# Computationally Efficient Searchable Symmetric Encryption

Peter van Liesdonk[1], Saeed Sedghi[2], Jeroen Doumen[2], Pieter Hartel[2], and Willem Jonker[2]

[1] Technical University of Eindhoven
[2] University of Twente

**Abstract.** Searchable encryption is a technique that allows a client to store documents on a server in encrypted form. Stored documents can be retrieved selectively while revealing as little information as possible to the server. In the symmetric searchable encryption domain, the storage and the retrieval are performed by the same client. Most conventional searchable encryption schemes suffer from two disadvantages. First, searching the stored documents takes time linear in the size of the database, and/or uses heavy arithmetic operations. Secondly, the existing schemes do not consider adaptive attackers; a search-query will reveal information even about documents stored in the future. If they do consider this, it is at a significant cost to the performance of updates. In this paper we propose a novel symmetric searchable encryption scheme that offers searching at constant time in the number of unique keywords stored on the server. We present two variants of the basic scheme which differ in the efficiency of search and storage. We show how each scheme could be used in a personal health record system.

## 1 Introduction

Searchable encryption is a technique that provides functionalities to retrieve encrypted documents from a (honest but curious) server selectively while revealing as little information as possible to the server. In case the retrieving and storing encrypted documents is performed by the same client, the searchable encryption is in the symmetric setting. Searchable encryption has many applications, particularly where client privacy is a main concern such as in E-mail servers [4] (public setting), keeping medical information of a client [18] (public and symmetric setting), storing private videos and photos, and backup applications [17] (symmetric setting).

Our work is motivated by the development of personal health care systems. Nowadays, keeping medical records is shifting from paper-based systems to digital record systems. Personal health record (PHR) systems which are initiated and maintained by an individual, are examples of digital record systems. An example of a PHR system is Google Health which offers a client the ability to store her medical records on Googles servers, and allows a general practitioner (GP) to get access to the medical records of her patients. Unlike paper-based systems,

W. Jonker and M. Petković (Eds.): SDM 2010, LNCS 6358, pp. 87–100, 2010.

where the privacy is mainly protected by chaos (since it is almost impossible to locate an individuals record from a multitude of providers) the PHR server might get information about the medical record of the individual after storing or retrieving the record. One way to protect the privacy of the clients is to use a searchable encryption scheme such that i) the medical records are stored in encrypted form, ii) the key used to encrypt the record is kept secret from the server, and iii) the record can be retrieved efficiently and securely.

We call this privacy enhanced PHR, which uses searchable encryption, PHR$^+$. Typical usage scenarios are i) a GP who uses it to retrieve the record of each patient before a visit and who updates the record afterwards, ii) a traveler who uses PHR$^+$ to get access to her medical record anywhere she prefers. In these examples, the reason that PHR$^+$ is used instead of PHR is that using PHR$^+$ the client can store the medical to any honest but curious server (e.g. Google server). Hence, trusting the server is not needed and the client can store the medical records more freely.

We will focus on a setting where a single symmetric key is used for encryption and searching. In more complicated usage scenarios this key can be distributed by means of existing key distribution or access control schemes.

**Problem.**Existing searchable encryption schemes offer a search algorithm which takes time linear in the number of the documents stored. There are some schemes which allow for a more efficient search, but updating the database is inefficient. Therefore, the problem is to have a searchable encryption scheme that allow efficient search and update.

**Contribution.** In this paper we propose a novel searchable symmetric encryption scheme that offers efficient searching of the documents stored on the server. Our scheme supports searching time logarithmic in the number of the unique keywords stored on the server, and the client can alter the content of the documents stored while the server learns as little as possible about the alteration. We propose two variants of the scheme proposed which differ in the efficiency of the search and the update operation.

In comparison to [10,6], our scheme deals with adaptive security while having a much lower computational complexity. In comparison with [9], our scheme achieve approximately the same. However, where that scheme the update very impractical it is possible in our scheme; we do not give a security proof however.

The rest of the paper is organized as follows: Section 2 describes the related work in this field. In section 3 we describe the problem with the conventional searchable encryption schemes. In section 4 we describe the background and the security definitions. Our approach and the two variants of the approach are presented in section 5. The appropriate applications of the schemes is described in section 6 and the conclusion is followed in section 7.

## 2 Related Work

In theory, the classical work of Goldreich and Ostrovsky [12] on oblivious RAMs can resolve the problem of doing private searches on remote encrypted data.

Their scheme is asymptotically efficient and nearly optimal, but does not appear to be efficient in practice as very large constants are hidden in a big-O notation.

Of related interest are private queries on remote public data, or Private Information Retrieval (PIR). This can be achieved efficiently and with perfect security [8] or with computational security [7] when two or more non-colluding servers are used. A computationally secure solution for only a single server is proposed by [14], though it is heavy in both communication and computation.

In [17], Song et al. the question for efficient *keyword* searches was raised. In that paper they propose a scheme that separately encrypts every word of a document independently. This approach has a number of disadvantages. First, it is incompatible with existing file encryption methods. Second, it cannot deal with compressed or binary data. Finally, as the authors themselves acknowledge, their scheme is not secure against statistical analysis across encrypted data. It also lacks a theoretically sound proof.

