Simple and Efficient Searchable Symmetric Encryption with Application

Bram Leenders
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
b.c.leenders@student.utwente.nl

ABSTRACT

Searchable Symmetric Encryption (SSE) allows for data-confidentiality while allowing searches in the encrypted data. Curtmola et al. (CCS '06) formulated strong security definitions for SSE schemes and constructed the first schemes that are secure in their model. Up to now, their schemes set the current state-of-the-art in SSE both in terms of security and efficency.
We propose an alternative and very simple searchable encryption scheme that is secure under Curtmola et al.'s definition for *adaptive* security. We discuss the theoretical and practical efficiency of our scheme and compare it to the performance of Curtmola et al.'s scheme. For a very concrete application (database of FBI arrest records), we show that our scheme is more efficient in terms of storage and communication.
Interestingly, the simplicity of our scheme reveals some insights about the security of so-called index-based SSE schemes in general, which we informally discuss. We conclude with the claim that Curtmola et al.'s security definitions are too strong for many applications.

Keywords

Searchable Symmetric Encryption, Database Security, Data Confidentiality

1. INTRODUCTION

The trend of outsourcing information processing combined with an increasing public focus on privacy and data security has put new demands on encryption schemes. Traditional schemes require data to be decrypted before processing. For instance, when outsourcing to a public cloud, these schemes cannot be used, since clients have to either download, decrypt and process all data themselves, which renders outsourcing to the cloud useless, or give their decryption key to the cloud or a third party, which sacrifices data confidentiality.

This demand has motivated the design of new encryption schemes that preserve properties of encrypted data during the encryption process. Using these properties, certain operations can be performed on a ciphertext without exposing its encrypted contents. Examples of schemes and their properties are order preserving schemes [1], where plaintexts have the same lexicographic order as their ciphertexts and homomorphic encryption [8], which allows algebraic operations on encrypted data without requiring the decryption key.

In this paper, we discuss another type of property preserving encryption, called *searchable encryption*, which preserves search capabilities. Initially, with searchable encryption, a client encrypts data and an index to search the data, and uploads this to some service provider (like a cloud). When speaking of searchable *symmetric* encryption (SSE), we refer to a construction where a collection of documents and an index of these documents are encrypted with a symmetric-key algorithm. After this initial client-side encryption and indexing, the client can query the provider with encrypted queries for keywords, and the provider can answer with encrypted answers. During this process the provider should get no knowledge of the encrypted data or the sent queries, other than what can be statistically inferred from the access and search pattern.

An example for practical use of SSE is an email provider that allows users to store their email encrypted but still enables users to search their emails. Another example is a doctor who outsources storage of his patients personal information. Using SSE, he could store his database anywhere without exposing privacy-sensitive information and still be able to query patient records.

1.1 Background and Related Work

The notion of searchable symmetric encryption has been introduced by Song et al. [15] who aimed to restore search functionality in encryption schemes. Goh [9] was the first to formalize a security definition for index-based SSE, namely IND-CKA security, which defines semantic security against chosen keyword attacks on indices and documents. This means that an adversary cannot deduce any document's content from the index. Additionally, Goh also describes an IND-CKA secure SSE scheme using keyed hashfunctions in combination with a Bloom filter.

In this paper, we will use the state-of-the-art security definitions given by Curtmola et al. [6]. They provide formal definitions of non-adaptive and adaptive security for index-based searchable symmetric encryption. These definitions presuppose an encrypted index that can only be queried for a keyword w if a trapdoor for w is provided. The difference between adaptive and non-adaptive security, is that in the adaptive case the adversary can choose his attack based on previous results, whereas a non-adaptive adversary must choose his attack without knowledge of previous results.

Further research on SSE addresses even more functionality, such as searching for multiple keywords [5], ranked

20th *Twente Student Conference on IT* 24th January, 2014, Enschede, The Netherlands.

searching [5] and fuzzy keyword search [11]. Shen et al. [14] describe a construction that does not leak the search pattern, at the cost of decreased performance.

Besides these works, there are also schemes based on public key encryption. A public key based searchable encryption scheme is introduced by Boneh et al. [3], to allow anyone with the public key to add records to the database, whilst requiring a private key to search and decrypt entries.

Searchable encryption has not been the only proposed solution for the problem of searching in encrypted data. Early works by Ostrovsky [13] and Goldreich and Ostrovsky [10] describe the concept of Oblivious RAM. Unlike SSE, their construction additionally hides the access and search pattern (see Section 2.3) and allows for private writes (meaning indistinguishability from reads). Although oblivious RAM is theoretically secure and scales well, it is not efficient enough for practical usage.

1.2 Our Contribution

We present a new and simple index-based SSE scheme which we will prove to be adaptively secure. It can search a document collection with a computional complexity that is linear in the amount of documents. The simplicity of our construction comes at the cost of increased computional complexity on the client-side, which is linear in the amount of documents.

We will also give a brief discussion on the security of index-based SSE schemes in general. Our results indicate that encrypting the index does not provide additional data-confidentiality. Therefore, we argue that the established security definition of SSE is too strong in the case of index-based schemes. The further analysis of this and the design of an improved security definition is considered as interesting future work.

1.3 Outline

Before discussing SSE schemes, we have to introduce some notation and define adaptive security for SSE schemes. We will do so in Section 2.

Once we have this basis, we will give a recap of the adaptively secure scheme provided by Curtmola et al. [6] and discuss its performance (Section 3).

