of the scheme and the way of index update. Therefore, in the case documents are added one by one (or only a few at a time), $Tr^{(3)}$-semantic security is very close to $Tr^{(4)}$-semantic security. With the same argument, $Tr^{(1)}$-semantic security and $Tr^{(2)}$-semantic security is also very close in such a situation.

5 Practical SSE Schemes

In this section, we show how we can improve the efficiency of SSE schemes by relaxing the security requirement. For this purpose, two efficient SSE schemes are given.

5.1 Simplest Scheme (Simple-SSE)

Before showing SSE schemes, we describe a search scheme with no security measure— SEARCH. In SEARCH, an index is built as $\mathcal{I}_0 = \{(w_i, \mathcal{D}(w_i))\}_{i=1,...,d}$ from document set $\mathcal{D}$ beforehand. (We assume that the entries in $\mathcal{I}_0$ are sorted in alphabetical order.) When a user requests a search for $w \in \Delta$, the server finds an entry $(w_i = w, \mathcal{D}(w_i))$ in $\mathcal{I}_0$ (with $O(\log d)$ computational cost), and answers $\mathcal{D}(w_i)$ to the user.

Needless to say, SEARCH is absolutely insecure since the server knows which keywords are included with which documents, and also learns which keywords the user searched for. By replacing all keywords with random strings we can obtain an SSE scheme which we call Simple-SSE. The description is as follows. In this scheme, $H : \{0,1\}^* \to \{0,1\}^{\ell_H}$ is a collision resistance hash function.

$\mathsf{Keygen}(1^k)$**:** Choose $K \xleftarrow{R} \{0,1\}^k$ and output K.
$\mathsf{BuildIndex}(K, \mathcal{D}, \Delta)$**:** Build $\mathcal{I}_0 = \{(w_i, \mathcal{D}(w_i))\}_{i=1,...,d}$. For each $w_i \in \Delta$ compute $\hat{w}_i = H(K \| w_i)$. Replace each entry $(w_i, \mathcal{D}(w_i))$ of $\mathcal{I}_0$ with $(\hat{w}_i, \mathcal{D}(w_i))$, and then sort the entries in alphabetical order of $\hat{w}_i$. The result is $\mathcal{I}$.
$\mathsf{Trapdoor}(K, w)$**:** Output $\hat{w} = H(K \| w)$.
$\mathsf{Search}(\mathcal{I}, T)$**:** Search $(\hat{w} = T, \mathcal{D}(w))$ in $\mathcal{I}$ and output $\mathcal{D}(w)$.

Theorem 1. *If a pseudo-random encryption function Enc is used to encrypt each document, Simple-SSE has $Tr^{(4)}$-semantic security in the random oracle model. More precisely,*

$$|\Pr(A(V_K(H_q)) = f(H_q)) - \Pr(Sim(Tr^{(4)}(H_q)) = f(H_q))| \le q_H/2^k + Adv_{Enc}$$

holds, where q_H is the number of oracle queries, k is the private key length, and Adv_{Enc} is an advantage of pseudo-randomness of Enc.

Proof. We consider Sim as follows. The input of Sim is

$$Tr^{(4)}(H_q) = (id(D_1), ..., id(D_n), |D_1|, ..., |D_n|, \mathcal{D}(w_1), ..., \mathcal{D}(w_q), \Pi_q, \hat{P}),$$

where $\hat{P} = \{p_{ij}\}$. Sim computes $\mathcal{I}$ as follows.

1) For all $i (1 \leq i \leq d)$,
   1a) choose a random string $\hat{w}$ with length ℓ_H bits and set $List \leftarrow \{\}$;
   1b) for all $j (1 \leq j \leq n)$, if $p_{ij} = 1$, add $id(D_j)$ to $List$;
   1c) set $Entry_i = (\hat{w}, List)$;
2) Sort d entries in alphabetical order of $\hat{w}$ to obtain $\mathcal{I}$.

Next, Sim computes the list of trapdoors as follows.
For all $i (1 \leq i \leq q)$,

1) search in $\mathcal{I}$ and find an entry $(\hat{w}_j, List_j)$ such that $List_j = \mathcal{D}(w_i)$;
2) set $T_i \leftarrow \hat{w}_j$.