Goh [10] introduced the formal IND-CKA (Indistinguishability against chosen keyword attacks) and IND2-CKA adversary models. He gives a new approach based on Bloom filters, which hides the amount of keywords used. Chang and Mitzenmacher [6] introduce a simulation-based security definition that is intended to be stronger than IND2-CKA

The first result for an asymmetric setting (multi-user) is *Public-key Encryption with keyword Search* (PEKS) based on identity-based encryption [4]. It uses a adversary model similar to Goh's, but require the use of computationally intensive pairings. This work was extended by [2] to use multiple keywords and to remove the need for secure channels. They also raised the issue of so-called *adaptive* adversaries, where storage occurs after search queries, without giving a solution. Abdalla et. al. [1] perform a more formal analysis of the relation between anonymous IBE and PEKS and discuss the consistency of such schemes.

Curtmola et. al. [9] use a tree-based approach of searchable encryption that takes care of adaptive adversaries. Their scheme is efficient, applicable in both symmetric and asymmetric settings. To prove the scheme secure against a stronger security definition *Adaptive indistinguishability security for SSE*. Unfortunately this tree-based approach also makes updating the index very expensive, making it only suitable for one-time construction of the database.

Of independent interest is work by Bellare et al. [3] and some related papers, which uses deterministic symmetric encryption to achieve a very efficient scheme, with a very weak security model. Finally, [5] give a symmetric scheme using Bloom filters and PIR, that provably leaks no information. However, huge communication and computational costs, makes it only of theoretical interest.

In this paper we propose a novel scheme which supports keyword-based searchable encryption. In contrast with the Curtmola et al. scheme, our scheme enables the client to update the storage efficiently each time required. We give a security proof for the case that the database is built once and searches are made afterwards. While the scheme also covers the case of adaptive updating of the database, this gives rise to very complicated security models. There are a lot of

attacks (mostly statistical) that are following not so much from our scheme, but from the scenario and this also applies to other existing schemes.

## 3 Description of the Problem

Assume that a client wishes to store $n$ documents on a server where each document $D_i = (M_i, W_i), i = 1, ..., n$ is a tuple consisting of a data item $M_i$ and an associated metadata item $W_i$. The metadata item $W_i = \{w_1, w_2...\}$ is actually a set of keywords appended to $M_i$. The objectives of the client for searchable encrypted storage on the server are as follows:

1. The documents are stored on the server in such a way that the confidentiality of the data items $(M_i), i = 1, .., n$ and the associated metadata items $(W_i), i = 1, .., n$ is preserved.
2. The client queries for a keyword $w$ in order to retrieve all data items $M_i$ where $w \in W_i$ in a secure and efficient way. Here, the security means that the server learns no information about the content of the metadata items when a search is performed except the metadata items retrieved with the query.

According to the client objectives, conventional searchable symmetric encryption schemes for each document $D = (M, W)$ proceed in four phases:

`Keygen`$(s)$**:** Given a security parameter $s$, output a private key $K = (k_M, k_W)$ where $k_M \in \{0,1\}^s$ and $k_W \in \{0,1\}^s$.

`Document-Storage`$(D)$**:** Given the private key $K$, the document $D = (M, W)$ is transformed to a suitable format for storage on the server using the following sub-algorithms:

- `Data-Storage`$(M, k_M)$: Given as input the data item $M$ and the private key $k_M$, output encrypted data $\mathcal{E}_{k_M}(M)$.
- `Metadata-Storage`$(W, k_W)$: Given as input the set of associated keywords $W$ and the private key $k_W$, transform $W$ to a searchable representation $S_W$.

`Trapdoor`$(w, k_W)$**:** Given a keyword $w$ and the private key $k_W$, output a trapdoor $T_w$.

`Search`$(T_w, S_W)$**:** Given the searchable representation $S_W$ and the trapdoor $T_w$, output 1 if $w \in W$.

Having described the construction of conventional searchable encryption schemes, it is evident that the `search` algorithm requires $O(n)$ time, where $n$ is the total number of the documents stored on the database. The reason is that, given a trapdoor $T_w$, the server has to invoke the `search` function for all the searchable representations stored on the server to check the output of the `search` function.

Although the SWP scheme [17] informally, and the SSE scheme [9] formally addresses the problem by transforming each unique keyword to a searchable representation rather than each metadata item, updating the database is totally

inefficient in these schemes since the client needs to alter the whole database for each update. In the rest of the paper we address this problem by proposing an approach which provides functionalities for an efficient search, while efficiently updating the database is still possible for the client.

## 4 Background and Definitions

*Notation* Throughout the paper we use the following notation. We use $x \leftarrow_R S$ to denote that $x$ is uniformly drawn from the set $S$. For a randomized algorithm $\mathcal{A}$, we use $x \leftarrow \mathcal{A}(.)$ to denote the random variable $x$ representing the output of the algorithm. We use $\|$ to denote string concatenation.