We will then define our SSE scheme (Section 4), prove adaptive security under the given definitions (5) and give a performance analysis (6). A comparison of our scheme with Curtmola et al.'s is also given in Section 6.2. We provide a discussion on adaptive security in general (7) and conclude in Section 8.

2. NOTATION AND DEFINITIONS

2.1 Notation

Let $\mathbf{D} = (d_1, \ldots, d_n)$ be a collection of $n = \mathsf{poly}(\lambda)$ files (e.g., documents). The function $id(d_i)$ returns the document identifier of document d_i, which can be any representation uniquely identifying the document.

Let $\Delta = (w_1, \ldots, w_m)$ be a pre-built dictionary of $m = \mathsf{poly}(\lambda)$ lexicographically ordered keywords of length polynomial in λ. The keyword used in the i-th query we denote by $s_i \in \Delta$. We denote the set of keywords matching a document by $u(d_i)$, and the set of documents matching a keyword by $\mathbf{D}(w_i)$.

The encrypted and permuted index (as visible to the server) is denoted by $I \in \{0,1\}^{m \times n}$. Every keyword corresponds to a row, and every document corresponds to a column. A decrypted entry $I[i][j]$ is 1 if and only if $d_j \in \mathbf{D}(w_i)$, i.e. if the document is a match for keyword w_i.

Given a tuple t, we refer to the i-th entry of t as $t[i]$. Given a vector $\mathbf{v}$, we refer to the i-th element of $\mathbf{v}$ as $\mathbf{v}_i$. As a convention, we always use a bold font to denote a vector. For two distribution ensembles X and Y, we denote computational indistinguishability by $X \equiv_c Y$. Let $x \oplus y$ denote the bitwise exclusive-or on l-bit strings; if $x = (x_1, \ldots, x_l)$ and $y = (y_1, \ldots, y_l)$ then $x \oplus y = (x_1 \oplus y_1, \ldots, x_l \oplus y_l)$. An element a chosen in a uniformly random way from a set A is denoted by $a \overset{\$}{\leftarrow} A$. The negligible function $\mathsf{negl}(n)$ is an unspecified function such that for any polynomial $f(n)$, there is an $N \in \mathbb{N}$ that satisfies $\mathsf{negl}(n) \leq \frac{1}{f(n)}$ for all $n \geq N$.

2.2 Searchable Symmetric Encryption Definition

We use Curtomola et al.'s [6] definition of an index-based SSE scheme over a dictonary Δ. Let $\mathsf{SSE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Trpdr}, \mathsf{Search}, \mathsf{Dec})$ be a collection of polynomial-time algorithms, such that;

- $K \leftarrow \mathsf{Gen}(1^\lambda)$: is a probabilistic algorithm run by the client to generate a key. It takes as an input a security parameter 1^λ and outputs a secret key K.
- $(I, \mathbf{c}) \leftarrow \mathsf{Enc}(K, \mathbf{D})$: is a probabilistic algorithm that takes a document collection $\mathbf{D}$ and a key K as input. It is run by the client to encrypt the documents and generate an encrypted index I and an encrypted document collection $\mathbf{c} = (c_1, \ldots, c_n)$.
- $t \leftarrow \mathsf{Trpdr}(K, w)$: is a deterministic algorithm run by the client to generate a trapdoor t for a keyword w, where $w \in \Delta$.
- $R \leftarrow \mathsf{Search}(I, t)$: is a deterministic algorithm run by the server to search for documents in $\mathbf{D}$ that contain a keyword w. It takes as input an encrypted index I describing $\mathbf{D}$ and a trapdoor t and outputs a set of document indentifiers such that $R[i] = id(\mathbf{D}(w)[i])$.
- $d_i \leftarrow \mathsf{Dec}(K, c_i)$: is a deterministic algorithm run by the client to decrypt a single encrypted document c_i. It takes as input a key K and a ciphertext c_i and outputs the unencrypted document d_i.

An index-based SSE scheme must satisfy the *correctness* condition:

$$\mathsf{Search}\,(I, \mathsf{Trpdr}(K, w)) = \mathbf{D}(w) \wedge \mathsf{Dec}(K, c_i) = d_i$$

for all $\lambda \in \mathbb{N}$, for all K generated by $\mathsf{Gen}(1^\lambda)$, for all $\mathbf{D} \subseteq 2^\Delta$, for all $(I, \mathbf{c})$ generated by $\mathsf{Enc}(K, \mathbf{D})$, for all $w \in \Delta$ and for all $1 \leq i \leq n$.

2.3 Security definition

Before defining adaptive security for SSE, we will first define some additional notation that will be used in the security definition. These notations are chosen to match with the definitions of Curtmola et al. [6].

DEFINITION 1. *(History) A history $H = (\boldsymbol{D}, \boldsymbol{s})$ over q queries, is a tuple including the document collection and the queried keywords. We denote the keywords queried for by $\boldsymbol{s} = (s_1, \ldots, s_q)$ where s_i is the keyword asked for in the i-th query, and every $s_i \in \Delta$. Note that it is possible to query twice for the same keyword, thus there may exist a pair s_i, s_j where $i \neq j$ but $s_i = s_j$.*

Definition 2. *(Access pattern) An access pattern $\alpha(H)$ induced from a history $H = (\boldsymbol{D}, \boldsymbol{s})$ contains the results of each query in H. Thus $\alpha(H) = (\boldsymbol{D}(s_1), \ldots, \boldsymbol{D}(s_q))$ is a vector containing the sets of document identifiers of the matched documents.*

Definition 3. *(Search pattern) The search pattern $\sigma(H)$ induced from a q-query history $H = (\boldsymbol{D}, \boldsymbol{s})$ is a $q \times q$ binary matrix such that $s_i = s_j \Leftrightarrow \sigma(H)[i][j] = 1$.*

When referring to a (fixed) output of $\alpha(H)$ and $\sigma(H)$, we denote the access pattern by α and the search pattern by σ.

