# Adaptively Secure Computationally Efficient Searchable Symmetric Encryption

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

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

**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 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 update. We show how each scheme could be used in a personal health record system.

## 1 Introduction

Searchable encryption is a technique that allows a client to outsource documents to a honest but curious server in encrypted form, such that the documents stored can be retrieved selectively while revealing as little information as possible to the server. Searchable encryption has many applications, particularly where client privacy is a main concern such as in E-mail servers [5], keeping medical information of a client [21], storing private videos and photos, and backup applications [4,20].

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 heath record (PHR) systems which are initiated and maintained by an individual, are examples of digital record systems. These systems become more and more popular, even nation-wide like the 'Elektronisch Patienten Dossier' in the Netherlands [19].

An example of a PHR system is Google Health which offers a client the ability to store her medical records on Google's servers, and allows a general practitioner (GP) to get access to the medical records of her patients. Unlike

paper-based systems, where the privacy is mainly protected by chaos (since it is almost impossible to locate an individual's 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 of PHR$^+$ are i) a GP who uses PHR$^+$ 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 his 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.

**Problem.** The 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 encryption scheme that offers efficient searching and updating 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.

The rest of the paper is organized as follows: Section 2 describes the related work in this field. In section 3 we describe the two problems with the conventional searchable encryption schemes. In section 4 we describe the background and definitions. The basic scheme and the two variants of the basic scheme are presented in section 5. In section 6 we describe an application scenario of the schemes proposes, and the conclusion is followed in section 7.

## 2 Related Work

In theory, the classical work of Goldreich and Ostrovsky [14] 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 practive as large constants are hidden in the 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 [9] or with computational security [8] when two or more noncolluding

servers are used. A computationally secure solution for only a single server is proposed by [16], though it is heavy in both communication and computation.

In [20] 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. Firstly, it is incompatible with existing file encryption methods. Secondly, 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 [12] 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 [7] 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 [5]. 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. [11] 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 [3], which uses deterministic symmetric encryption to achieve a very efficient, but not very secure scheme. Finally, [6] give a symmetric scheme using Bloom filters and PIR, that provably leaks no information . However, because of the huge communication and computational costs it is only of theoretical interest.

## 3 Description of the Problem

Assume that a client wants 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 to the maximum extend.

2. The client queries for a keyword $w$ to retrieve the data item $M_i$ in case $w \in W_i$ 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 encryption schemes for each document $D = (M, W)$ proceed in three phases:

`Keygen`$(s)$**:** Given a security parameter $s$, output a master private key $K = (k_m, k_w) \in (\{0,1\}^s, \{0,1\}^s)$.

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

- `DataStorage`$(M, k_m)$: Given the private key $k_m$, and the data item $M$, transform $M$ to an encrypted form $E_{k_m}(M)$ for storage on the server. This algorithm is invoked by the client.
- `MetadataStorage`$(W, k_w)$: Given the private key $k_w$, and the metadata item $W$, transform $W$ to a searchable representation $S(W)$ for storage on the server. This algorithm is invoked by the client.

`Trapdoor`$(w, k_w)$**:** Given a keyword $w$ and the private key $k_w$, output a trapdoor $T_w$. This algorithm is invoked by the client.

`Search`$(T_w, S)$**:** Let $S = S(W_1), ..., S(W_n)$ be the set of the searchable representation of $n$ metadata items stored on the server. Given the trapdoor $T_w$ and each searchable representation $S(W) \in S$, output 1 if $w \in W$. This algorithm is invoked by the server.

Having described the construction of the 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 `Search`$(T_w, S(W_i))$, for $i = 1, .., n$, to check if there is a match between $T_w$ and $S(W_i)$. Although the SWP scheme [20] informally, and the SSE scheme [11] formally addresses the problem by transforming each unique keyword to a searchable representation rather than each metadata item, update is totally inefficient in these schemes.