*Pseudo-random function.* A pseudo-random function $f(.)$ which is by definition computationally indistinguishable from a truly random function, transforms each element $x$ of the set $\mathcal{X}$ to an output $y \in \mathcal{Y}$ with a secret key $k_f \in \mathcal{K}$ such that the output is not predictable. We say that a pseudo-random function $f : \mathcal{X} \times \mathcal{K} \longrightarrow \mathcal{Y}$, is a $(t, q, \varepsilon_f)$ secure pseudo-random function if for every oracle algorithm $A$ making at most $q$ oracle queries and with running time at most $t$:

$$|Pr[A^{f(.,k_f)} = 1 | k_f \leftarrow \mathcal{K}] - Pr[A^g = 1 | g \leftarrow \{F : \mathcal{X} \rightarrow \mathcal{Y}\}]| < \varepsilon_f$$

*Pseudo-random generator.* A pseudo-random generator $G(.)$ outputs strings that are computationally indistinguishable from random strings. A pseudo-random generator $G : \mathcal{X} \rightarrow \mathcal{Y}$ is $(t, \varepsilon_G)$ secure if for every algorithm $A$ with running time at most $t$:

$$|Pr[A(G(x)) = 1 | x \leftarrow \mathcal{X}] - Pr[A(y) = 0 | y \leftarrow \mathcal{Y}]| < \varepsilon_G$$

*Pseudo random permutation, (block cipher).* We say that $\mathcal{E} : \mathcal{X} \times \mathcal{K} \rightarrow \mathcal{X}$ is a pseudo-random permutation if every oracle algorithm $A$ making at most $q$ queries and win running time at most $t$ has advantage:

$$|Pr[A^{\mathcal{E}_{k_\mathcal{E}}, \mathcal{E}^{-1}_{k_\mathcal{E}}} = 1] - Pr[A^{\pi, \pi^{-1}} = 1]| < \varepsilon_\mathcal{E}$$

where $\pi$ is a random permutation selected uniformly from the set of all bijections on $\mathcal{X}$, and where the probabilities are taken over the choice of $\mathcal{K}$ and $\pi$.

### 4.1 Security Definitions

In this paper we use the security definitions introduced in [9]. Security for searchable encryption is intuitively characterized as the requirement that no information beyond the outcome of a search is leaked. However, aside from [11] and the theoretical result of [5], there are no practical schemes that satisfy this characterization; all current practical schemes leak the user's *search pattern* in addition. We take leakage of the access pattern into account by following the simulation-based security definition from [9]. For this definition we need three auxiliary

notions: the *history*, which defines the user's input to the scheme; the server's *view*, or everything he sees during the protocols; and the *trace*, which defines the information we allow to leak.

Note that this definition from [9] only considers adaptive search queries, but not adaptive storage or update queries. This means that it only covers the scenario where an initial database is made and searched, but not the scenario with adaptive additions and updates. This would really complicate the security-model, mostly because the choice that the server has to directly return matching documents. From this fact the server learns a lot of information, which would have to be included in the security model.

An interaction between the client and the server will be determined by a document collection and a set of words that the client wishes to search for (and that we wish to hide from the adversary); an instantiation of such an interaction is called a *history*.

**Definition 1 (History).** *A* history $h_q$*,is an interaction between a client and a server over* $q$ *queries, consisting of a collection of documents* $\mathcal{D}$ *and the keywords* $w_i$ *used for* $q$ *consecutive search queries. The* partial history $h_q^t$ *of a given history* $h_q = \big(\mathcal{D}, w_1, \ldots, w_q\big)$*, is the sequence* $h_q^t = \big(\mathcal{D}, w_1, \ldots, w_t\big)$*, where* $t \leq q$*.*

The server's view consists of all the information it can gather during a protocol run. This includes the encrypted documents, their associated document identifiers (ID), the set of searchable representations $S$ which are stored on the server, and all the trapdoors $T_{w_i}$ used for the search queries.

**Definition 2 (View).** *Let* $\mathcal{D}$ *be a collection of* $n$ *documents and let* $h_q = \big(\mathcal{D}, w_1, \ldots, w_q\big)$ *be a history over* $q$ *queries. An adversaries* view *of* $h_q$ *under secret key* $k$ *is defined as*

$$V_k(h_q) = \big(ID(M_1), \ldots, ID(M_n), \mathcal{E}_{k_M}(M_1), \ldots, \mathcal{E}_{k_M}(M_n), S, T_{w_1}, \ldots, T_{w_q}\big).$$

*The* partial view $V_k^t(h_q)$ *of a history* $h_q$ *under secret key* $k$ *is the sequence*

$$V_k^t(h_q) = \big(ID(M_1), \ldots, ID(M_n), \mathcal{E}_{k_M}(M_1), \ldots, \mathcal{E}_{k_M}(M_n), S, T_{w_1}, \ldots, T_{w_t}\big).$$

The trace can be considered as all the information that the server is allowed to learn, i.e. information that we allow to leak. This information includes the IDs and length of the encrypted documents (The number of keyword appear in each metadata item), the documents were returned on each search query and the user's search pattern. A user's search pattern $\Pi_q$ can be thought of as a symmetric binary matrix where $(\Pi_q)_{i,j} = 1$ iff. $w_i = w_j$. Additionally, we include $|\mathcal{W}_\mathcal{D}|$, the total amount of keywords used in all documents together. See Section 5.5 on how to hide the amount of keywords.

