Privacy Preserving Keyword Searches on Remote Encrypted Data

Yan-Cheng Chang
Harvard University
ycchang@eecs.harvard.edu

Michael Mitzenmacher
Harvard University
michaelm@eecs.harvard.edu

Abstract

We consider the following problem: a user $\mathcal{U}$ wants to store his files in an encrypted form on a remote file server $\mathcal{S}$. Later the user $\mathcal{U}$ wants to efficiently retrieve some of the encrypted files containing (or indexed by) specific keywords, keeping the keywords themselves secret and not jeopardizing the security of the remotely stored files. For example, a user may want to store old e-mail messages encrypted on a server managed by Yahoo or another large vendor, and later retrieve certain messages while travelling with a mobile device.

In this paper, we offer solutions for this problem under well-defined security requirements. Our schemes are efficient in the sense that no public-key cryptosystem is involved. Indeed, our approach is independent of the encryption method chosen for the remote files. They are also incremental, in that $\mathcal{U}$ can submit new files which are totally secure against previous queries but still searchable against future queries.

1 Introduction

We consider the following distributed file system: a user $\mathcal{U}$ pays a file server $\mathcal{S}$ for storage service, with the goal being that $\mathcal{U}$ can retrieve the stored files anytime and anywhere through Internet connections. For example, $\mathcal{U}$ may store files containing personal data that $\mathcal{U}$ may want to later access using his wireless PDA. A user might be willing to pay for such a service in order to have access to data without carrying devices with large amount of memory, and to have the data well-maintained by professionals. Such distributed file services already exist, such as the "Yahoo! Briefcase".[1] We expect such services will grow with the expansion of mobile and pervasive computing.

In many cases $\mathcal{U}$ will not want to reveal the contents of his files to $\mathcal{S}$ in order to maintain security or privacy. It follows that the files will often be stored in encrypted form. Suppose, however, that later $\mathcal{U}$ wants to retrieve files based on a keyword search. That is, $\mathcal{U}$ wants to retrieve files containing (or indexed by) some keyword. If the files are encrypted, there is no straightforward way for $\mathcal{S}$ to do keyword search unless $\mathcal{U}$ is willing to leak the decryption key. A trivial solution that preserves the security of $\mathcal{U}$'s files is to have $\mathcal{S}$ send all the encrypted files back to him. This may not be feasible solution if $\mathcal{U}$ is using mobile devices with limited bandwidth and storage space. An additional complication is that $\mathcal{U}$ may naturally also want to keep secret the keyword that he is interested in as well.

We provide practical solutions to this problem with strong theoretical security guarantees that require only small amounts of overhead in terms of bandwidth and storage, as we describe more fully in the main text. Our solution utilizes the notion of a keyword index, which is created by $\mathcal{U}$. The keyword index associates each keyword with its associated files. We picture the keyword index being created offline, with a more powerful home machine, before the user wishes to access

[1]Emails are actually a typical example, as they are stored on remote servers [BCOP03].

the files remotely with a mobile device. All keyword searches by $\mathcal{U}$ are based on this index; hence our scheme does not offer full pattern-matching generality with the actual text. In practice, this should be sufficient for most users. It is worth noting that in this framework $\mathcal{U}$ can have complete control over what words are keywords and which keywords are associated with which files, a power that can be useful for many applications.

We take care in defining a proper notion of security for this problem. Intuitively, after processing one of $\mathcal{U}$'s queries, $\mathcal{S}$ learns something: it learns that the encrypted files that $\mathcal{S}$ returns to $\mathcal{U}$ share some keyword. We want this to be all that $\mathcal{S}$ learns. We formalize this notion in cryptographic terms and prove that our schemes satisfies our formalization.

To set up our solution, we clarify further our methodology and our contributions. Our solutions are two-phased. In the first phase, we assume $\mathcal{U}$ is at home and is going to submit his files to $\mathcal{S}$, and assume that sufficient space is available to store a dictionary. In the second phase, we assume $\mathcal{U}$ becomes a mobile user and wants to retrieve some encrypted files from $\mathcal{S}$ by keyword searches. This is a very natural framework describing realistic distributed computing situations. We consider both the case that $\mathcal{U}$ can store a dictionary on his mobile device and the case that he cannot. The first case may be practical in some situations, where the mobile device has sufficient storage, and is useful for framing the solution to the second case, which is our main result.

Our main idea is the following: we let $\mathcal{U}$ use pseudo-random bits to mask a dictionary-based keyword index for each file and send it to $\mathcal{S}$ in such a way that later $\mathcal{U}$ can use the short seeds to help $\mathcal{S}$ recover selective parts of the index, while keeping the remaining parts pseudo-random. This requires some additional storage overhead on $\mathcal{S}$ as we clarify later.

No public-key cryptosystem is required in our schemes; only pseudo-random functions are used. We claim that this property significantly increases the practicability of our schemes, since in practice heuristic pseudo-random functions (that is, functions that appear pseudo-random enough for the specific application) can be implemented efficiently. Moreover, because our methodology is independent of the encryption method chosen for the remote files, our schemes have the advantage of working for different file formats (including compressed files, multimedia files, etc.), as long as a keyword index on the corresponding content can be built a priori.