Finally, we introduce the notion of a trace, which is the information from the history that is leaked to the server. This means that while a client generates a history of documents, queries and trapdoors, the server can induce the corresponding trace from the information visible to the server.

Definition 4. *(Trace): A trace $\tau(H) = (|d_1|, \ldots, |d_n|, \alpha(H), \sigma(H))$ contains the lengths of all documents in $\boldsymbol{D}$ and the access and search pattern induced by the input history H.*

Note that histories must not be uniquely linkable to traces, otherwise knowing the trace would leak the history. Thus, it is required that for every history H_1 there exists at least one history H_2 such that $H_1 \neq H_2$ but $\tau(H_1) = \tau(H_2)$.

2.3.1 Adaptive security for SSE

An adaptively secure scheme is secure against an adaptive adversary. By secure, we mean that the adversary should not be able to learn anything about the contents of the documents or the keywords, other than what can be statistically inferred from the access and search pattern. This means an adversary is allowed to see whether a query has been asked before and what results belong to an query. The adversary should not learn what keyword is asked for in the query.

We recall the simulation-based definition of adaptive security by Curtmola et al. [6]. The basic idea is to build a simulator which is given only the trace, and can produce an index, ciphertexts and trapdoors that are indistinguishable from the real index, ciphertexts and trapdoors. We allow the adversary to build the history linked to the trace adaptively; the adversary can query for a keyword, receive a trapdoor and query again. Since the adversary is polynomially bounded, it can only submit polynomially many queries.

Definition 5. *(Adaptive semantic security): Let $\mathsf{SSE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Trpdr}, \mathsf{Search}, \mathsf{Dec})$ be an index-based SSE scheme, $\lambda \in \mathbb{N}$ the security parameter, $q \in \mathbb{N}$, $\mathcal{A} = (\mathcal{A}_0, \ldots, \mathcal{A}_q)$ be a q-query adversary and $\mathcal{S} = (\mathcal{S}_0, \ldots, \mathcal{S}_q)$ be a q-query simulator and consider the probabilistic experiments $\boldsymbol{Real}_{\mathsf{SSE},\mathcal{A}}(k)$ and $\boldsymbol{Sim}_{\mathsf{SSE},\mathcal{A},\mathcal{S}}(k)$:*

```
$\boldsymbol{Real}_{\mathsf{SSE},\mathcal{A}}(\lambda)$
  $K \leftarrow \mathsf{Gen}(1^\lambda)$
  $(\boldsymbol{D}, st_\mathcal{A}) \leftarrow \mathcal{A}_0(1^k)$
  $(I, \mathbf{c}) \leftarrow \mathsf{Enc}(K, \boldsymbol{D})$
  $(s_1, st_\mathcal{A}) \leftarrow \mathcal{A}_1(st_\mathcal{A}, I, \mathbf{c})$
  $t_q \leftarrow \mathsf{Trpdr}_K(s_i)$
  For $2 \le i \le q$:
    $(s_i, st_\mathcal{A}) \leftarrow \mathcal{A}_i(st_\mathcal{A}, I, \mathbf{c}, t_1, \ldots, t_{i-1})$
    $t_i \leftarrow \mathsf{Trpdr}_K(s_i)$
  let $\boldsymbol{t} = (t_1, \ldots, t_q)$
  output $v = (I, \mathbf{c}, \boldsymbol{t})$ and $st_\mathcal{A}$
```

```
$\boldsymbol{Sim}_{\mathsf{SSE},\mathcal{A},\mathcal{S}}(\lambda)$
  $(\boldsymbol{D}, st_\mathcal{A}) \leftarrow \mathcal{A}_0(1^\lambda)$
  $(I, \mathbf{c}, st_\mathcal{S}) \leftarrow \mathcal{S}_0(\tau(\mathbf{D}))$
  $(w_1, st_\mathcal{A}) \leftarrow \mathcal{A}_1(st_\mathcal{A}, I, \mathbf{c})$
  $(t_1, st_\mathcal{S}) \leftarrow \mathcal{S}_1(st_\mathcal{S}, \tau(\mathbf{D}, s_1))$
  For $2 \le i \le q$:
    $(s_i, st_\mathcal{A}) \leftarrow \mathcal{A}_i(st_\mathcal{A}, I, \mathbf{c}, t_1, \ldots, t_{i-1})$
    $(t_i, st_\mathcal{S}) \leftarrow \mathcal{S}_i(st_\mathcal{A}, \tau(\mathbf{D}, s_1, \ldots, s_i)))$
  let $\boldsymbol{t} = (t_1, \ldots, t_q)$
  output $v = (I, \mathbf{c}, \boldsymbol{t})$ and $st_\mathcal{A}$
```

An SSE scheme is adaptively secure if for all polynomial-size adversaries $\mathcal{A}$ where q is polynomial in λ, there exists a non-uniform simulator $\mathcal{S}$ of polynomial size, such that no polynomial-size distinguishers $\mathcal{D}$ can distinguish between the distribution of the simulator's outputs and the adversary's outputs:

$$P(\mathcal{D}(v, st_\mathcal{A}) = 1 : (v, st_\mathcal{A}) \leftarrow \boldsymbol{Real}_{\mathsf{SSE},\mathcal{A}}(\lambda)) - \\ P(\mathcal{D}(v, st_\mathcal{A}) = 1 : (v, st_\mathcal{A}) \leftarrow \boldsymbol{Sim}_{\mathsf{SSE},\mathcal{A},\mathcal{S}}(\lambda)) \le \mathsf{negl}(\lambda)$$

where the probabilities are over the random coin tosses of Gen and Enc.