**Definition 3 (Trace).** *Let* $\mathcal{D}$ *be a collection of* $n$ *documents and let* $h_q = \big(\mathcal{D}, w_1, \ldots, w_q\big)$ *be a history over* $q$ *queries. The* trace *of* $h_q$ *is the sequence*

$$\mathrm{Tr}(h_q) = \left(ID(M_1), \ldots, ID(M_n), |M_1|, \ldots, |M_n|, |\mathcal{W}_\mathcal{D}|, \mathcal{D}(w_1), \ldots, \mathcal{D}(w_n), \underset{q}{\Pi}\right).$$

Now we are ready for the security definition for semantic security, where we use a simulation-based approach, like [9,13]. In this definition we assume that the client initially stores an amount of documents and afterwards does an arbitrary amount of search queries. Intuitively, it says that given all the information the server is allowed to learn (Trace), he learns nothing from the information he receives (View) about the user's input (History) that he could not have generated on his own. Note that this security definition does not take updates into account.

**Definition 4 (Adaptive Semantic Security for SSE).** *A SSE scheme is* adaptively semantically secure *if for all $q \in \mathbb{N}$ and for all (non-uniform) probabilistic polynomial-time adversaries $\mathcal{A}$, then there exists a (non-uniform) probabilistic polynomial-time algorithm (the simulator) $\mathcal{S}$ such that for all traces $\mathrm{Tr}_q$ of length $q$, and for all polynomially sampleable distributions $\langle_q = \{h_q : \mathrm{Tr}(h_q) = \mathrm{Tr}_q\}$ (i.e. the set of histories with trace $\mathrm{Tr}_q$), all functions $f : \{0,1\}^m \to \{0,1\}^{l(m)}$ (where $m = |h_q|$ and $l(m) = poly(m)$, all $0 \leq t \leq q$ and all polynomials $p$ and sufficiently large $\kappa$:*

$$\left|\Pr\left[\mathcal{A}\big(V_k^t(h_q)\big) = f(h_q^t)\right] - \Pr\left[\mathcal{S}\big(\mathrm{Tr}(h_q^t)\big) = f(h_q^t)\right]\right| < \frac{1}{p(s)}$$

*where $h_q \xleftarrow{R} \langle_q$, $k \leftarrow keygen(s)$, and the probabilities are taken over $\langle_q$ and the internal coins of keygen, $\mathcal{A}$, $\mathcal{S}$ and the underlying Storage algorithm.*

## 5 Efficiently Searchable Encryption Schemes

Consider a set of documents $\mathcal{D} = \{D_1, ..., D_n\}$, a set of metadata items $\mathcal{W} = \{W_1, ..., W_n\}$, and a set of data items $\mathcal{M} = \{M_1, ..., M_n\}$, where each document $D_i = (M_i, W_i)$ consists of a data item $M_i$ and a metadata item (a set of associated keywords) $W_i$. Let $ID_i$ be the document identifier (ID) associated to document $D_i$ (and $W_i$ and $M_i$ as well). Let $I_w = \{ID_i | w \in W_i\}$ be a collection of IDs showing in which metadata items a keyword $w$ occurs.

In this section, we first present our approach which supports computationally efficient searching on symmetrically encrypted documents. The main idea of our approach is transforming each unique keyword $w$ to a searchable representation $S_W$, in a way that the client can keep track of the metadata items in which $w$ occurs $\{W_i | w \in W_i\}$ via a trapdoor $T_w$. Our approach also allows the client to update the searchable representation of $w$ efficiently each time needed. The construction of the searchable representation for each unique keyword $w$ is:

$$S_W = (f_{k_f}(w), m(I_w), R(w)).$$

Here $f_{k_f}$ is a pseudo-random function that identifies the searchable representation of $w$, $k_f$ is the private key $k_W$ or part of $k_W$, $m(.)$ is a masking function and $R(w)$ is a function that keeps some information for unmasking $I_w$. Each time the client wants to retrieve the encrypted data items whose metadata items contain a keyword $w$, a trapdoor $T_w = (f_{k_f}(w), R'(w))$ is sent to the server. Given the

trapdoor $T_w$, the server first searches the searchable representations for $f_{k_f}(w)$. In case $f_{k_f}(w)$ occurs, the associated masked $m(I_w)$ is unmasked using $R(w)$ and $R'(w)$. The server then sends the encrypted data items whose IDs occur in $I_w$ to the client. This idea allows faster search compared to many other searchable encryption schemes since the time taken for the search is logarithmic in the number of unique keywords stored on the server. Note that since our scheme reveals the search pattern, the document identifiers that occur in the set $I_w$ are hidden until a search occurs.