## 4 Backgrounds and Definitions

*Notation* Throughout the paper we use the following notation. The domain over which the random variable is defined is denoted by a script letter (e.g. $\mathcal{X}$). We use $x \leftarrow_R \mathcal{X}$ to denote $x$ is uniformly drawn from the set $\mathcal{X}$. For a randomized algorithm $\mathcal{A}$, we use $x \leftarrow \mathcal{A}(.)$ to denote the random variable $x$ representing the output of the algorithm.

*Pseudo-random function.* A pseudo-random function $f(.) : \mathcal{X} \times \mathcal{K} \longrightarrow \mathcal{Y}$ ,which is by definition computationally indistinguishable from a truly random function, transforms an element $x \leftarrow_R \mathcal{X}$ to an output $y \leftarrow_R \mathcal{Y}$ with a secret key $k \leftarrow_R \mathcal{K}$

such that the output is not predictable. We say that a pseudo-random function $f(.,k)$ is $(t,q,\varepsilon_f)$ secure if for every oracle algorithm $A$ making at most $q$ oracle queries and with running time at most $t$:

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

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

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

*Pseudo random permutation, (i.e. a 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:

$$|Pr[A^{\mathcal{E}_K,\mathcal{E}_K^{-1}}=1] - Pr[A^{\pi,\pi^{-1}}=1]| < \varepsilon_E$$

where $\pi$ represents 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

Security for searchable encryption is intuitively characterized as the requirement that no information beyond the outcome of a search is leaked. However, aside from [13] and the theoretical result of [6], 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 [10]. 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 the definition from [10] only considers adaptive search queries, but not adaptive storage or update queries.

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).** *Let $\mathcal{W}$ be a dictionary consisting of all possible keywords. 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 $t \leq q$.*

Intuitively, the server's view consists of all the information it can gather during a protocol run. This includes the encrypted documents and their identifier, the set of searchable representations $S$ 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 $H_q = (\mathcal{D}, w_1, \ldots, w_q)$ be a history over $q$ queries. An adversary's* view *of $H_q$ under secret key $K$ is defined as*

$$V_K(H_q) = \big(\mathrm{id}(M_1), \ldots, \mathrm{id}(M_n), \mathrm{E}_{k_m}(M_1), \ldots, \mathrm{E}_{k_M}(M_n), S, T_{w_1}, \ldots, T_{w_q}\big).$$

*The* partial view $V_K^t(H_q)$ *of a history $H_q$ under secret key $K$ is the sequence*

$$V_K^t(H_q) = \big(\mathrm{id}(M_1), \ldots, \mathrm{id}(M_n), \mathrm{E}_{k_m}(M_1), \ldots, \mathrm{E}_{k_m}(M_n), S, T_{w_1}, \ldots, T_{w_t}\big).$$

Finally, the trace can be considered as all the information that the server is allowed to learn, i.e. information that we allow to leak. In this information we include the indexes and length of the encrypted documents, which documents indexes 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 5.7 on how to hide the amount of keywords.

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

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

Now we are ready for the security definition for semantic security, where we use a simulation-based approach, like [11,15]. In this definition we assume 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*

$$\mathcal{H}_q = \{H_q \ : \ \mathrm{Tr}(H_q) = \mathrm{Tr}_q\}$$

*(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 $0 \le t \le q$ and all polynomials $p$ and sufficiently large $\kappa$:*

$$\left|\Pr\Big[\mathcal{A}\big(V_K^t(H_q)\big) = f(H_q^t)\Big] - \Pr\Big[\mathcal{S}\big(\mathrm{Tr}(H_q^t)\big) = f(H_q^t)\Big]\right| < \frac{1}{p(k)}$$

*where* $H_q \xleftarrow{R} \mathcal{H}_q$, $K \leftarrow Keygen(s)$, *and the probabilities are taken over* $\mathcal{H}_q$ *and the internal coins of Keygen,* $\mathcal{A}$, $\mathcal{S}$ *and the underlying* `Storage` *algorithm.*

## 5 Efficiently Searchable encryption Schemes

In this section we propose our basic scheme followed by the two variants of the basic scheme. In the rest of the paper we assume that each document $D_i$ is associated with a unique document identifier $i$, which is generated by the client.