3. CURTMOLA ET AL.'S SCHEME

We will now discuss the adaptively secure construction given by Curtmola et al. [6]. This requires additional notation: let the family of a keyword w be defined by $FAM_w = \{w||1, \ldots, w||j\}$ where j is the amount of documents that contain w. An example of such a family is $FAM_{\text{foo}}\{\text{"foo1"}, \text{"foo2"}\}$ where there are two documents containing "foo". A single entry of a family is called a label, and all labels are unique since all keywords are unique. Let ℓ be the address length of the index: the length of an identifier of a single entry in the index. Simply put, the index should roughly be able to contain a match for every keyword-document combination, thus $\ell \ge \log_2(m \cdot n)$.

A fixed pseudo-random permutation $\pi : \{0,1\}^\lambda \times \{0,1\}^\ell \rightarrow \{0,1\}^\ell$ takes as input a secret key K and a label, and outputs a bitstring indistinguishable from random. This output is unique for a fixed key $K \in \{0,1\}^\lambda$ and any given label, and can thus be used to represent a label in an irreversible unique way (reversing is only possible with the key K).

The index I is an FKS dictionary [7] that stores document identifiers. The document identifiers are stored in such a way that if a document d is the j-th match for a keyword w, then the label is $w||j$ and $id(d) = I[\pi(K, w||j)]$. Thus, π evaluated over the family of a keyword gives all the positions in the index that store the document identifiers of the documents matching the keyword.

If $I[\pi(K, w||j)]$ is an empty cell, then we know that w matches $j-1$ documents and no further lookups (for any values higher than j) are needed. To ensure that adversaries cannot distinguish the index based on the amount of empty cells, the index is padded with document identifiers until it reaches its maximum size. This maximum size is the maximum amount of hits, i.e., keyword-document pairs for which the document matches the keyword.

The maximum size can be calculated by multiplying the number of documents by the number of keywords that fit inside a document. We refer to the maximum size as $s = n \cdot \mathtt{max}$, where `max` is the number of keywords that fit inside a document. Section 3.1 explains how to determine `max`.

The client, knowing w and K, can generate a trapdoor $t = (\pi(K, w||1), \ldots, \pi(K, w||n))$ which is a list of all addresses that may contain document identifiers. The server can check these addresses and return the results. These results are the document identifiers of the documents that matched the keyword.

We can write the following algorithm to describe Curtmola et al.'s SSE scheme:

- $K \leftarrow \mathsf{Gen}(1^\lambda)$: the client generates two secret keys $K = (K_1, K_2)$, where $K_1 \in \{0,1\}^\lambda$ and K_2 is generated to match the document encryption scheme.
- $(I, c) \leftarrow \mathsf{Enc}(K, \mathbf{D})$: the client encrypts every document in $\mathbf{D}$ with the key K_2 using a PCPA secure symmetric encryption scheme. The look-up table is calculated as follows:
  - For every keyword $w_i \in \Delta$:
    - Scan all documents, and generate the set of matching documents: $\mathbf{D}(w_i)$. Let $id(D_{i,j}$ be the j-th document matching w_i.
    - Set $I[\pi(K_1, (w_i||j))] = id(D_{i,j})$ The id of the document is stored in a pseudo-random location of the index.
  - For every document d:
    - Let $c = u(d)$: c is the number of keywords occurring in d.
    - For $1 \leq i \leq \texttt{max} - c$: $I[\pi(K_1, 0^\ell||n+i)] = id(d)$.

  This results in a look-up table with both matching and non-matching document identifiers stored in pseudo-random locations. Thus, every document identifier occurs exactly max times in I.

  Output the encrypted documents and the look-up table.
- $t \leftarrow \mathsf{Trpdr}(K, w_i)$: the client can calculate the set of used entries by calculating $(\pi(K, (w_i||1)), \ldots, \pi(K, (w_i||n)))$. This generates as many addresses as there are documents, in case all documents match the keyword.
- $X \leftarrow \mathsf{Search}(I, t)$: given set of addresses, the server outputs all non-null entries corresponding to those entries. Once the first empty entry is found, the following entries do not need to be checked.
- $d_i \leftarrow \mathsf{Dec}(K, c_i)$: given a key K_d and a ciphertext c_i, the client decrypts a document using the PCPA secure scheme used in Enc.

3.1 Calculating max

The number of keywords that can fit inside a document is limited by two factors: the number of distinct keywords and the size of a document. Note that we can use padding to make every document equally large, which allows us to use the same value of max for every document.