Depending on how masking is performed we present two schemes which are the two variants of our approach. The first scheme is more efficient than the second one in terms of computation for the search, but the search and the storage should be performed interactively between the client and the server. The second scheme performs searching and storing the database non-interactively.

### 5.1 Scheme 1: Interactive Search and Storage

Let $I_w$ be the set of IDs showing in which metadata items keyword $w$ occurs and $S_w$ be the searchable representation of keyword $w$, which have been already stored on the server. In this scheme $I_w$ is represented as an array of bits where each bit is zero unless its position equals to at least one of the keywords that occur in the metadata items. In this scheme the construction of the searchable representation is:

$$S_w = (f_{k_f}(w), I_w \oplus G(r), \mathcal{E}_{k_\mathcal{E}}(r))$$

Our basic scheme comprises of the following algorithms:

`Keygen`$(s)$ : Given a security parameter $s$, outputs a master key $K = (k_W, k_M)$.

`Document-Storage`$(\mathcal{D}, K)$ : Given a set of documents $\mathcal{D}$, and the private key $K$, transform $\mathcal{D}$ to an appropriate format for storage on the server using the following sub algorithms:

- `Data-Storage`$(\mathcal{M}, k_M)$: Given the data items $\mathcal{M}$ and the private key $k_M$, transform each data item $M_i \in \mathcal{M}$ to encrypted form $< \mathcal{E}_{k_M}(M_i), ID_i >_{i=1,..,n}$.
- `Metadata-Storage`$(\mathcal{W}, k_W)$: Let $k_W = (k_f, k_\mathcal{E})$, where $k_f \in \{0,1\}^s, k_\mathcal{E} \in \{0,1\}^s$. Given a set of metadata items $\mathcal{W} = \{W_1, ..., W_m\}$, and the key for each unique keyword $w \in \mathcal{W}$, this algorithm has to updates the list of the indexes of searchable representation $S_w$ (in case $S_w$ is not stored on the server, we assume that $I_w = 0$ such that the updated searchable representation $S'_w$ contain indexes of the metadata items in which $w$ occur. This algorithm is performed interactively between the client and the server.
  1. Client: Build $U_w = \{ID_i | w \in W_i, W_i \in \mathcal{W}\}$ which is the indexes of the metadata items in which $w$ occurs. Send $f_{k_f}(w)$ to the server.
  2. Server: Search the first components of the searchable representations for $f_{k_f}(w)$. Send the $\mathcal{E}_{k_\mathcal{E}}(r)$ associated with $f_{k_f}(w)$ to the client.
  3. Client: Given $\mathcal{E}_{k_\mathcal{E}}(r)$, decrypt $r = \mathcal{E}^{-1}_{k_\mathcal{E}}(\mathcal{E}_{k_\mathcal{E}}(r))$. Pick a new random value $r'$, and send the server $C = (f_{k_f}(w), U_w \oplus G(r) \oplus G(r'), \mathcal{E}_{k_\mathcal{E}}(r'))$.

4. Server: Let $C = (C_1, C_2, C_3)$. Let $S_w = (S_1, S_2, S_3)$. The updated searchable representation is $S'_w = (f_{k_f}(w), U_w \oplus I_w \oplus G(r'), \mathcal{E}_{k_\mathcal{E}}(r'))$. Here the updated list of index for $w$, $I'_w = I_w \oplus U_w$, contains the IDs showing where $w$ occurs after the storage is performed.

`Trapdoor`$(w, k_W)$**:** Given a keyword $w$ and the private key $k_W = (k_f, k_\mathcal{E})$, output a trapdoor $T_w = f_{k_f}(w)$.

`Search`$(T_w, S)$**:** This algorithm is performed interactively between the server and the client.

1. Server: Given a trapdoor $T_w$, search the first component of the searchable representations for $T_w$. Let $S_w = (S_1, S_2, S_3)$. Send $S_3$ to the client.
2. Client: Send back $\mathcal{E}^{-1}_{k_\mathcal{E}}(S_3)$ to the server, c) compute $S_2 \oplus G(\mathcal{E}^{-1}_{k_\mathcal{E}}(S_3))$ to obtain $I_w$, d) send the encrypted data items whose IDs occur in $I_w$ to the client.

### 5.2 Security for Scheme 1

**Theorem 1.** *The scheme described in Section 5.1 is secure in the sense of Adaptive Semantic Security for SSE in definition 4.*

*Proof.* Let $q \in \mathbb{N}$, and let $\mathcal{A}$ be a probabilistic polynomial-time adversary. We will show the existence of a probabilistic polynomial-time algorithm $\mathcal{S}$ (Simulator) as in definition 4. Let

$$\mathrm{Tr}_q = \left(\mathrm{id}(M_1), \ldots, \mathrm{id}(M_n), |M_1|, \ldots, |M_n|, |W_\mathcal{D}|, \mathcal{D}(w_1), \ldots, \mathcal{D}(w_q), \underset{q}{\Pi}\right) \quad (1)$$

be the trace of an execution after $q$ search queries and let $H_q$ be a history consisting of $q$ search queries such that $\mathrm{Tr}(H_q) = \mathrm{Tr}_q$. Algorithm $\mathcal{S}$ works as follows:

$\mathcal{S}$ chooses $n$ random values $R_1, \ldots, R_n$ such that $|R_i| = |M_i|$ for all $i = 1, \ldots, n$. He constructs a simulated index $\bar{S}$ by making a table consisting of entries $(A_i, B_i, C_i)$ with random $A_i$, $B_i$ and $C_i$, for $i = 1, \ldots, |W_\mathcal{D}|$. Next, $\mathcal{S}$ simulates the trapdoor for query $t$, $(1 \leq t \leq q)$ in sequence. If $(\Pi_q)_{jt} = 1$ for some $j < t$ set $T_t = T_j$. Otherwise choose a $j$ in $1 \leq j \leq |W_\mathcal{D}|$ such that for all $i$, $1 \leq i < t$, $A_j \neq T_i$ set $T_j = A_j$. $\mathcal{S}$ then constructs for all $t$ a simulated view

$$\bar{V}^t_K(h_q) = \big(\mathrm{id}(D_1), \ldots, \mathrm{id}(D_n), R_1, \ldots, R_n, \bar{S}, T_1, \ldots, T_t\big), \quad (2)$$

and eventually outputs $\mathcal{A}(\bar{V}^t_k)$.

We now claim that $\bar{V}^t_k$ is indistinguishable from $V^t_k(h_q)$ and thus that the output of $\mathcal{A}$ on $V^t_k(h_q)$ is indistinguishable from the output of $\mathcal{S}$ on input $\mathrm{Tr}(h_q)$. Therefore we first state that: the $\mathrm{id}(M_i)$ in $V^t_k(h_q)$ and $\bar{V}^t_k(h_q)$ are identical, thus indistinguishable; $\mathcal{E}_k$ is a pseudorandom permutation, thus $\mathcal{E}_k(M_i)$ and $R_i$ are distinguishable with negligible probability; $f_{k_f}$ is a pseudorandom function, thus $t_i = f_{k_f}(w_i)$ and $T_i$ are distinguishable with negligible probability. Also the relations between the elements are correct by construction.

What is left is to show that $\bar{S}$ is indistinguishable from $S$, i.e. that the tuples $(A_i, B_i, C_i)$ are indistinguishable from tuples $(f_{k_f}(w_i), I_{w_i} \oplus G(r_i), \mathcal{E}_{k_\mathcal{E}}(r_i))$. First note again that $f_{k_f}(w_i)$ is indistinguishable from the random $A_i$ since $f_{k_f}$ is a pseudorandom function. Given $I_{w_i}$ and the fact that $G$ is a pseudorandom generator there exists an $s_i$ such that $I_{w_i} \oplus G(s_i) = B_i$. Given that $\mathcal{E}$ is an IND-CPA permutation $C_i$ is indistinguishable from $\mathcal{E}(s_i)$.

Since $\bar{V}_k^t$ is indistinguishable from $V_k^t(h_q)$, the output of $\mathcal{A}$ will also be indistinguishable. This completes the proof.

### 5.3 Scheme 2: Non-interactive Search and Storage

Although scheme 1 is efficient in terms of computation for searching searchable representations, there are two disadvantages with the scheme: i) the `Metadata-Storage` and the `search` algorithms are performed interactively with the server, ii) the `Metadata-Storage` algorithm requires a large bandwidth since the size of $U_w$ should be equal to the size of $I_w$ (which is large for the large scale databases).

Here we present scheme 2 which efficiently addresses the disadvantages described above with the cost of more computation for the search. The key idea to perform the storage and the search non-interactively is to deploy a hash chain. A hash chain of length $N$ ,

$$H^N(a) = \underbrace{H(H(...H(a)...))}_{N}$$

is constructed by applying repeatedly a hash function $H(.)$ to an initial seed value $a$ [15]. Since only the client knows the seed, she is able to traverse the chain forward and backward, while the server is able to traverse the chain forward only.

In contrast with scheme 1, where the IDs showing in which metadata items a keyword $w$ occurs are stored in one array of bits $I_w$, in scheme 2 the list of the IDs are stored in the masked form individually each time the `metadata-storage` algorithm is invoked. This approach diminishes the bandwidth required for storing the metadata items.

In this scheme the masking function $m(.)$ is actually the pseudorandom permutation function function $\mathcal{E}$. Let $I_i(w)$ be the set of IDs showing in which metadata item $w$ occurs, which have been added to $S_W$ in the $i$th updating $S_W$ occurs. The searchable representation of $w$ after updating $i$ times is:

$$S_{old}(w) = (f_{k_f}(w), \mathcal{E}_{k_1(w)}(I_1(w)), H'(k_1(w)), ..., \mathcal{E}_{k_i(w)}(I_i(w)), H'(k_i(w))),$$

where $k_j(w) = H^{N-ctr}(w||k)$. Here $ctr$ is a counter which is incremented each time the `Metadata-Storage` function is invoked (for any keyword). The counter $ctr$ is stored ion the client side. This construction encrypts each list $I_j(w)$ with a unique secret key $k_j(w)$ such that the server is able to compute the encryption key of the previously added lists $k_{j-1}(w), ..., k_1(w)$ given the encryption key of

the latest storage $k_j(w)$, by traversing the hash chain forward. Since the server cannot traverse the chain backward, it is not possible for the server to compute $k_{j+1}(w)$ given $k_j(w)$.