Then, Sim runs adversary A as a subroutine with input

$$view = (id(D_1), \ldots, id(D_n), \zeta, \mathcal{I}, T_1, \ldots, T_q),$$

where ζ is a set of random strings and each length is determined by $|D_i|$.

The adversary A may issue random oracle queries. To answer them, Sim chooses random key K^* at first, and initializes a list $L_H = \emptyset$. When A makes query $(K||w)$, Sim first checks if $K = K^*$. If so, Sim aborts. Otherwise, Sim searches $K||w$ in L_H. If there exists $\langle K||w, \hat{w}\rangle$, then Sim returns $\hat{w}$ to A. Otherwise, chooses ℓ_H-bit random string $\hat{w}$, adds $\langle K||w, \hat{w}\rangle$ in L_H, and returns $\hat{w}$ to A.

When A outputs $f(H_q)$, Sim outputs it as own result.

If Sim does not abort, A's view is the same as the real attack scenario except the distribution of ζ; it consists of random strings in the above simulation, while real ciphertexts in the real attack scenario. Therefore,

$$|\Pr(A(V_K(H_q)) = f(H_q)) - \Pr(Sim(Tr^{(4)}(H_q)) = f(H_q))| \leq \Pr(Sim \text{ aborts}) + \mathrm{Adv}_{Enc}.$$

Sim aborts only if A queries K^*. So, $\Pr(Sim \text{ aborts}) \leq q_H/2^k$, where q_H is the number of oracle queries. That is, Simple-SSE holds $\mathrm{Tr}^{(4)}$-semantic security. □

The computational costs for searching in Simple-SSE are almost the same as those in SEARCH. The computational costs of BuildIndex are d hashes and sorting, which are very lightweight.

We can therefore say that the extra computational cost needed to guarantee $Tr^{(4)}$-semantic security is very small.

5.2 Lightened SSE-1 (SSE-1′)

We consider a lightened version of SSE-1, called SSE-1′, in which A has only $m' (= W_H(P))$ entries, that is, we eliminate all entries that store random strings.

Theorem 2. *SSE-1′ has $Tr^{(1)}$-semantic security, if $\mathcal{E}$ is a secure symmetric encryption function, f is a pseudo-random function, and π, ψ are pseudo-random permutations.*

Proof. SSE-1$'$ is the same as SSE-1 except the size of array A. In the security proof of SSE-1, Sim simulates adversary's view, which includes encrypted data, index (T, A), and trapdoors. (A has m entries, and m is determined by the sizes of documents.)

We consider a simulator Sim' that operates in the same way to Sim except that A has only m' entries. Note that Sim' knows $m' = W_H(P)$ because it is included in $Tr^{(1)}(H_q)$ (but not in $Tr(H_q)$). Then, the simulated view by Sim' and the real view are indistinguishable from the same reason in the proof of original SSE-1. □

5.3 Index Size of Proposed Schemes

To show the efficiency of our schemes, we first compare the size of index in SEARCH, Simple-SSE, and SSE-1$'$. We denote them with $\mathcal{I}_0$ and $\mathcal{I}_{\text{Simple}}$ and $\mathcal{I}_{\text{SSE1}'}$. In the following discussion, we use $\rho = W_H(P)/dn$, which is the average hit rate.

Since $\mathcal{I}_0 = \{(w_i, \mathcal{D}(w_i))\}_{i=1,\ldots,d}$,

$$|\mathcal{I}_0| = \sum_{i=1}^{d} |w_i| + W_H(P)\lceil \log n \rceil \;=\; d(ave(|w_i|) + n\rho\lceil \log n \rceil)$$

where d and n are the number of keywords in Δ and the number of documents in $\mathcal{D}$, respectively.