- Let $i = 0$, $\texttt{max} = 0$ and S be the document size in bytes.
- While $S > 0$:
  - If $2^{8\cdot i} \cdot i \leq S$, set $i = i + 1$, $\texttt{max} = \texttt{max} + 2^{8\cdot i}$ and $S = S - 2^{8\cdot i} \cdot i$
  - Otherwise, set $\texttt{max} = \texttt{max} + \frac{S}{i}$ and $S = 0$.
- Let $\texttt{max} = \min(\texttt{max}, |\Delta|)$: if there are not enough keywords to fill the entire document, use the size of the dictionary as max value.

For instance, when supposing the Oxford dictionary with 180,000 distinct keywords and a 1GB file, the $\texttt{max} = 180,000$ for the file can easily contain all keywords. Using the same dictionary, the max for a 1KB file is 628, since one can fit all distinct $2^8 = 256$ 1-byte keywords and at most $\frac{1000-2^8}{2} = 372$ 2-byte keywords fit in the remaining space.

3.2 Performance analysis

Querying requires a single round of communication. The trapdoors sent are of space complexity $O(n \cdot \ell)$, for the client sends n addresses of length ℓ. The results are $O(\mathbf{D}(w) \cdot log(n))$, since the document identifiers of matching documents are returned.

The serverside computional complexity is $O(\mathbf{D}(w))$, and the storage complexity is polynomial in n. The size of the index is $O(n \log(n))$ (n document identifiers of length $\log n$), which is asymptotically less than the size of the ciphertexts. The computional complexity on the clientside is $O(n)$, and the space complexity is $O(1)$.

Note that $\mathbf{D}(w) \leq n$, since at most all documents are matches for a keyword. For any applications with a constant chance that a keyword matches a document, that means $O(\mathbf{D}(w))$ equals $O(n)$. We could, for example, think of an indexed set of emails: if the amount of emails doubles, the amount of results for a query will probably double as well.

4. AN ADAPTIVELY SECURE CONSTRUCTION

Before describing the construction, we must first define an n-bit pseudo-random function f and a pseudo-random permutation π:

- $f : \{0,1\}^\lambda \times \{0,1\}^n \rightarrow \{0,1\}^n$
- $\pi : \{0,1\}^\lambda \times [1,m] \rightarrow [1,m]$

Where λ is the security parameter, n is the number of documents and m the number of keywords. Note that, since we use a fixed dictionary Δ with a known size m, we know m up front.

4.1 An adaptively secure SSE construction

We will now provide a construction of an SSE scheme that we prove adaptively secure in Section 5. The algorithms are defined as follows:

- $K \leftarrow \mathsf{Gen}(1^\lambda)$: the client generates a set of three secret keys $K = (K_d, K_f, K_p)$, for the document encryption, the row encryption and the table permutation respectively.
- $(I, \mathbf{c}) \leftarrow \mathsf{Enc}(K, \mathbf{D})$: the client encrypts every document in $\mathbf{D}$ with the key K_d using a PCPA secure symmetric encryption scheme, e.g. AES in CTR mode [12]. An encrypted and permuted index I is calculated as follows:
  - For every keyword w_i in Δ:
    - For all documents $d_j \in \mathbf{D}$; let $\mathcal{I}[i]_j = 1$ if $w_i \in u(d_j)$, otherwise set $\mathcal{I}[i]_j = 0$.
    - Reassign $\mathcal{I}[i] \leftarrow \mathcal{I}[i] \oplus f(K_f, w_i)$, which encrypts the row $\mathcal{I}[i]$.
  - Generate a permuted index I by applying π to the encrypted rows, such that for all $1 \leq i \leq m$: $I[\pi(K_p, i)] = \mathcal{I}[i]$.

Output the encrypted documents $\mathbf{c}$ and the encrypted and permuted index I.

- $t \leftarrow \mathsf{Trpdr}(K, w_i)$: using the key K_p, the client can calculate the trapdoor $t = (K_p(i), f(K_f, w_i))$. The trapdoor contains the position of the row in the permuted index corresponding to w_i and the encryption/decryption key for the row.
- $X \leftarrow \mathsf{Search}(I, t)$: given an index and a trapdoor, the server does the following:
  - it finds and decrypts the row $r = f(K_f, w_i) \oplus I[K_p(i)]$.
  - from the decrypted row, the server deduces the set of document identifiers $id(d_i)|d_i \in \mathbf{D} \wedge r[i] = 1$. Note that the server only has to know what document identifier corresponds to the i-th bit.
- $d_i \leftarrow \mathsf{Dec}(K, c_i)$: given a key K_d and a ciphertext c_i, the client decrypts the encrypted document c_i using the key K_d of the PCPA secure scheme used in Enc.

5. SECURITY ANALYSIS

We will prove the given construction to be adaptively secure. We will first define the q-query simulator $\mathcal{S} = (\mathcal{S}_0, \ldots, \mathcal{S}_q)$ that, given a trace $\tau(H)$, generates $v^* = (I^*, c^*, t^*)$ and a state $st_{\mathcal{A}}$. The simulator $\mathcal{S}_0$ only creates an index, as no keywords have been queried for at this stage. The i-th simulator $\mathcal{S}_i$ returns trapdoors up until the i-th query. We will then prove that no polynomial-size distinguisher $\mathcal{D}$ can distinguish between the distributions of v^* and the outputs of an adaptive adversary that runs the real algorithm.

- $\mathcal{S}_0(1^k, \tau(D))$: given $(|d_1|, \ldots, |d_n|)$, choose $I^* \xleftarrow{\$} \{0, 1\}^{m \times n}$. Recall that m is public as it is the size of the dictionary Δ, and that n is included in the trace as the number of $|d_i|$'s.