Scheme 2 comprises of the following algorithms:

`Keygen`$(s)$**:** Given a security parameter s, output a master key $K = (k_W, k_M)$.

`Document-Storage`$(\mathcal{D}, K)$ : Given a set of documents $\mathcal{D}$, and the private key $K$, transform $\mathcal{D}$ to an appropriate format for storage on the server using the following sub algorithms:

- `Data-Storage`$(\mathcal{M}, k_M)$: Given the data items $\mathcal{M}$ and the private key $k_M$, transform each data item $M_i \in \mathcal{M}$ to encrypted form $< E_{k_M}(M_i), ID_i >_{i=1,..,n}$.
- `Metadata-Storage`$(\mathcal{W}, k_W, ctr)$: Let $k_W = (k_f, k)$, where $k_f, k \in \{0,1\}^s$. Let $N$ be the length of the hash-chain. Assume that $S_{old}(w)$ has been already $i$ times updated. For each unique keyword $w \in \mathcal{W}$ construct $I_{i+1}(w) = \{ID_j | w \in W_j, W_j \in \mathcal{W}\}$ and then perform the following; i) increment the counter $ctr = ctr+1$, ii) compute the secret key $k_{i+1}(k) = H^{N-ctr}(w||k)$, iii) encrypt $\mathcal{E}_{k_{i+1}(w)}(I_{i+1}(w))$, iv) sends the tuple

$$(f_{k_f}(w), \mathcal{E}_{k_{i+1}(w)}(I_{i+1}(w)), H'(k_{i+1}(w)))$$

  to the server, who adds the received tuple to $S_{old}(w)$. Figure illustrates the `metadata-storage` algorithm in scheme 2.

`Trapdoor`$(w, k_W, ctr)$**:** Given a keyword $w$ and the private key $k_W$, output a trapdoor $T_w = (f_{k_f}(w), H^{N-ctr}(w||k))$.

`Search`$(S, T_w)$**:** Let $T_w = (T_1, T_2)$. Assume that $S_W$ has been updated for $i$ times. Given the trapdoor $T_w$ and the searchable representations $S$, search $S$ for $T_1$. If $T_1$ occurs, compute $H'(T_2)$, if $H'(T_2) = H'(k_i(w))$, decrypt $I_i$ using $T_2$, otherwise keep computing $T_2 = H(T_2)$ until $H'(T_2) = H'(k_i(w))$. After $k_i(w)$ is computed, repeats the same procedure to compute the previously added list of document identifiers. Having decrypted $I_i(w), ..., I_1(w)$ send the documents whose IDs occur in the lists.

**Optimization. 1**. Each time the server decrypts each list $I_j(w)$ after a search, the list is kept in plaintext, such that for later searches, the server has to decrypt only the list of the document identifiers that have been added to $S_W$ since the last search. This modification will decrease the computation for the Search algorithm.

**Optimization. 2**. Schemes 2 suffers from a limitation that the maximum number of times the storage can be updated is limited. The limitation comes from the finite length of the pseudo-random chain used in the scheme. In other words, after the counter *ctr* reaches the value of $N$, where $N$ is the length of the chain, the chain cannot be used. At this point the pseudo-random chain is said to be exhausted and the whole process should be repeated again with a different seed to re-initialize the chain. One way to decrease the exhaustion rate is that the counter *ctr* is only incremented in case a search has occurred since the latest update. The reason is that without performing the search, the server does not

know anything about the key $k_i(w)$ used in the last update. Hence, the exact $k_i(w)$ used for the last time that update occurred can be used for the current update.

### 5.4 Security for Scheme 1

**Theorem 2.** *The scheme described in Section 5.3 is one way secure for SSE in definition 4.*

In scheme 2, the trace and the simulation view will be the same as Eq.1 and Eq.2 respectively. Without loss of generality let $S_W = (f_{k_f}(w), \mathcal{E}_{k_1(w)}(I_1(w)), H'(k_1(w)))$. Then, $f_{k_f}(w)$ is indistinguishable from random. However, since the key $k_1$ is constructed by the one way hash function $H(.)$, the key is one way secure. Therefore $\mathcal{E}_{k_1(w)}(I_1(w))$ will be one way secure.

### 5.5 Security of Updates

In the security proof in Section 5.4 we did not consider the security of updates. In fact there is information leakage in this case, specifically the amount of keywords in each update and information on which keyword are in common over several updates. For our extended scheme in Section 5.3 we did not discuss security at all. There the security is similar to that of 5.1, but the improvements does not make sense when updates are not considered. However, there are several tricks to minimize this information leakage:

**Batched updates.** Updating a single document reveals the amount of keywords used for that document. However, our scheme allows us to update many documents at once. In that case the update only reveals information about the aggregated keywords over all updated documents. In this way the information leakage goes asymptotically towards zero bits if the amount of simultaneously updated documents increases.