$|\mathcal{I}_{\text{Simple}}|$ is estimated as

$$|\mathcal{I}_{\text{Simple}}| = d\ell_H + W_H(P)\lceil \log n \rceil \;=\; d(\ell_H + n\rho\lceil \log n \rceil).$$

$|\mathcal{I}_{\text{SSE1}'}|$ is estimated as

$$\begin{aligned}|\mathcal{I}_{\text{SSE1}'}| &= W_H(P)(\lceil \log n \rceil + \ell + \lceil \log W_H(P) \rceil) + d(\ell + \lceil \log W_H(P) \rceil + p) \\ &= d\left((\ell + p) + n\rho(\lceil \log n \rceil + \ell + \frac{n\rho + 1}{n\rho}\lceil \log dn\rho \rceil)\right)\end{aligned}$$

If $\ell_H = 160$ (as in SHA-1) and it is longer than the average length of keywords, $\mathcal{I}$ in Simple-SSE is larger than $\mathcal{I}_0$. However, the difference between them is not so large.

Next, we compare $|\mathcal{I}_{\text{SSE1}'}|$ and $|\mathcal{I}_{\text{Simple}}|$. Assuming that $\ell_h = 160, \ell = 128, p = 160$, the first term of $|\mathcal{I}_{\text{SSE1}'}|$ is not as large as twice the first term of $|\mathcal{I}_{\text{Simple}}|$. The ratio of the second terms is $1 + (\log n)^{-1}(\ell + \frac{n\rho+1}{n\rho}\log dn\rho)$, which is $1 + (\log n)^{-1}(\ell + 2\log d)$ when $n\rho \approx 1$ and $2 + (\log n)^{-1}(\ell + \log d\rho)$ when $n\rho \gg 1$. When $n = 2^{10} \sim 2^{20}$, $\rho = 2^{-4} \sim 2^{-10}$, $d = 2^{10} \sim 2^{20}$, and $\ell = 128$, it is estimated between 8 and 18.

5.4 Scalability of Simple-SSE and SSE-1$'$

In Simple-SSE, we can update the index as follows.

- Let $\mathcal{D}'$ be the additional documents. $\mathcal{U}$ first builds $\mathcal{I}'_0 = \{(w_i, \mathcal{D}'(w_i))\}_{i=1,\ldots,d}$. For each $w_i \in \Delta$, computes $\hat{w}_i = H(K \| w_i)$ as in $\mathsf{BuildIndex}$, replaces w_i in $\mathcal{I}'_0$ which $\hat{w}_i$, and then sorts $(\hat{w}_i, \mathcal{D}'(w_i))$ to obtain $\mathcal{I}'$. $\mathcal{U}$ sends $\mathcal{I}'$ to $\mathcal{S}$ along with ciphertext of $\mathcal{D}'$.
- Upon receiving $\mathcal{I}'$, $\mathcal{S}$ updates $\mathcal{I}$ as follows. For all $(\hat{w}_i, \mathcal{D}'(w_i)) \in \mathcal{I}'$, replaces $(\hat{w}_i, \mathcal{D}(w_i))$ in $\mathcal{I}$ with $(\hat{w}_i, \mathcal{D}(w_i) \cup \mathcal{D}'(w_i))$.

After an update of the index, an original private key K can be used and the user can search as in the same process as before. Therefore, the update satisfies (R1). Computation and communication costs for an update of the index are also proportional to the size of $\mathcal{I}'$. The size of $\mathcal{I}'$ depends on the bit length of new documents $\mathcal{D}'$, but not on the existing document set. So, the update satisfies (R2). That is, we can say that Simple-SSE has scalability.

On the other hand, updating of index in SSE-1$'$ can be done similar way to [11] to satisfy scalability.[2] (Unfortunately, Hirano et al.'s technique [10] does not satisfy (R1); the technique introduced in [14] leaks additional information and degrades security.)

6 Implementation and Evaluation of SSE

To confirm that the new schemes are practical, we implement SSE-1, Simple-SSE, and SSE-1$'$, and evaluate the index size and execution time of the search for each scheme. We use Java for implementation of each program.

6.1 Preparation of Implementation

Document set $\mathcal{D}$: We use the following presented papers (total of 974) as documents of the targeted search.

- USENIX Security Symposium (2002–2011)
- IEEE Symposium on Security and Privacy (2003–2012)
- ACM Conference on Computer and Communications Security (2002–2011)

Since these papers are published on the Web by the PDF file, we convert them into text files.[3] The sum total of the size of converted documents is 65,011,003 B.