Last but not least, we solve the update problem, which says how to ensure the security of the consequent submissions in presence of previous queries. Our solution enjoys very simply key management. Deletion is also simple in all our schemes.

1.1 Related Work

In theory, the classical work of Goldreich and Ostrovsky [GO96] on oblivious RAMs can resolve the problem of searching on encrypted data. Although their scheme is asymptotically efficient and nearly optimal, it does not appear to be efficient in practice as large constants are hidden in the big-O notation.

A closely related cryptographic problem is Single Database **P**rivate **I**nformation **R**etrieval (PIR for short) [KO97]. PIR is a protocol between two parties: $\mathcal{S}$, who has an n-bit string x, and $\mathcal{U}$, who has an index $1 \leq i \leq n$. The goal is for $\mathcal{U}$ to learn the i-th bit of x without leaking i, while sending fewer than n bits from $\mathcal{S}$ to $\mathcal{U}$ (as otherwise the problem becomes trivial). PIR protocols can be used to construct Oblivious Transfer protocols [CMO00], which are complete for all secure computations [K88]. PIR differs substantially from our problem. Part of the difference is perhaps cosmetic: our units are not individual bits but encrypted files and keywords. More importantly, our security guarantee differs from that of PIR.

The question how to do keyword searches on encrypted data was previously raised in [SWP00].

In that paper, they proposed a scheme which encrypts each word (or each pattern) of a document separately. Such an approach has the following disadvantages. First, it is not compatible with existing file encryption schemes. Instead, a specific encryption method must be used. Second, it cannot deal with compressed data, while we believe users will often want to save in storage costs by compressing their files, since generally the service fee is proportional to the storage space. Finally, as the authors themselves acknowledge, their scheme is not secure against statistical analysis across encrypted data [SWP00]. Although some heuristic remedies (and an index construction alternative) were proposed, their security proof is at least not theoretically sound.

Recently, a new scheme aiming to solve this problem was proposed in [G04]. The idea of that scheme is to build an index of keywords for each file using a Bloom filter [B70], with pseudo-random functions used as hash functions. When $\mathcal{U}$ submits a document to $\mathcal{S}$, he also submits the corresponding Bloom filter. We suspect that the security model defined in [G04] is insufficient, as the scheme cannot resist certain attack. Specifically, because the number of 1's of a Bloom filter is (roughly) proportional to the number of the distinct keywords of a document, some information is immediately leaked from the Bloom filters themselves, and we believe that statistical analysis based on this (and related) facts can compromise the security of the encrypted files.[2] On the other hand, one inherent problem with this Bloom-filter-based approach is that Bloom filters inevitably induce false positives, which would potentially cause mobile users may download extra files not containing the keyword. While sufficiently rare false positive might be acceptable, we note that our scheme avoids this problem.

As further related work, [BCOP03] studies the problem how to search on data encrypted by a public-key cryptosystem. In particular, they consider the problem of a user that wants to retrieve e-mails containing a certain keyword from his e-mail server, with the e-mails encrypted by the user using his public key. The problem setting is related to but different from ours.

2 Preliminaries

We use the notation $a \leftarrow A$ to denote choosing an element a uniformly at random from the set A, and use *PPT* to denote *probabilistic polynomial time*. Also, for a positive integer $n \in \mathbb{N}$, let $[n]$ denote the set $\{1, 2, \cdots, n\}$; for a string s, let $s[i]$ denote its i-th bit; for a function f, let $|f|$ denote its output length. Moreover, we say a function is negligible in t if for any polynomial p there exists a t_0 such that for all $t > t_0$ we have $f(t) < 1/p(t)$. Last, all logarithms in this paper have base 2.

2.1 Cryptographic basics

For completeness we first define pseudo-random permutations and functions. Our definitions are standard; see, e.g., [G01].

Definition 1. (Pseudo-random permutations) *We say a permutation family* $\{P_K : \{0,1\}^n \rightarrow \{0,1\}^n | K \in \{0,1\}^t\}$ *is* pseudo-random *if it satisfies the following:*

- *Given* $x \in \{0,1\}^n$ *and* $k \in \{0,1\}^t$*, there is a PPT algorithm to compute* $P_k(x)$*.*
- *For any PPT oracle algorithm* A*, the following value is negligible in* t*:*

$$|\mathbf{Pr}_{k \leftarrow \{0,1\}^t}[A^{P_k}(1^t) = 1] - \mathbf{Pr}_{p \leftarrow U_p}[A^{p}(1^t) = 1]|,$$

where U_p *is the set of all the permutations on* $\{0,1\}^n$*.*

[2] The latest version of [G04] proposes a new security model and a new scheme to deal with this problem.

Definition 2. (Pseudo-random functions) *We say a function family* $\{F_K : \{0,1\}^n \rightarrow \{0,1\}^m | K \in \{0,1\}^t\}$ *is* pseudo-random *if it satisfies the following:*

- *Given* $x \in \{0,1\}^n$ *and* $k \in \{0,1\}^t$, *there is a PPT algorithm to compute* $F_k(x)$.
- *For any PPT oracle algorithm* A, *the following value is negligible in* t*:*

$$|\mathbf{Pr}_{k \leftarrow \{0,1\}^t}[A^{F_k}(1^t) = 1] - \mathbf{Pr}_{f \leftarrow U_f}[A^f(1^t) = 1]|,$$

where U_f *is the set of all the functions mapping* $\{0,1\}^n$ *to* $\{0,1\}^m$.