### 5.1 Basic Scheme

In this section, we present the basic scheme which supports efficiently updateable searchable encrypted documents. The main idea of the basic scheme 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 this keyword occurs $\{W_i | w \in W_i\}$ by a trapdoor $T_w$. This idea allows faster search compared to conventional searchable encryption schemes since the time taken for the search is logarithmic in the number of unique keywords stored on the server (assuming a tree structure for the searchable representations).

Our basic scheme comprises of the following algorithms:

`Keygen`$(s)$ Given a security parameter $s$, output a master key $K = (k_m, k_w) \in (\{0,1\}^s \times \{0,1\}^s)$.

`Storage`$(D_1, ..., D_n)$, $K$ For the client to store a collection of documents on the server, first an exclusive document identifier $i$ is associated with each document $D_i = (M_i, W_i)$. Then, given the master key $K = (k_m, k_w)$, the set of documents are transformed to a suitable format for storage using the following sub-algorithms:

- `DataStorage`$((M_1, ..., M_n), k_m)$: Given the data items $(M_1, ..., M_n)$ and the secret key $k_m$, transform each data item $M_i, i = 1, ..., n$ to an encrypted form $E_{k_m}(M_i)$ and store the tuple $(E_{k_m}(M_i), i)$ on the server, where $i$ is the document identifier of $D_i$.
- `MetadataStorage`$((W_1, ..., W_n), k_w)$: This algorithm consists of the following steps:
  1. Gather all the unique keywords that occur in the metadata items $(W_1, ..., W_n)$.
  2. For each unique keyword $w$, build a set $I(w) = \{i | w \in W_i\}$ consisting of the identifier of the documents in which $w$ occurs.
  3. The keyword $w$ is transformed to a searchable representation $S(w) = (f_{k_w}(w), m(I(w)))$, where $f_{k_w}(w)$ identifies the searchable representation of $w$, and $m(.)$ is a masking function.

`Trapdoor`$(w)$: Each time the client wants to retrieve the set of encrypted data items $\{E_{k_m}(M_i) | w \in W_i\}$ from the server, a trapdoor $T_w = (f_{k_w}(w), t_w)$ is computed and is sent to the server, where $t_w$ is some information that helps the server to unmask $I(w)$.

`Search`$(S, T_w)$: Let $S = \{S(w_1), ..., S(w_u)\}$ be the set of searchable representations of all the $u$ unique keywords. Given the trapdoor $T_w$, the server searches $S$ for $f_{k_w}(w)$. If $f_{k_w}(w)$ occurs, the server unmasks the associated set $I(w)$, using the trapdoor $T_w$. The server then reads the document identifiers that occur in $I(w)$ to send back the client the set $\{E_{k_m}(M_i)|i \in I(w)\}$.

Having clarified our approach for an efficient `Search` algorithm in terms of computation, we present two variants of the basic scheme where the difference between the schemes comes from how the masking and unmasking functionalities are performed.

### 5.2 Scheme 1: A computationally efficient scheme

Here, the set $I(w)$ is represented as an array of bits where each bit is 0 unless the position of this bit is equal to one of the document identifiers which occur in $I(w)$. In this scheme the searchable representation $S(w)$ of each unique keyword $w$ stored on the server is a triple:

$$S(w) = (f_{k_w}(w), I(w) \oplus G(r), F(r)).$$

The components of the searchable representation $S(w)$ are:

- The pseudo-random value $f_{k_w}(w)$ identifies the searchable representation of $w$.
- The masking function $m(I(w)) = I(w) \oplus G(r), F(r)$ is the bitwise XOR of $I(w)$ with a random array of bits $G(r)$ generated from a nonce $r$. The nonce $r$ is used exclusively for $w$.
- The function $F(.)$ is an IND-CPA trapdoor permutation (e.g. an ElGamal encryption) with the inverse $F^{-1}(.)$, which allows the client to recover the nonce $r = F^{-1}(F(r))$ when needed.