  The ciphertexts are simulated by creating random strings of the same lengths as the documents; $c_i^* \xleftarrow{\$} \{0, 1\}^{|d_i|}$, where $|d_i|$ is included in the trace. Also, a random permutation $p^* : [1, m] \rightarrow [1, m]$ is generated.

  The simulator stores I^*, p^* and a counter $c = 0$ in the state $st_{\mathcal{S}}$, and outputs $v^* = (I^*, c^*, st_{\mathcal{S}})$.
- $\mathcal{S}_i(st_{\mathcal{S}}, \tau(D, s_1, \ldots, s_i))$ for any $1 \leq i \leq q$: given the state (which includes I^* and any previous trapdoors) and the access pattern α, the simulator can generate a trapdoor t_i^* as follows:
  - Check if the keyword has been queried before; if there is a $j \neq i$ such that $\sigma[i][j] = 1$, set $t_i^* = t_j^*$. Otherwise:
    - Increase the counter by one and generate a unique row index $p^*(c)$, using the counter and the random permutation. Note that the counter will never exceed m, as there are only m unique keywords.
    - Calculate a bitstring $r \in \{0, 1\}^n$ such that for $1 \leq j \leq n$: $r[j] = 1 \Leftrightarrow id(d_j) \in \alpha[i]$. We now have what should be the unencrypted row of the index corresponding to the keyword queried for.
    - Calculate $k^* = r \oplus I^*[p^*(c)]$. We now have a dummy key which satisfies the property $k^* \oplus I^*[p^*(c)] = r$.
    - Let $t_i^* = (p^*(c), k^*)$

  Include t_i^* in $st_{\mathcal{S}}$, and output $(t_i^*, st_{\mathcal{S}})$.

We will now show that the outputs of $\mathbf{Real}_{SSE,\mathcal{A}}$ and $\mathbf{Sim}_{SSE,\mathcal{A},\mathcal{S}}$, being v and v^*, can not be distinguished by a distinguisher $\mathcal{D}$ that is given $st_{\mathcal{A}}$. Recall that $v = (I, \mathbf{c}, t_1, \ldots, t_q)$ and $v^* = (I^*, \mathbf{c}^*, t_1^*, \ldots, t_q^*)$.

- (Indistinguishability of I and I^*) The output of $f(K_f, w_i)$ is indistinguishable from random by definition of f. Therefore, the XOR of entries of the index and the output of f is indistinguishable from random bitstrings of the same length [2]. Since I^* is a random bitstring of the same length as I, and with all but negligible probability $st_{\mathcal{A}}$ does not contain the key, we conclude that I and I^* are indistinguishable.
- (Indistinguishability of c_i and c_i^*) Since c_i is PCPA-secure encrypted, c_i cannot be distinguished from a random string. Since every c_i^* is random and of the same length as c_i, and with all but negligible probability $st_{\mathcal{A}}$ does not contain the encryption key, c_i is distinguishable from c_i^*.
- (Indistinguishability of $t_i = (K_p(i), k = f(K_f, w_i))$ and $t_i^* = (p^*(c), k^*)$) With all but negligible probability, $st_{\mathcal{A}}$ will not contain the key K_p, so the pseudorandomness of π guarantees that each $p^*(c)$ is computionally indistinguishable from $\pi(K_p, i)$.

  As stated above, I and I^* are indistinguishable, thus $I[i]$ and $I^*[i]$ are indistinguishable, thus $I[i] \oplus r = k$ and $I^*[i] \oplus r = k^*$ are indistinguishable. Thus, t_i and t_i^* are indistinguishable.

6. PERFORMANCE ANALYSIS

6.1 Asymptotical Analysis

Our construction requires only a single round of communication for each query. The communication cost is $O(log(m) + n)$, since the trapdoors contain the row location and the decryption key, which is as long as the row.

The computional complexity on both the client and the server is proportional to the amount of documents. This is of complexity $O(n)$, but with a very low constant because we can choose an efficient pseudo-random function, and the decryption process is simply XORing data. Modern CPUs can perform XOR operations at a speed of multiple gigabytes per second, which corresponds to tens of billions entries per second.

The client-side storage demands are $O(1)$. The server-side storage space complexity is dependent on both the amount of words and the amount of documents ($O(m \cdot n)$). Concretely, the server must store a single bit for every keyword-document combination which is 1 if and only if the document is a match for the keyword.

6.2 Concrete Analysis

To compare our scheme with Curtmola et al's, we will consider the following application scenario. Note that the numbers may not be fully representative, since this is for illustrative puposes only.

Suppose the Federal Bureau of Investigation wants to outsource the storage of all arrest records over a period of twenty years without exposing these records. Annually, 13 million arrests are reported to the Uniform Crime Reporting program (see [16] and [17]), which amounts to a dataset of roughly 250 million records. Given that a record is one

Scheme comparison

Scheme	Index size	Query size	Communication rounds
Our scheme	640GB	31MB	1 round
Curtmola et al.	18TB	1.3GB	1 round

Table 1. The performance of the schemes discussed in Section 6.2.

megabyte in size (including the mugshot), the dataset is 250 terabyte. Each ticket is tagged with a state (of which there are 50), nearest city (estimated to 500 per state), an offense type (from a set of 40) and the responsible special agent (supposing an active force of 20,000).

This adds up to a dictionary with 20590 keywords, and a billion keyword/document combinations that form a match ($w_i \in \Delta, d_j \in \mathbf{D}$ such that $w_i \in u(d_j)$) since every record matches four tags.