Next, the following lemma says it is safe to feed pseudo-random functions with pseudo-random seeds instead of truly random seeds.

Lemma 1. *Consider two pseudo-random function families* $\{F_K : \{0,1\}^n \rightarrow \{0,1\}^m | K \in \{0,1\}^t\}$ *and* $\{G_K : \{0,1\}^\ell \rightarrow \{0,1\}^t | K \in \{0,1\}^t\}$. *For any PPT oracle algorithm* A *and any* $x \in \{0,1\}^\ell$, *the following value is negligible in* t*:*

$$|\mathbf{Pr}_{\sigma \leftarrow \{0,1\}^t, k = G_\sigma(x)}[A^{F_k}(1^t) = 1] - \mathbf{Pr}_{f \leftarrow U_f}[A^f(1^t) = 1]|,$$

where U_f *is the set of all the functions mapping* $\{0,1\}^n$ *to* $\{0,1\}^m$.

Proof. If (A, x) is a counterexample, then there is a construction of a PPT algorithm B using A, x, F_K such that the following value is not negligible in t:

$$|\mathbf{Pr}_{\sigma \leftarrow \{0,1\}^t}[B^{G_\sigma}(1^t) = 1] - \mathbf{Pr}_{g \leftarrow U_g}[B^g(1^t) = 1]|,$$

where U_g is the set of all the functions mapping $\{0,1\}^\ell$ to $\{0,1\}^t$. Clearly, it induces a contradiction. □

In practice, we can use HMAC-SHA1 [BCK96] to implement a pseudo-random function. Also, it is well known that a pseudo-random permutation can be constructed using a pseudo-random function in three rounds [LR85, NR97].

2.2 Problem setting

We define the problem of **P**rivacy **P**reserving Keyword **S**earches on Remote **E**ncrypted **D**ata (PPSED for short) in this section, and will hereafter use PPSED to denote this problem. Recall that we allow the user $\mathcal{U}$ to specify the relationship between files and keywords. That is, $\mathcal{U}$ can associate any collection of keywords with a file. Generally, when files are text files, keywords will be actual words of text. In order to formalize a clear definition, we only consider queries containing a single keyword. We emphasize that to deal with queries containing Boolean operations on multiple keywords in the security setting of PPSED remains a challenging open problem.

The formal definition of PPSED is as follows:

Definition 3. (PPSED) PPSED *is a multi-round protocol between a remote file server* $\mathcal{S}$ *and a user* $\mathcal{U}$. $\mathcal{S}$ *has a set of* n *encrypted files* $\zeta = \{\mathcal{E}_1(m_1), \mathcal{E}_(m_2), \cdots, \mathcal{E}_n(m_n)\}$ *where for each* $i \in [n]$, $\mathcal{E}_i$ *is an encryption function and* m_i *is a file. The user* $\mathcal{U}$ *has decryption algorithms* $\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_n$ *such that* $\mathcal{D}_1(\mathcal{E}_1(m_1)) = m_1, \mathcal{D}_2(\mathcal{E}_2(m_2)) = m_2, \cdots, \mathcal{D}_n(\mathcal{E}_n(m_n)) = m_n$. *Moreover, in each round* $j \in \mathbb{N}$, $\mathcal{U}$ *prepares a keyword* $w_j \in \{0,1\}^*$. *An implementation of* PPSED *with security parameter* t *must satisfy the following:*

1. *Correctness: In round j, for $i \in [n]$, if w_j is a keyword of m_i, $\mathcal{U}$ can obtain $\mathcal{E}_i(m_i)$.*

2. *Limits on the bandwidth and the storage space:*
   Consider round j, and let $\mathcal{I}_j = \{i | \ i \in [n], w_j \text{ is a keyword of } m_i\}$.

   - *The number of bits sent from $\mathcal{S}$ to $\mathcal{U}$ is $\sum_{i \in \mathcal{I}_j} |\mathcal{E}_i(m_i)| + O(1)$.*
   - *The number of bits sent from $\mathcal{U}$ to $\mathcal{S}$ is $O(t)$ per keyword search.*
   - *The number of bits stored on $\mathcal{U}$ is $O(t)$.*

3. *Security requirement:*
   For $k \in \mathbb{N}$, let C_k be all the communications $\mathcal{S}$ receives from $\mathcal{U}$ before round k, and let $C_k^* = \{\zeta, Q_0 \equiv \emptyset, Q_1, \cdots, Q_{k-1}\}$, where for each $j \in [k-1]$, Q_j is an n-bit string such that for $i \in [n]$, $Q_j[i] = 1$ if and only if w_j is a keyword of m_i.