**Fake updates.** The `Metadata-Storage` algorithm allows us to update the searchable representation of a keywords without actually changing the indexed documents, similar in idea to the technique in [2] to hide the amount of keywords. This allows the client to always do an update with an identical amount of keywords, or even to update all keywords at once.

## 6 Application

Having described the schemes we proposed, we revisit the two scenarios from the introduction to show how each exploits the advantages of the schemes. The first scheme is appropriate for the traveller who uses PHR$^+$ to store his medical record such that the record can be retrieved selectively anywhere. As an example, a journalist using PHR$^+$ to check the validation of a vaccination. In this case, since the client (journalist) uses a broadband internet connection, the time delay due to the second round of communication for the search is not a problem. The

second scheme is appropriate for instance for a GP who uses $PHR^+$ to store the record of a patient, and who retrieves the record of each patient before or during a visit. The GP also updates the record of the patient afterwards.

## 7 Conclusion

We propose a novel searchable symmetric encryption scheme which supports the searching time logarithmic in the number of the unique keywords stored on the server while it is updatable. We propose two variants of the approach which differ in the efficiency of the search and the update. We now present a general assessment of the two schemes proposed. The first scheme is after each update more efficient in terms of computation, but requires two rounds of communication between the server and the client for each search. The second scheme enables the client to update the stored documents with a minimum bandwidth and high efficiency. This scheme also makes possible to undo update for the client. However, the update and the search should be interleaved and the maximum number of times the documents stored can be updated is limited. Table 1 summarizes the features of the schemes proposed.

**Table 1.** Summary of the features of the schemes proposed. In this table $u$ is the number of unique keywords, $N$ is the length of the hash chain, and $l$ is the average number of encrypted document identifiers added to $S_W$ since the last search.

| | **Variants of the Basic Scheme** | |
|---|---|---|
| **Features** | Scheme 1 | Scheme 2 |
| Communication overhead | Two rounds | One round |
| Searching Computation | $log(u)$ | $log(u) + \frac{N}{2}l$ |
| Condition on Update | Occurs rarely | Interleaved with search |

## Acknowledgment

This research is supported by the SEDAN project, funded by the Sentinels program of the Technology Foundation STW under project number EIT.7630.

## References

1. Abdalla, M., Bellare, M., Catalano, D., Kiltz, E., Kohno, T., Lange, T., Malone-Lee, J., Neven, G., Paillier, P., Shi, H.: Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. J. Cryptology 21(3), 350–391 (2008)

2. Baek, J., Safavi-Naini, R., Susilo, W.: Public key encryption with keyword search revisited. Cryptology ePrint Archive, Report 2005/191 (2005), http://eprint.iacr.org/
3. Bellare, M., Boldyreva, A., O'Neill, A.: Deterministic and efficiently searchable encryption. In: Menezes [16], pp. 535–552
4. 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)
5. Boneh, D., Kushilevitz, E., Ostrovsky, R., Skeith III, W.E.: Public key encryption that allows PIR queries. In: Menezes [16], pp. 50–67
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. Chor, B., Gilboa, N.: Computationally private information retrieval (extended abstract). In: STOC, pp. 304–313 (1997)
8. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. J. ACM 45(6), 965–981 (1998)
9. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: Juels, A., Wright, R.N., di Vimercati, S.D.C. (eds.) ACM Conference on Computer and Communications Security, pp. 79–88. ACM, New York (2006)
10. Goh, E.-J.: Secure indexes. Cryptology ePrint Archive, Report 2003/216 (2003), http://eprint.iacr.org/
11. Goldreich, O., Goldwasser, S., Halevi, S.: Public-key cryptosystems from lattice reduction problems. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 112–131. Springer, Heidelberg (1997)
12. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious rams. J. ACM 43(3), 431–473 (1996)
13. Goldwasser, S., Micali, S.: Probabilistic encryption. J. Comput. Syst. Sci. 28(2), 270–299 (1984)
14. Kushilevitz, E., Ostrovsky, R.: Replication is not needed: Single database, computationally-private information retrieval. In: FOCS, pp. 364–373 (1997)
15. Lamport, L.: Password authentification with insecure communication. Commun. ACM 24(11), 770–772 (1981)
16. Menezes, A. (ed.): CRYPTO 2007. LNCS, vol. 4622. Springer, Heidelberg (2007)
17. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: IEEE Symposium on Security and Privacy, pp. 44–55 (2000)
18. Troncoso-Pastoriza, J.R., Katzenbeisser, S., Celik, M.U.: Privacy preserving error resilient dna searching through oblivious automata. In: Ning, P., di Vimercati, S.D.C., Syverson, P.F. (eds.) ACM Conference on Computer and Communications Security, pp. 519–528. ACM, New York (2007)