6.2.1 Our Construction

The index size of our scheme is, optimally, $250,000,000 \times 20590$bit$\approx$ 640GB in size, since the matrix contains one bit for every keyword-document combination.

A trapdoor is $\log_2(m)+n$ bit $\approx$ 31MB in size, since it contains a row address (of length $\log_2(m)$ bit) and a decryption key of one bit per document. The key-generation/ matching/decryption takes a negligible amount of time (a modern processor can generate stream AES in CTR mode at multiple gigabytes per second, thus key generation takes milliseconds). Sending the trapdoor would take several seconds over a fast LAN, but several minutes over ADSL.

6.2.2 Curtmola et al.'s Scheme

Curtmola et al.'s implementation stores $s = \mathtt{max} \cdot n$ entries in the index (see Section 3.1 for an explanation of `max`). By storing an entry for every keyword-document combination, adversaries cannot see how many actual matches exist. Every entry stores a document identifier of $\log_2(n)$ bits. Since we have 1MB documents `max` is set at the dictionary size (20590 keywords), since that is less than the 355349 keywords that might fit in the documents.

The FKS dictionary can store n items with space complexity $O(n)$. We will suppose a dictionary without overhead for addressing, which requires exactly as much storage as its contents. Thus, the size of the index is $\log_2(n) \cdot s$ bits $\approx$ 18TB.

This is significantly larger than the 640GB our scheme needs to store the index. The reason for this is that Curtmola et al.'s scheme must store a document identifier of several bytes for every possible combination, whereas our scheme only has to store a single bit for every combination.

Recall that the trapdoors in the scheme are evaluations of a pseudo-random permutation π over the family of a keyword: $t = (\pi(K, w||1), \ldots, \pi(K, w||n))$. The output length of a single evaluation of π is ℓ bits long (since the output is a virtual address). Thus, the size of a trapdoor is $n \cdot \ell$, where $\ell \geq \log_2(s)$ is 43 bit. Such a trapdoor is 1.3GB, which is significantly larger than the 31MB than our scheme. Again; this is because Curtmola et al.'s scheme sends identifiers, rather than single bits.

Table 1 gives a summary of the performances of the schemes mentioned above.

7. DISCUSSION

7.1 Leaking of Row Plaintexts

In order to satisfy the adaptive security definition, we have to encrypt the rows of our index. Since encryption and decrypting is an expensive process, we would like to minimize the amount of data that needs to be encrypted. If we could somehow skip the encryption of the index, relying only on permutation of the row order, this would drastically lower our computional and bandwidth demands. This raises the question of what is lost by not encrypting the index, only permuting the row order.

By using the index, users leak their access and search pattern. This leakage allows the simulator to simulate the index and trapdoors, since it knows what the results should be. Thus, the simulator learns what the plaintext rows of the index are. Once all keywords have been asked for, all row plaintexts are leaked (though still in permuted order). This means an adversary knows what documents match a keyword, just not what the keyword is; the adversary knows the answer, but not the question. It's almost like a game of Jeopardy.

If we allow this information to be leaked during usage of an encrypted database, why would we not allow it to be known beforehand? If the row plaintexts are leaked, why not send them unencrypted, just in permuted order? In a way, this is what Curtmola et al.'s scheme [6] does: the document id's are not encrypted, just in a permuted order.

In our scheme, encrypting the rows is nessacery to allow the simulator to choose the row plaintexts adaptively. Recall that our simulator chooses the key for decrypting a row after is has learned what the plaintext of the row should be. Without being able to do this, the scheme would not be adaptively secure under this definition. It would not even be non-adaptively secure, since the simulator only knows the access pattern for queries that have been asked, which may not be covering the entire set of keywords.

7.2 Reducing the Index Size

In this Section, we will differentiate between ID storing indices, like Curtmola et al.'s [6] and boolean storing indices, like ours. The difference is that the former stores document id's of the results, whereas the latter stores bits indicating whether a keyword-document pair is a match, and relies on the position of a bit to indicate what document it corresponds to.

7.2.1 ID Storing Indices

As demonstrated in the performance analysis, the indices produced by our scheme and Curtmola et al.'s scheme are far larger than a minimal index. Storing a mere billion hits can be done in several gigabytes, as opposed to hundreds of gigabytes or terabytes, if we sacrifice adaptive security.

Storing the id's of documents that form a match with a keyword, like Curtmola et al.'s scheme [6], requires the storage of non-matches as well. Not storing these non-matches requires knowledge of how many matches exist, which the adaptive simulator cannot have beforehand. This requires a match-storing scheme to have an index of size at least $\log_2(n) \cdot s$ bit.

This implementation has room for significant efficiency improvement, if we are willing to leak some information. Knowing that every document is matched by exactly four keywords, we can set `max` to four. Given that information, $s = 4 \cdot n$, and the size of the index is approximately 3.5GB. Allowing this `max` value, adversaries can deduce that documents match (exactly) four keywords, but they still cannot deduce what keywords a document matches.

Making `max` small and public would also affect the communication cost; since the lookup-table stores less entries, the virtual address length can be decreased. This would cause queries to be approximately 950MB, as opposed to 1.3GB, saving over a quarter of the data traffic. Note that the query size is almost a third of the index size, which implies that it may be more efficient to download the index and search locally.