   - *For $k \in \mathbb{N}$, for any PPT algorithm A, any $\Delta_k = \{m_1, \cdots, m_n, w_0 \equiv \emptyset, w_1, \cdots, w_{k-1}\}$, any function h, there is a PPT algorithm A^* such that the following value is negligible in t:*
     $$|\mathbf{Pr}[A(C_k, 1^t) = h(\Delta_k)] - \mathbf{Pr}[A^*(C_k^*, 1^t) = h(\Delta_k)]|.$$
     (*Note the requirement captures the following: everything about Δ_k that can be computed given C_k can also be computed given C_k^*.*)

On the security requirement. Recall that our goal is the following: in round j, $\mathcal{S}$ can learn nothing more than "a keyword is shared by the sent encrypted files." To this end, consider an ideal case: $\mathcal{U}$ records in advance a set of linked lists such that each file index is associated with a list of all the keywords of the corresponding file. In this case, $\mathcal{U}$ knows for sure which files contain the keyword in round j, namely w_j, and hence it is enough for $\mathcal{U}$ to send $\mathcal{S}$ an n-bit string Q_j such that for $i \in [n]$, $Q_j[i] = 1$ if and only if w_j is a keyword of m_i (and $\mathcal{S}$ has to send $\mathcal{E}_i(m_i)$ back). Note C_k^* exactly consists of such communications from $\mathcal{U}$ before round k. To be sure that the security of an implementation P of PPSED is not worse than that of the ideal case, we ask all the communications from $\mathcal{U}$ before round k in the execution of P, namely C_k, cannot leak more information than C_k^*. Specifically, we ask everything about Δ_k that can be computed given C_k can also be computed given C_k^*. Notice that this ideal case is not a practical solution itself to the PPSED problem, since it would require $\mathcal{U}$ store these linked lists, which would be $\Omega(n)$ bits in total. (In particular, these lists would generally require significantly more storage than a dictionary.) It would also require potentially sending n bits from $\mathcal{U}$ to $\mathcal{S}$ for every query.

3 Efficient Schemes

In this section, we consider the following two cases separately: (1) a dictionary can be stored on $\mathcal{U}$'s mobile device, and (2) a dictionary cannot be stored on $\mathcal{U}$'s mobile device (ostensibly due to lack of space). We study the first case both for its own merit and to lead us to the solution of the second case. In either case, our scheme consists of two phases. In the first phase, we assume $\mathcal{U}$ is at home and is going to submit his files to $\mathcal{S}$, and assume a keyword dictionary is always available to $\mathcal{U}$. However, we do not exclude the possibility that $\mathcal{S}$ has a dictionary that is totally the same (i.e. the dictionary may be publicly accessible). In the second phase, $\mathcal{U}$ becomes a mobile user, and wants to retrieve certain files by keyword searches via his mobile device. The main idea behind our schemes is the following: $\mathcal{U}$ uses pseudo-random bits to mask a keyword index for each file and sends it to $\mathcal{S}$ so that later $\mathcal{U}$ can use the short seeds to help $\mathcal{S}$ recover selective parts of the index, while keeping the remaining parts pseudo-random.

3.1 When a dictionary can be stored on $\mathcal{U}$'s mobile device

We formalize the keyword dictionary as 2^d index-word pairs (i, w_i), with $i \in [2^d], w_i \in \{0,1\}^*$ for some constant d. Next, given the security parameter t, for $K \in \{0,1\}^t$, let $P_K(x)$ be a family of pseudo-random permutations with domain $\{0,1\}^d$, let $F_K(x)$ be a family of pseudo-random functions mapping $\{0,1\}^d$ to $\{0,1\}^t$, and let $G_K(x)$ be a family of pseudo-random functions mapping $[n]$ to $\{0,1\}$. Here is our two-phase PPSED scheme.

$\boxed{\textit{Scheme1}}$

<u>Noninteractive Setup at Home</u>

- $\mathcal{U}$ chooses $s, r \in \{0,1\}^t$ uniformly at random and keeps them secret.
- Initially, for each file m_j, $1 \le j \le n$, $\mathcal{U}$ prepares a 2^d-bit index string I_j such that if m_j contains w_i, $\mathcal{U}$ sets $I_j[P_s(i)]$ to be 1, and otherwise $I_j[P_s(i)]$ is set to 0.
- Next, $\mathcal{U}$ computes $r_i = F_r(i)$ for $i \in [2^d]$, and for each file $m_{j,j\in[n]}$, computes a 2^d-bit *masked* index string M_j such that $M_j[i] = I_j[i] \oplus G_{r_i}(j)$.
- For $1 \le j \le n$, $\mathcal{U}$ submits $\mathcal{E}_j(m_j)$ to $\mathcal{S}$ along with the corresponding masked index string M_j.
- $\mathcal{U}$ copies the two secret keys s, r and the dictionary to his mobile device before leaving home.

<u>1-round Mobile Retrieval</u>

- In order to retrieve files with a keyword w_λ, $\mathcal{U}$ first retrieves the corresponding index λ from his dictionary, and then sends $p = P_s(\lambda)$ and $f = F_r(p)$ to $\mathcal{S}$.
- $\mathcal{S}$ then computes $I_j[p] = M_j[p] \oplus G_f(j)$ for $j \in [n]$. If $I_j[p] = 1$, $\mathcal{S}$ sends $\mathcal{E}_j(m_j)$ to $\mathcal{U}$.

Theorem 2. *Scheme1 is an implementation of* PPSED *where $\mathcal{S}$ sends $\sum_{i\in\mathcal{I}_j} |\mathcal{E}_i(m_i)|$ total bits, $\mathcal{U}$ stores $2t$ bits plus a dictionary of constant size, and $\mathcal{U}$ sends $(d+t)$ bits per keyword search.*

Proof. Because the correctness and the communication complexity of *Scheme1* can be easily verified, it suffices to prove $\mathcal{U}$'s security. W.l.o.g. we assume $\mathcal{U}$ does not make the same query twice, and hence the protocol consists of at most 2^d retrieval rounds.