Each time the client wants to add a new document to the database, the identifier of the new document should be added to $I(w)$, in such a way that minimum information is leaked to the server. Let $U(w)$ denote the list of the document identifiers to be added to the database. In this scheme $U(w)$ is represented by an array of bits, where each bit is zero unless the position of the bit is equal to one of the elements of $U(w)$. Observe that $I'(w) = I(w) \oplus U(w)$ is the updated list of the document identifiers stored on the server.

We now describe the algorithm of scheme 1:

`Keygen`$(s)$ Given a security parameter $s$, output a master key $K = (k_m, k_w) \in (\{0,1\}^s, \{0,1\}^s)$.

`Storage`$(D_1, ..., D_n)$ For the client to store a collection of documents on the server, an exclusive document identifier $i$ is associated with each document $D_i = (M_i, W_i)$. Then, given the master key $K = (k_m, k_w)$, the set of documents are transformed to a suitable format for storage using the following sub-algorithms:

– `DataStorage`$((M_1, ..., M_n), k_m)$: Given the data items $(M_1, ..., M_n)$ and the secret key $k_m$, transform each data item $M_i, i = 1, ..., n$ to an encrypted form $E_{k_m}(M_i)$ and store the tuple $(E_{k_m}(M_i), i)$ on the server.
– `MetadataStorage`$((W_1, ..., W_n), k_w)$: Given the metadata items $(W_1, ..., W_n)$, all the unique keywords are gathered to build a set $U(w) = \{i | w \in W_i\}$ for each unique keyword $w$. Let the searchable representation $S(w)$ stored on the server be $(f_{k_w}(w), I(w) \oplus G(r), F(r))$. For the client to receive the nonce $r$ from the server, the pseudorandom value $f_{k_w}(w)$ is sent to the server who responds to the client by sending back $F(r)$. Given $F(r)$, the client recovers $r = F{-}1(F(r))$, and generates a new nonce $r'$ to compute $U(w) \oplus G(r) \oplus G(r')$. The client eventually sends $(U(w) \oplus G(r) \oplus G(r'), F(r')$ to the server who computes $(I(w) \oplus G(r)) \oplus (U(w) \oplus G(r) \oplus G(r'))$ to obtain $I'(w) \oplus G(r')$ and to replace $I(w) \oplus G(r), F(r)$ by $I'(w) \oplus G(r'), F(r')$.
Figure 1 illustrates the message exchange of the `MetadataStorage` algorithm. In this figure, $p$ is a large prime number.