Dictionary Δ: The dictionary in our implementation is $\Delta = \Delta_1 \cup \Delta_2$, where Δ_1 is SINGLE.TXT on Moby Word Lists[16], and Δ_2 was produced by Lucene

[2] In [11], the following techniques are used: (a) To keep the key size to be constant, a secret key used to make each index is generated from a unique master secret key. (b) To keep the trapdoor size to be constant, all indexes are linked by putting a trapdoor in the next index.

[3] We use the pdftotext command of Xpdf 3.03 for conversion to text files.

from $\mathcal{D}$.[4] This dictionary has 514,045 words, that is, $d = 514,045$. The longest words in Δ is 248 letters[5], i.e., the bit length is 1,984 bit.

Other Parameters: For implementing a pseudo-random function f, we use HMAC which is in the javax.crypto package and javax.crypto.spec package. More precisely, $f_k(w)$ is computed by

$$f_k(w) = \mathrm{HMAC}(w\|0)\|\mathrm{HMAC}(w\|1)\| \cdots \|\mathrm{HMAC}(w\|(s-1)),$$

where $s = \lceil n/160 \rceil$ and n is the output length of f.

We implement pseudo-random permutations ψ and π by using AES[3]. The input length of ψ is $\log m$ in SSE-1 and $\log m'$ in SSE-1$'$, which are less than the block length of AES, 128. However, the input length of π, 1,984 bit, is much longer than 128. Therefore, we adopt the ECB-mode[6], considering a word as a 16-block plaintext.

We also adopt AES as the symmetric encryption $\mathcal{E}$.

In π, ψ and $\mathcal{E}$, the shortest key length, $\ell = 128$, is used.

6.2 Execution Environment

We measure execution time via a machine with the following specifications.

- OS: Linux 2.6.35 x86_64, Ubuntu server 10.10
- CPU: Intel Core i7 2600
- Memory: DDR3-1333 SDRAM 4GB $\times$ 2
- Software: JRE 1.6.0_29

6.3 Numerical Results

Index Size: Table 3 shows the index sizes of SSE-1, SSE-1$'$, and Simple-SSE. As a comparison with the case in which privacy protection is not taken into consideration, we also measured the size of the index using StandardAnalyzer in Lucene[2].This table also shows the size comparison.

Table 3 shows that the index becomes large as compared with original documents or the index of Lucene. However, Simple-SSE and SSE-1$'$ have succeeded in drastic reduction of the size of the index as compared with SSE-1.

[4] Here we use Lucene only to create a set of words from a targeted file set.

[5] Such a long word is because the documents include numerical data and binary data. If we exclude such long (pseudo)words, the index sizes in SSE-1 and SSE-1$'$ would become 100MB smaller than our results.

[6] AES-ECB is a permutation but not pseudo-random. Therefore, we have to adopt other implementation to satisfy the security definition. Though evaluation time of π increases by this change, it is thought that the increment does not affect searching time so much since π is evaluated only once in a search.

Table 3. Comparison of index size

	Index size: $\|\mathcal{I}\|$	$\|\mathcal{I}\|/\|\mathcal{D}\|$	Ratio to SSE-1	Ratio to Lucene
SSE-1	1,836MB	28.24	(1.00)	22.12
SSE-1′	397MB	6.10	0.22	4.78
Simple-SSE	275MB	4.23	0.15	3.32
Lucene	83MB	1.28	–	(1.00)

Table 4. Execution time of Search

	Execution time of Search (msec)		
	words in $\Delta - \Delta'$	words in Δ'	random character string
SSE-1	0.0941	0.8383	0.0817
Simple-SSE	0.0603	0.6406	0.0602
SSE-1′	0.0603	0.7534	0.0609

Search Time: We measured each execution time of Search in order to evaluate the performance of SSE. Since the execution time of Search may depend on the number of search results, we measured it by classifying keywords into the following three cases.

– Words in $\Delta - \Delta'$: Words that can be searched although not contained in documents of a targeted search. The number of search results is zero.
– Words in Δ': Words contained in documents of the targeted search. The number of search results changes in accordance with words.
– Random character string: Words that cannot be searched. The number of search results is zero. Here, we make 1,000 random character strings of 16 characters.