In the following, by "the view of $\mathcal{S}$" we mean all the communications $\mathcal{S}$ receives from $\mathcal{U}$. Let ζ denote $\{\mathcal{E}_1(m_1), \mathcal{E}_2(m_2), \cdots, \mathcal{E}_n(m_n)\}$. Next, let

$$\begin{aligned}\mathcal{I}(a) &= \{I_1[a], I_2[a], \cdots, I_n[a]\},\\ \mathcal{M}(a) &= \{M_1[a], M_2[a], \cdots, M_n[a]\},\\ \mathcal{G}(a) &= \{G_a(1), G_a(2), \cdots, G_a(n)\},\end{aligned}$$

and let $\mathcal{M} = \{\mathcal{M}(1), \mathcal{M}(2), \cdots, \mathcal{M}(2^d)\}$. Moreover, let λ_v denote the dictionary index of the keyword in round v, and define $p_v = P_s(\lambda_v)$ and $f_v = F_r(p_v)$. In addition, let C_v denote the view of $\mathcal{S}$ before round v, so we have

$$C_1 = \{\zeta, \mathcal{M}\},\ C_2 = \{\zeta, \mathcal{M}, p_1, f_1\},\ C_3 = \{\zeta, \mathcal{M}, p_1, p_2, f_1, f_2\},\ \ldots\ .$$

Consider the ideal case which meets our security requirement perfectly: $\mathcal{U}$ records in advance a set of linked lists such that each file index is associated with a list of all the keywords of the corresponding file. In this case, the only message $\mathcal{U}$ needs to send in round v is the n-bit string Q_v such that for $j \in [n]$, $Q_v[j] = 1$ if and only if m_j contains the keyword in round v (and $\mathcal{S}$ has to send $\mathcal{E}_j(m_j)$ back). So if we let C_v^* denote the view of $\mathcal{S}$ before round v in the ideal case, we have

$$C_1^* = \{\zeta\},\ C_2^* = \{\zeta, Q_1\},\ C_3^* = \{\zeta, Q_1, Q_2\},\ \dots\ .$$

Observe that $Q_v = \mathcal{I}(p_v)$ for $v \in [2^d]$.

Our goal is to prove the following (for $k \in [2^d + 1]$): for any PPT algorithm A, any $\Delta_k = \{m_1, \cdots, m_n, w_0 \equiv \emptyset, w_1, \cdots, w_{k-1}\}$, any function h, there is a PPT algorithm A^* such that the following value is negligible in t:

$$\rho = |\mathbf{Pr}[A(C_k, 1^t) = h(\Delta_k)] - \mathbf{Pr}[A^*(C_k^*, 1^t) = h(\Delta_k)|.$$

Intuitively, suppose A^* on input C_k^* can generate a view C_k' that is indistinguishable from C_k. Then A^* can simulate running A with C_k' to give the desired result (that is, that ρ is negligible in t). We shall follow this intuition.

For $k = 1$, A^* just needs to choose $\mathcal{M}'$ from $\{0,1\}^{n2^d}$ uniformly at random, and feeds A with $\{\zeta, \mathcal{M}'\}$. We claim A^* is as desired as otherwise the pair (A, A^*) is a PPT distinguisher for pseudo-random bits and truly random bits. For $k > 1$, the strategy of A^* is as follows:

- A^* chooses $f_1', f_2', \cdots, f_{k-1}'$ uniformly at random from $\{0,1\}^t$, and chooses $s' = (p_1', p_2', \cdots, p_{k-1}')$ uniformly at random from $S = \{s \mid s \subset \{1, 2, \cdots, 2^d\}, |s| = k-1\}$.
- A^* computes $\mathcal{M}' = \{\mathcal{M}'(1), \mathcal{M}'(2), \cdots, \mathcal{M}'(2^d)\}$ in the following way:
  - For $i \in [2^d], i \neq p_1', p_2', \cdots, p_{k-1}'$, choose $\mathcal{M}'(i)$ uniformly at random from $\{0,1\}^n$.
  - For $i \in [k-1]$, set $\mathcal{M}'(p_i') = Q_i \oplus \mathcal{G}(f_i')$.
- A^* feeds A with $C_k' = \{\zeta, \mathcal{M}', p_1', p_2', \cdots, p_{k-1}', f_1', f_2', \cdots, f_{k-1}'\}$.

We explain why this strategy works as follows. First, recall $Q_v = \mathcal{I}(p_v)$ for $v \in [k-1]$, and consider the following imaginary case: for each $i \in [2^d], i \neq p_1, p_2, \cdots, p_{k-1}$, $\mathcal{U}$ does not generate $\mathcal{M}(i)$ according to *Scheme1*; instead, $\mathcal{U}$ chooses $\mathcal{M}(i)$ from $\{0,1\}^n$ uniformly at random. Clearly, in this case, the only difference between the generation of C_k' and the generation of C_k comes from the employment of truly randomness in place of pseudo-randomness. Specifically, C_k' is generated using truly random p_j' and f_j' for $j \in [k-1]$, yet C_k is generated using pseudo-random p_j and f_j for $j \in [k-1]$. So we claim ρ must be negligible in t in this case as otherwise the pair (A, A^*) can be used to invalidate either P_K or F_K.