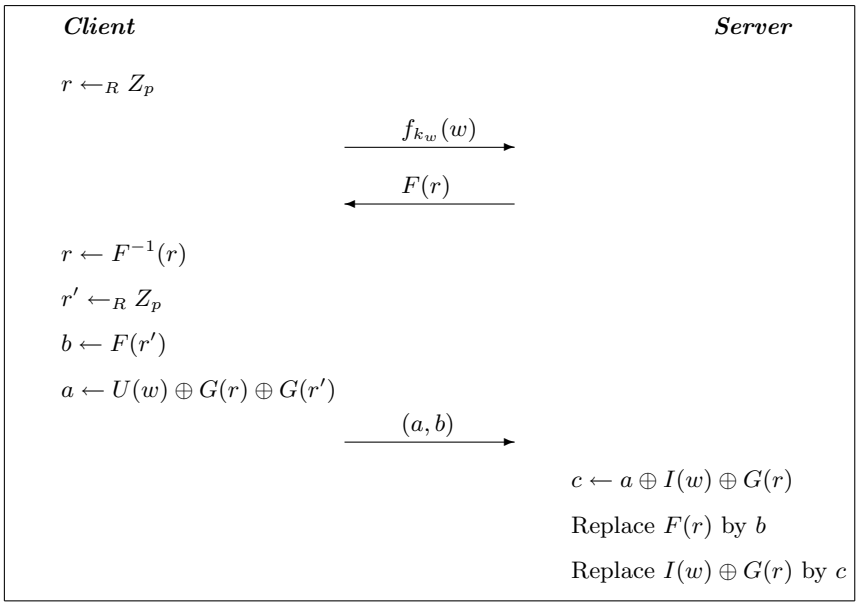


**Fig. 1.** `MetadataStorage` algorithm in scheme 1

`Trapdoor`$(w)$ Given a keyword $w$, output a trapdoor $T_w = f_{k_w}(w)$.
`Search`$(T_w, S)$ Let $S = \{S(w_1), ..., S(w_u)\}$ be the set of searchable representation of all the $u$ unique keywords stored on the server. Given the trapdoor $T_w$, the server searches $S$ for $T_w$. If $T_w$ occurs, the server sends back the associated $F(r)$ with $T_w$ to the client who responds by computing $r = F^{-1}(F(r))$

and sending the nonce $r$ to the server. Given the nonce $r$, the server computes $(I(w) \oplus G(r)) \oplus G(r)$ to obtain $I(w)$. The server then reads $I(w)$ to send the set $\{E_{k_m}(M_i) | i \in I(w)\}$ to the client. The message exchange of the `Search` algorithm is illustrated in Fig. 2

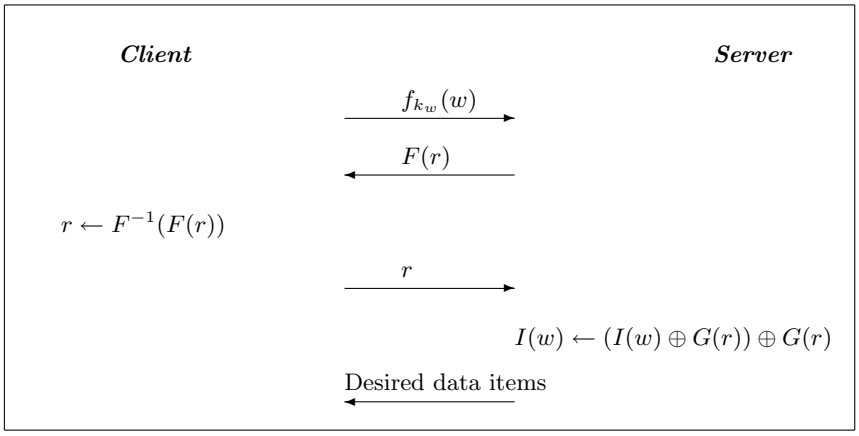


**Fig. 2.** The message exchange of the `Search` algorithm in scheme 1

### 5.3 Adaptive Semantic Security for SSE

**Theorem 1.** *Scheme 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

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

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:

Algorithm $\mathcal{S}$ 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, algorithm $\mathcal{S}$ 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

$$\bar{V}_K^t(H_q) = \big(\mathrm{id}(D_1), \ldots, \mathrm{id}(D_n), R_1, \ldots, R_n, \bar{S}, T_1, \ldots, T_t\big),$$

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

We now claim that $\bar{V}_K^t$ is indistinguishable from $V_K^t(H_q)$ and thus that the output of $\mathcal{A}$ on $V_K^t(H_q)$ 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 $V_K^t(H_q)$ and $\bar{V}_K^t(H_q)$ are identical, thus indistinguishable; $\mathrm{E}_{k_m}$ is a pseudorandom permutation, thus $\mathrm{E}_{k_m}(M_i)$ and $R_i$ are distinguishable with negligible probability; $f_{k_w}$ is a pseudorandom function, thus $t_i = f_{k_w}(w_i)$ 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 $(f_K(w_i), I(w_i) \oplus G(r_i), F_K(r_i))$. First note again that $f_K(w_i)$ is indistinguishable from the random $A_i$ since $f_K$ 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 $F$ is an IND-CPA trapdoor permutation $C_i$ is indistinguishable from $F(s_i)$.

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.

### 5.4 Scheme 2: Diminishing the communication cost

Although scheme 1 offers an efficient in terms of computation for the `Search` algorithm, there are two disadvantages: i) the `Search` algorithm requires two rounds of communication ii) the `MetadataStorage` algorithm requires a large bandwidth, which comes from the fact that the size of the sent $U(w)$ should be equal to the size of $I(w)$ (which could be large for large databases).