Table 4 shows the results in execution time very small in all schemes. This means that the measures for privacy protection do not have a bad influence on efficiency.

From the evaluation results concerning SSE-1, Simple-SSE, and SSE-1′, we can say that both Simple-SSE and SSE-1′ satisfy the objectives of "efficient search" and "reasonable index size" mentioned in section 3.

7 Conclusion

In this paper, we reconsidered the balance between efficiency and security/ privacy of a searchable symmetric encryption scheme.

By excluding consideration of active attacks, we proposed light searchable symmetric encryption schemes. We showed that they have some leakage, but this would pose no problems in most of practical cases.

We also showed experimental results of our scheme and comparison with existing schemes. The result showed that the index sizes in our schemes are only a few times of that of a general search engine (without encryption). Thus, our schemes are sufficiently secure and efficient enough for practical use.

References

1. Abdalla, M., et al.: Searchable Encryption Revisited: Consistency Properties, Relation to Anonymous IBE, and Extensions. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 205–222. Springer, Heidelberg (2005)
2. Apache Lucene - Welcome to Apache Lucene, `http://lucene.apache.org/`
3. Announcing the ADVANCED ENCRYPTION STANDARD (AES). Federal Information Processing Standards Publication 197 (November 2001)
4. Bellare, M., Boldyreva, A., O'Neill, A.: Deterministic and Efficiently Searchable Encryption. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 535–552. Springer, Heidelberg (2007)
5. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer, Heidelberg (2004)
6. Chang, Y.-C., Mitzenmacher, M.: Privacy Preserving Keyword Searches on Remote Encrypted Data. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 442–455. Springer, Heidelberg (2005)
7. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable Symmetric Encryption: Improved Definitions and Efficient constructions. In: ACM Conference on Computer and Communications Security (CCS 2006), pp. 79–88. ACM, New York (2006)
8. Goh, E.-J.: Secure indexes. Technical Report 2003/216, IACR ePrint Cryptography Archive (2003)
9. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. J. of the ACM 43(3), 431–473 (1996)
10. Hirano, T., Mori, T., Hattori, M., Ito, T., Matsuda, N., Kawai, Y., Sakai, Y., Ohta, K.: Security Notions for Searchable Symmetric Encryption with Extra Multiple Documents. In: The 29th Symposium on Cryptography and Information Security, SCIS 2012, 2B3-1 (2012) (in Japanese)
11. Iwanami, J., Ogata, W.: Secure and Efficient Searchable Symmetric Encryption with Document Addition. In: The 30th Symposium on Cryptography and Information Security, SCIS 2013, 3A3-1 (2013) (in Japanese)
12. Kamara, S., Lauter, K.: Cryptographic Cloud Storage. In: Sion, R., Curtmola, R., Dietrich, S., Kiayias, A., Miret, J.M., Sako, K., Sebé, F. (eds.) FC 2010 Workshops. LNCS, vol. 6054, pp. 136–149. Springer, Heidelberg (2010)
13. Kamara, S., Papamanthou, C., Roeder, T.: CS2: A searchable cryptographic cloud storage system. MSR Tech Report no. MSR-TR-2011-58. Microsoft, Redmond (2011)
14. Kamara, S., Roeder, T.: Dynamic Searchable Symmetric Encryption. In: Proc. of the 2012 ACM Conference on Computer and Communications Security, pp. 965–976. ACM, New York (2012)
15. van Liesdonk, P., Sedghi, S., Doumen, J., Hartel, P., Jonker, W.: Computationally efficient searchable symmetric encryption. In: Jonker, W., Petković, M. (eds.) SDM 2010. LNCS, vol. 6358, pp. 87–100. Springer, Heidelberg (2010)
16. Moby Word Lists, The Institute for Language, Speech and Hearing, `http://icon.shef.ac.uk/Moby/`
17. Song, D., Wagner, D., Perrig, A.: Practical techniques for searching on encrypted data. In: Proc. of 2000 IEEE Symposium on Security and Privacy, pp. 44–55 (2000)