Next, consider the real case (that $\mathcal{U}$ does follow every step of *Scheme1*). An observation is for each $i \in [2^d], i \neq p_1, p_2, \cdots, p_{k-1}$, $\mathcal{M}(i)$ remains pseudo-random before round k. However, since this is the only difference between the real case and the imaginary case, we claim ρ must be negligible in t in the real case as otherwise the pair (A, A^*) can be used to invalidate G_K. In consequence, we have proven the desired security guarantee. □

Analysis. We examine the practicability of the above scheme with realistic parameters. First, if we set $d = 18$, the storage overhead on server is 32 kilobytes per file. Note the latest *Merriam-Webster's Collegiate Dictionary* contains only 225,000 definitions [M03]. So even if $\mathcal{U}$ adds new words by himself, 2^{18} could be a reasonable upper-bound in practice for the number of all the distinct words in $\mathcal{U}$'s dictionary as well as in his documents. Second, notably only a few bits are sent from $\mathcal{U}$ per keyword search. If we set $t = 2030$, for example, only 256 bytes are required. Clearly, our scheme is independent of the encryption method chosen for the remote files, so it works for different file formats (including compressed files, multimedia files, etc.), as long as a keyword index on the corresponding content can be built. Moreover, only pseudo-random functions (and permutations) are used in the construction of our scheme. As mentioned earlier, these functions can be implemented efficiently by heuristic algorithms.

Although we assume the availability of a dictionary on $\mathcal{U}$'s mobile device, the assumption is not far-fetched as most of today's mobile devices are equipped with built-in electronic dictionaries (or can store one on a memory card). Actually, if we estimate the average length of a keyword by 2^3 ASCII characters, a dictionary only amounts to $(2^{18})(2^3)(8) = 2$ megabytes, which can be improved further using compression.

3.2 When a dictionary cannot be stored on $\mathcal{U}$'s mobile device

We now consider the same setting as the previous section, except that a dictionary cannot be stored on $\mathcal{U}$'s mobile device. Our new scheme is almost the same with *Scheme1*, with the pivotal difference being that $\mathcal{U}$ is asked to store an encrypted dictionary on $\mathcal{S}$.

Let w_{max} upper-bound the length of a word in $\mathcal{U}$'s local dictionary at home, let Φ be a family of pseudo-random permutations on $\{0,1\}^{w_{max}}$, and let $F^*_K(x)$ be a family of pseudo-random functions mapping $\{0,1\}^{d+w_{max}}$ to $\{0,1\}^t$. Here is our two-phased PPSED scheme.

Scheme2

Noninteractive setup at home

- $\mathcal{U}$ follows the first two steps of *Scheme1*, except he also chooses $\tau \in \{0,1\}^t$ uniformly at random and keeps it secret.
- Next, $\mathcal{U}$ sends to $\mathcal{S}$ the following in order: $\varphi_1 = \Phi_\tau(w_{i_1}), \varphi_2 = \Phi_\tau(w_{i_2}), \cdots, \varphi_{2^d} = \Phi_\tau(w_{i_{2^d}})$ such that $P_s(i_j) = j$ for $j \in [2^d]$. ($\mathcal{S}$ then records (j, φ_j) for $j \in [2^d]$, following the order.)
- Next, $\mathcal{U}$ computes $r_i = F^*_\tau(i, \varphi_i)$ for $i \in [2^d]$, and for each file $m_{j,j\in[n]}$, computes a 2^d-bit *masked* index string M_j such that $M_j[i] = I_j[i] \oplus G_{r_i}(j)$.
- $\mathcal{U}$ follows the last two steps of *Scheme1*, except he copies τ, instead of the dictionary, to his mobile device before leaving home.

2-round mobile retrieval

- In order to retrieve files with keyword w_λ, $\mathcal{U}$ sends $\varphi = \Phi_\tau(w_\lambda)$ to $\mathcal{S}$.
- Let (p, φ_p) be such that $\varphi_p = \varphi$. $\mathcal{S}$ sends p to $\mathcal{U}$, who then sends $f = F^*_\tau(p, \varphi)$ to $\mathcal{S}$.
- $\mathcal{S}$ then computes $I_j[p] = M_j[p] \oplus G_f(j)$ for $j \in [n]$. If $I_j[p] = 1$, $\mathcal{S}$ sends $\mathcal{E}_j(m_j)$ to $\mathcal{U}$.