Here we present scheme 2, which addresses the shortcomings of scheme 1 to reduce the cost of communication. The key idea to remove the second round of communication is to use a pseudo-random chain. A pseudo-random chain of length $l$, which is denoted by $f^l(a) = \underbrace{f(f(...f(a)...))}_{l}$ is constructed by applying repeatedly a pseudo-random function $f(.)$ to an initial seed value $a$ [17]. Only the party who knows the seed value is able to traverse the chain forward and backward, while the other parties are able to traverse the chain forward only. The key idea also to diminish the bandwidth required for the `MetadataStorage` algorithm is to store the list of the document identifiers individually in masked form with a unique making key, each time an update of the database occurs.

### 5.5 Construction

In this scheme the set $I(w) = \{i | w \in W_i\}$ is represented by a list of document identifiers, and the masking function $m(.)$ is a secure permutation function $\mathcal{E}_k(.)$ with a masking key $k$. Let $S(w) = (f_{k_w}(w), \mathcal{E}_k(I(w)))$ be the searchable representation of the keyword $w$ stored on the database. Let $D_j = (M_j, W_j)$, where $w \in W_j$, be a new document which is about to store on the server. To update the searchable representation $S(w)$, the set of the new document identifiers is constructed, say $I'(w) = \{j\}$. Then a new key $k'$ is generated to

mask the list $\mathcal{E}_{k'}I'(w)$. Given $\mathcal{E}_{k'}I'(w)$, the updated searchable representation is $S(w) = (f_{k_w}(w), E_k(I(w)), E_{k'}(I'(w)))$.

Taking into account the example presented above, the keys $k, k'$ used to mask $I(w), I'(w)$ respectively, should satisfy two requirements: firstly, the latest key $k'$ cannot be computed when the older key $k$ is known, and secondly, the older key $k$ can be computed when the latest key $k'$ is known. To fulfill these requirements we use a pseudorandom chain to construct the masking key. Let $j-1$ be the total number of times that a searchable representation $S(w)$ has been updated. Then, to update $S(w)$ for the $j$th time, the secret key used for the permutation function is $k_j(w) = h^{l-ctr}(w||k_w)$. Here, $ctr$ is a global counter that is incremented each time the database is updated, and $l$ is the length of the chain. In other words, the elements of the pseudo-random chain are used as a key to encrypt the nonce one by one, each time the searchable representation is updated.

Let $I_i(w)$ be the list of the document identifiers added to the searchable representation $S(w)$ after the $i$th time an update has occurred, and $k_i(w)$ be the secret key used to mask $I_i(w)$. Then, the searchable representation $S(w)$ after $j$ times update is:

$$S(w) = (f_{k_w}(w), \mathcal{E}_{k_1(w)}(I_1(w)), f'(k_1(w)), ..., \mathcal{E}_{k_j(w)}(I_j(w)), f'(k_j(w))).$$

where $f'(.)$ is a pseudo-random function.

### 5.6 Details

Scheme 2 comprises of the following algorithms:

`Keygen`$(s)$ Given a security parameter $s$, outputs a master key $K = (k_m, k_w) \in (\{0,1\}^s, \{0,1\}^s)$.

`Storage`$(D_1, ..., D_n)$ for the client to store a collection of documents on the server, an exclusive document identifier $i$ is associated with each document $D_i = (M_i, W_i)$.

- `DataStorage`$((M_1, ..., M_n), k_m)$: Given the data items $(M_1, ..., M_n)$ and the secret key $k_m$, transforms each data item $M_i, i = 1, ..., n$ to an encrypted form $E_{k_m}(M_i)$ to store the tuple $(E_{k_m}(M_i), i)$ on the server.
- `MetadataStorage`$(((W_1, ..., W_n), k_w))$: Let
  $S(w) = (f_{k_w}(w), \mathcal{E}_{k_1(w)}(I_1(w)), f'(k_1(w)), ..., \mathcal{E}_{k_j(w)}(I_j(w)), f'(k_j(w)))$
  be the searchable representation of $w$ stored on the server, where $j$ is the total number of times that $S(w)$ has been updated. Given the metadata items $(W_1, ..., W_n)$, the unique keywords are gathered to build a set $I_{j+1}(w) = \{i | w \in W_i\}$. For each unique keyword $w$, the client first increments the counter stored $ctr' = ctr + 1$ and then computes a new encryption key $k_{j+1} = f^{l-ctr'}(w||k_w)$ by traversing the pseudo-random chain one step backward. The client then sends the triple