7.2.2 Boolean Storing Indices

Storing a matrix where each entry is a bit indicating whether the corresponding keyword-document pair is a match requires $n \cdot m$ bit, optimally. This index could be compressed using common compresion algorithms, but such compression is outside the scope of this paper.

Any compression of this data requires the data to be non-random in some way. The real scheme might be able to compress the index if, for example, only one percent of all keyword-document pairs is actually a match. The simulator cannot simulate the compression, since the simulator has no knowledge of the contents of the documents.

If we allow the plaintext of the index to be visible, as explained above and in our leaking version of Curtmola et al.'s scheme, we can compress the index to a fraction of the adaptively secure indices.

7.3 Query optimization

Curtmola et al. suggest a change to their scheme, to lower the communication cost: by storing encryptions of $|\mathbf{D}(w)|$, the client can query for that information first and then send the right amount of evaluations of π. This will result in two rounds of communication:

1. Query 1: the client asks for a single evaluation of π, which is ℓ bits.
   Response 1: the server responds with a single document id of $\log_2(n)$ bits and $|\mathbf{D}(w)|$ encrypted. $|\mathbf{D}(w)|$ is on average $\frac{\Sigma_{w\in\Delta}|d(w)|}{|\Delta|} \approx 4,8 \cdot 10^4$.
2. Query 2: the clients asks on average for $|\mathbf{D}(w)| - 1$ evaluations of π. In our case, such a query is on average $\ell\frac{4n}{|\Delta|}$ bit $\approx$ 180KB (supposing keywords have equal chance of being asked).
   Response 2: the response is on average $\log_2(n)\frac{4n}{|\Delta|} \approx$ 170KB.

We consider this not a good change, since it adds a communication round. If we allow an extra round of communication, there are a lot more searchable encryption schemes we could use. For example, Bösch et al. [4] propose an efficient scheme that even hides the access pattern. Therefore, we deem this change to the scheme not a good one, for there are better alternatives using two rounds of communication.

Table 2 gives an overview of the performances of our scheme (Section 4), Curtmola et al.'s scheme (Section 3), the leaking implementation (Section 7.2.1) and Curtmola et al.'s bandwidth-saving scheme (Section 7.3).

8. CONCLUSION

We have presented a very simple adaptively secure searchable symmetric encryption scheme. The performance is better than Curtmola et al.'s performance when only a single round of communication is allowed.

Scheme comparison

Scheme	Index size	Query size	Communication rounds
Our scheme	640GB	31MB	1 round
Curtmola et al.	18TB	1.3GB	1 round
Reduced index size	7GB	1.3GB	1 round
Curtmola et al. bandwidth saving	18TB	350KB	2 rounds

Table 2. The performance of the schemes discussed above.

As demonstrated in Sections 6.2.2 and 7, one can sacrifice confidentiality of some information to massively improve the performance of either scheme. This information is leaked via the access pattern during the usage, therefore we see no problem in leaking it a priori even though it is not adaptively secure.

Looking at the sacrifices we must do to achieve adaptive security for SSE, and its limited security guarantees, we ask the question whether this is really worth it. The theoretical value of adaptive security we do not doubt; proving a scheme to be adaptively secure under the definitions used in this paper gives strong guarantees as to whether certain information is leaked or confidental. However, for practical usage, one should either choose a weaker scheme in terms of security to improve efficiency or choose a scheme that achieves even stronger security, for example a scheme that also hides the access pattern.

Acknowledgements

The author would like to thank Andreas Peter for the helpful comments and discussions and Christoph Bösch for posing the original research question.

9. REFERENCES

[1] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 563–574. ACM, 2004.

[2] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 394–403. IEEE, 1997.

[3] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Advances in Cryptology-Eurocrypt 2004*, pages 506–522. Springer, 2004.

[4] C. Bösch, Q. Tang, P. Hartel, and W. Jonker. Selective document retrieval from encrypted database. In *Information Security*, pages 224–241. Springer, 2012.

[5] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou. Privacy-preserving multi-keyword ranked search over encrypted cloud data. In *INFOCOM, 2011 Proceedings IEEE*, pages 829–837. IEEE, 2011.

[6] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 79–88. ACM, 2006.

[7] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with 0 (1) worst case access time. *Journal of the ACM (JACM)*, 31(3):538–544, 1984.

[8] C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

[9] E.-J. Goh. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.

[10] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, May 1996.

[11] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou. Fuzzy keyword search over encrypted data in cloud computing. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–5. IEEE, 2010.

[12] H. Lipmaa, D. Wagner, and P. Rogaway. Comments to nist concerning aes modes of operation: Ctr-mode encryption. 2000.

[13] R. Ostrovsky. Efficient computation on oblivious rams. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 514–523. ACM, 1990.

[14] E. Shen, E. Shi, and B. Waters. Predicate privacy in encryption systems. In *Theory of Cryptography*, pages 457–473. Springer, 2009.

[15] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 44–55. IEEE, 2000.

[16] United States Department of Justice, Federal Bureau of Investigation. *Crime in the United States, 1995.* `http://www.fbi.gov/about-us/cjis/ucr/crime-in-the-u.s/1995/95sec4.pdf`. Retrieved: 2013-12-28.

[17] United States Department of Justice, Federal Bureau of Investigation. *Crime in the United States, 2012.* `http://www.fbi.gov/about-us/cjis/ucr/crime-in-the-u.s/2012/crime-in-the-u.s.-2012/tables/29tabledatadecpdf`. Retrieved: 2013-12-28.