Theorem 3. *Scheme2 is an implementation of* PPSED *where $\mathcal{S}$ sends $\sum_{i\in\mathcal{I}_j} |\mathcal{E}_i(m_i)| + d$ total bits, $\mathcal{U}$ stores $3t$ bits, and $\mathcal{U}$ sends $(w_{max} + t)$ bits per keyword search.*

Proof. We first prove $\mathcal{U}$'s security, employing some of the notation in the proof of Theorem 2. Let $\tilde{C}_k$ denote the view of $\mathcal{S}$ before round k in *Scheme2*. It suffices to prove the following: for all k, for any PPT algorithm $\tilde{A}$, any $\Delta_k = \{m_1, \cdots, m_n, w_0 \equiv \emptyset, w_1, \cdots, w_{k-1}\}$, and any function h, there is a PPT algorithm A such that the following value is negligible in t:

$$|\mathbf{Pr}[\tilde{A}(\tilde{C}_k, 1^t) = h(\Delta_k)] - \mathbf{Pr}[A(C_k, 1^t) = h(\Delta_k)|.$$

Recall C_k is the view of $\mathcal{S}$ before round k in *Scheme1*. In other words, we ask everything about Δ_k that can be computed given $\tilde{C}_k$ can also be computed given C_k. In other words, the information leakage of *Scheme2* is essentially no worse than that of *Scheme1*.

Since the retrieval phase is interactive, we know $\mathcal{U}$'s ongoing action depends on $\mathcal{S}$'s message, namely p. So we must consider the case that $\mathcal{S}$ might dishonestly send an arbitrary $p' \neq p$ to $\mathcal{U}$. However, let us start from the simplified case that $\mathcal{S}$ always sends the correct p to $\mathcal{U}$.

In the simplified case, we can assume w.l.o.g. that $\mathcal{U}$ always sends back p, along with f, to $\mathcal{S}$. Note this does not jeopardize $\mathcal{U}$'s security since $\mathcal{U}$ learns p from $\mathcal{S}$, while the difference between $\tilde{C}_k$ and C_k now comes from $\{\varphi_j\}_{j \in [2^d]} + \{\varphi = \varphi_p\}$.[3] An observation is A can simulate each φ_j by flipping coins and can simulate φ by setting φ to be the simulated φ_p, in that each φ_j represents t pseudo-random bits and p is known to A (as $p \in C_k$ is part of the input to A). So all A needs to do is to feed $\tilde{A}$ with C_k and the simulated results.

Next, let us consider the case that $\mathcal{S}$ might be dishonest. Note if $\mathcal{S}$ sends $p' \neq p$ to $\mathcal{U}$, then the returning message from $\mathcal{U}$, namely $f' = F_r^*(p', \varphi)$, cannot be used for decryption and represents nothing more than t pseudo-random bits. Hence we can assume w.l.o.g. that $\mathcal{S}$ always simulate f' by flipping t coins and discarding f' from his view ($\tilde{C}_k$) in this case. Accordingly, it is enough to prove that for all k, for any PPT algorithm $\tilde{A}$, any $\Delta_k = \{m_1, \cdots, m_n, w_0 \equiv \emptyset, w_1, \cdots, w_{k-1}\}$, any function h, there is a PPT algorithm A such that the following value is negligible in t:

$$|\mathbf{Pr}[\tilde{A}(\tilde{C}_k, 1^t) = h(\Delta_k)] - \mathbf{Pr}[A(c_k, 1^t) = h(\Delta_k)|,$$

where $c_k \subset C_k$ is the *reduced* C_k defined as follows: c_k is constructed by mimicking $\mathcal{S}$'s dishonest behavior to discard the corresponding f from C_k. Clearly, A just needs to do the same simulations as in the simplified case and feeds $\tilde{A}$ with c_k and the simulated results. Hence, we have finished the security proof by describing this PPT algorithm A.

There server $\mathcal{S}$ must send an additional d bits (namely p) beyond the files themselves. The on-mobile-device dictionary is replaced by a small storage overhead of t bits (namely the key to Φ_K). Moreover, we claim the correctness follows the fact that Φ_K is injective, and the user-side communication complexity can be easily verified. $\square$

Analysis. If we estimate the maximal length of a word by 2^4 ASCII characters, we have $w_{max} = (2^4)(8) = 128$. Hence the encrypted dictionary amounts to $(2^{18})(128) = 4$ megabytes per user. The server-side storage overhead is the same with *Scheme1*. On the other hand, the communication complexity changes only slightly: $\mathcal{S}$ needs to send additional d bits per keyword search, while $\mathcal{U}$ now needs to send $(w_{max} + t)$ bits per keyword search.

4 Secure Updating

In this section, we study the problem how to securely update the remote files. First, we have two observations: (1) to delete a remote file along with its encrypted index is straightforward in

[3]W.l.o.g. we can assume $F_K(j) \equiv F_K^*(j, \varphi_j)$ for all $j \in [2^d]$, i.e. $\{\varphi_j\}_{j \in [2^d]}$ is part of the description of F_K.

our schemes (we simply ignore it and its index henceforth), and (2) to update a remote file can be thought as deleting the old one and then adding a new file. Hence we can reduce the update problem to how to securely submit new files.

Basically $\mathcal{U}$ can follow all the steps in the first phase of either *Scheme1* or *Scheme2* to submit a new set of files, but some additional care is indispensable. First note if $\mathcal{U}$ treats the new files as a continuum of the old ones, or say, if $\mathcal{U}$ still uses the old pseudo-random seeds $\{r_i\}_{i\in[2^d]}$, then for any keyword that $\mathcal{U}$ has queried, $\mathcal{S}$ can learn (for free) whether the newly added files contain the (unknown) keyword or not as he already knows the corresponding pseudo-random seed. This says the newly submitted files suffer from a-prior information leakage before any query.