  $$(f_{k_w}(w), \mathcal{E}_{k_{j+1}(w)}(I_{j+1}(w)), f'(k_{j+1}(w)))$$

  to the server who adds the received triple to $S(w)$. The `MetadataStorage` algorithm is illustrated in Fig. 3.

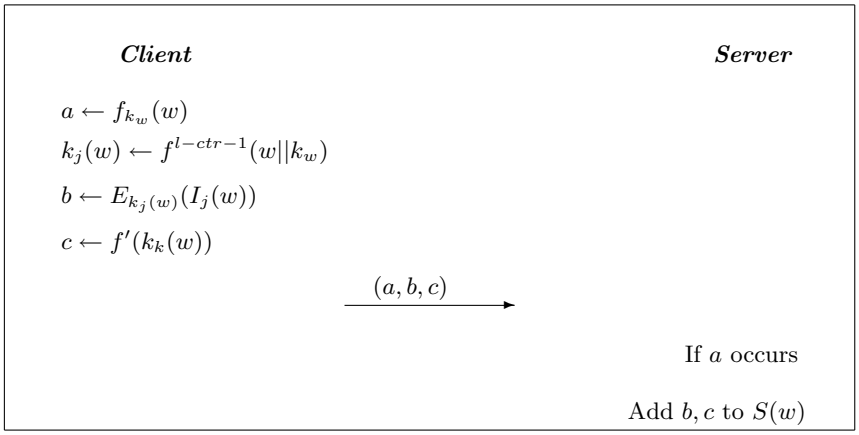


**Fig. 3.** The message exchange of the `MetadataStorage` algorithm in scheme 2

`Trapdoor`$(w)$ : Given a keyword $w$ output a trapdoor $T_w = (t_w, t'_w)$ where, $t_w = f_{k_w}(w)$ and $t'_w = f^{l-ctr}(w||k_w)$.

`Search`$(w)$ : Given the trapdoor $T_w = (t_w, t'_w)$, the server searches the searchable representations for $t_w$. If $t_w$ occurs, the server computes the masking key of the latest update $k_j(w)$ by traversing the chain forward as follows: check if $f'(t'_w) = f'(k_j(w))$ then $k_j(w) = t'_w$ otherwise $t_w = f(t'_w)$ and perform the checking again. The `Search` algorithm is illustrated in Fig. 4.

**Optimization. 1.** Each time the server decrypts the list of the document identifiers 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 $l$, where $l$ 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 $k(w)$ used in the last update. Hence, the exact $k(w)$ used for the last time that update occurred can be used for the current update.

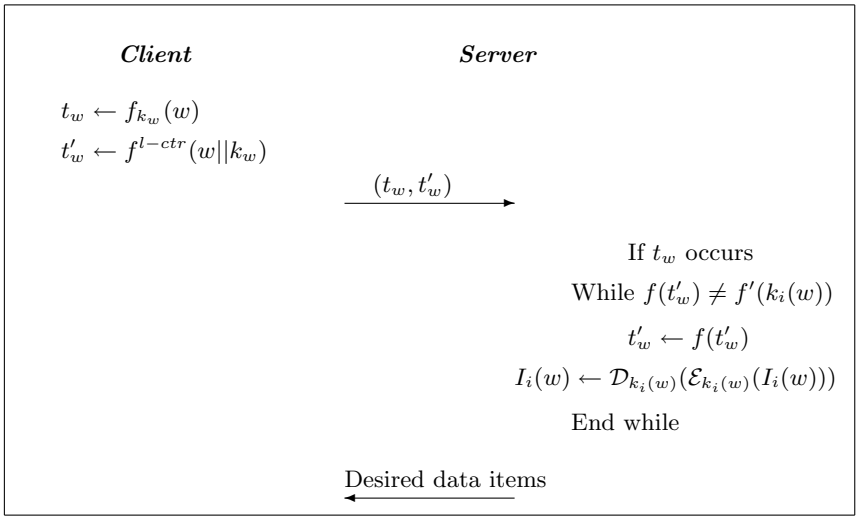


**Fig. 4.** The message exchange of the `Search` algorithm in scheme 2

### 5.7 Security of Updates

In the security proof of section 5.3 we do not consider the security of updating the database which is performed by the `MetadataStorage` algorithm. In fact there is information leakage in this case, specifically the amount of keywords in each update and information on which keywords are in common over several updates. For scheme 2 we did not discuss security at all. There the security is similar to that of scheme 1, but the improvement 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 `MetadataStorage` 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 traveler 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. In this case, since there is a balance between search and update (updating the record occurs before a search), both search and update are performed with high efficiency at a minimum cost.

## 7 Conclusion

We propose a novel searchable encryption scheme which has searching time logarithmic in the number of unique keywords stored on the server while it is efficiently updatable. We propose two variants of the approach which differ in the efficiency of the `Search` and the `MetadataStorage` algorithms. We now present a general assessment of the two schemes proposed. The first scheme is more efficient in terms of computation for the `Search` algorithm, but requires two rounds of communication between the server and the client for each search. Moreover, a large bandwidth for the `MatadataStorage` algorithm is required. The second scheme enables the client to invoke the `MetadataStorage` with a minimum bandwidth and high efficiency. However, the `Search` algorithm is efficient under the condition that the `MetadataStorage` and the `Search` algorithms are interleaved, and the maximum number of times the database is updated (the `MetadataStorage` algorithm is invoked) is limited. Table 1 summarizes the features of the schemes proposed.

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

**Table 1.** Summary of the features of the schemes proposed. In this table, $u$ is the number of unique keywords, $l$ is the length of the pseudo-random chain, and $x$ is the average number of times updating the database between every two searches occurs.

## References


1. Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. *J. Cryptology*, 21(3):350–391, 2008.
2. Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. Public key encryption with keyword search revisited. Cryptology ePrint Archive, Report 2005/191, 2005. `http://eprint.iacr.org/`.
3. Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In Menezes [18], pages 535–552.
4. John Bethencourt, Dawn Xiaodong Song, and Brent Waters. New constructions and practical applications for private stream searching (extended abstract). In *S&P*, pages 132–139. IEEE Computer Society, 2006.
5. Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.
6. Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E. Skeith III. Public key encryption that allows pir queries. In Menezes [18], pages 50–67.
7. Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In John Ioannidis, Angelos D. Keromytis, and Moti Yung, editors, *ACNS*, volume 3531 of *Lecture Notes in Computer Science*, pages 442–455, 2005.
8. Benny Chor and Niv Gilboa. Computationally private information retrieval (extended abstract). In *STOC*, pages 304–313, 1997.
9. Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.
10. Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. Cryptology ePrint Archive, Report 2006/210, 2006. `http://eprint.iacr.org/`.
11. Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM Conference on Computer and Communications Security*, pages 79–88. ACM, 2006.
12. Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. `http://eprint.iacr.org/`.
13. Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 112–131. Springer, 1997.
14. Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.
15. Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.
16. Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *FOCS*, pages 364–373, 1997.
17. Leslie Lamport. Password authentication with insecure communication. *Commun. ACM*, 24(11):770–772, 1981.
18. Alfred Menezes, editor. *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*. Springer, 2007.

19. Ministerie van Volksgezondheid, Welzijn en Sport. Informatiepunt bsn en landelijke epd.
20. Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55, 2000.
21. Juan Ramón Troncoso-Pastoriza, Stefan Katzenbeisser, and Mehmet Utku Celik. Privacy preserving error resilient dna searching through oblivious automata. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM Conference on Computer and Communications Security*, pages 519–528. ACM, 2007.