A solution is to choose independently a truly random seed r^θ to generate $\{r_i^\theta\}_{i\in[2^d]}$ for each file set ζ^θ, and to let $\mathcal{S}$ memorize the separating points amongst the asynchronous sets. When $\mathcal{U}$ makes a query, he should compute for each set ζ^θ a pseudo-random seed corresponding to the keyword index (using the truly random seed r^θ), and send all of them to $\mathcal{S}$, who then decodes the encrypted index accordingly. In this way, the aforementioned a-priori information leakage can be avoided. However, this approach suffers from an increasing number of truly random seeds that have to be stored on the mobile device: it works well only when the updating process is not so frequent.

Fortunately, we can apply another pseudo-random function to generate each r^θ, which is thus not truly random anymore. But similarly to our previous approaches, we know it is safe to feed pseudo-random functions with pseudo-random seeds, and therefore we just need one truly random seed for the new pseudo-random function (and for all the sets). In consequence, we claim our schemes are incremental in the following sense: $\mathcal{U}$ can submit new files which are totally secure against previous queries but still searchable against future queries. Moreover, they both have very simple key management.

5 Discussion

It is worth considering the extending the security requirement of PPSED to something stronger. For example, we could require indistinguishability, in the manner of PIR. Clearly, to work out an efficient solution with reasonable bounds on the number of bits transmitted and stored under such a strong security requirement is very difficult. If we give up on such bounds, and ask instead just that the communication from $\mathcal{S}$ to $\mathcal{U}$ should be strictly less than the sum of remote encrypted files (just as how PIR is defined), we can develop a possible solution using PIR as a subroutine. Here the trick is simple: follow the ideal case to record a set of linked lists, and then use PIR to retrieve the matched files. Other variations are possible (e.g. [CGN98]), but they do not seem as efficient or practicable as our schemes.

Dealing with queries containing Boolean operations on multiple keywords remains the most significant and challenging open problem.[4] Similarly, allowing general pattern matching, instead of keyword matching, remains open. Solving these open questions would greatly enhance the utility of these schemes.

Our schemes can also deal with occurrence queries in a less efficient way. An occurrence query is a query like "I want to retrieve all the files containing more than 10 occurrences of PRIVACY." One simple solution coupled with our schemes is to also record each occurrence of a word in the encrypted index. Hence if the word PRIVACY appears 12 times in a document, each appearance would be

[4] [G04] proposed a method to deal with such Boolean queries by letting $\mathcal{S}$ learn both which files contain x and which files contain y and send the intersection set back to $\mathcal{U}$. However, this approach cannot capture the intuition that $\mathcal{S}$ should only learn the intersection set instead of separate sets when a Boolean query $x \wedge y$ is made. In fact, the same method can also be applied to our schemes, yet the security of this method is unsatisfactory.

labelled separately as PRIVACY1, PRIVACY2, ..., and a query could be done on PRIVACY10. This approach can dramatically increase the storage overhead on the sever-side. We shall consider more efficient methods for this problem in the final version of this paper.

We believe these problems are of growing importance, as keyword searches on encrypted data might have a broad range of applications in distributed multi-user settings. For example, [BGHP02] studies the problem how to efficiently share encrypted data on P2P networks. In similar settings, keyword searches on encrypted data are indispensable.

References

[B70] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," in *Communications of the ACM, 13(7)*, pp. 422–426, 1970.

[BCK96] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *Proceedings of CRYPTO'96*, Lecture Notes in Computer Science 1109, pp. 1–15.

[BCOP03] D. Boneh, G. Crescenzo, R. Ostrovsky, and G. Persiano, "Searchable public key encryption," to appear in *Proceedings of Eurocrypt 2004*.

[BGHP02] K. Bennett, C. Grothoff, T. Horozov, and I. Patrascu, "Efficient sharing of encrypted data," in *Proceedings of ACISP 2002*, Lecture Notes in Computer Science 2384, pp. 107–120.

[CGN98] B. Chor, N. Gilboa, and M. Naor, "Private information retrieval by keywords," in Technical Report TR CS0917, Department of Computer Science, Technion, 1998.

[CMO00] G. Crescenzo, T. Malkin, and R. Ostrovsky, "Single database private information retrieval implies oblivious transfer," in *Proceedings of Eurocrypt 2000*, Lecture Notes in Computer Science 1807, pp. 122–138.

[G01] O. Goldreich, *Foundations of Cryptography: Basic Tools*, Cambridge University Press, 2001.

[G04] E.-J. Goh, "Secure indexes," in Cryptology ePrint Archive: Report 2003/216, February 25, 2004. (See http://eprint.iacr.org/2003/216/ for the latest version.)

[GO96] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," in *Journal of ACM 43(3)*, pp. 431–473, 1996.

[K88] J. Kilian, "Founding cryptography on oblivious transfer," in *Proceedings of ACM STOC'88*, pp. 20–31.

[KO97] E. Kushilevitz and R. Ostrovsky, "Replication is not needed: single database, computationally-private information retrieval," in *Proceedings of IEEE FOCS'97*, pp. 364–373.

[LR85] M. Luby and C. Rackoff, "How to construct pseudo-Random permutations from pseudo-random functions (abstract)," in *Proceedings of CRYPTO'85*, Lecture Notes in Computer Science 218, pp. 447.

[M03] F. Mish (editor in chief), *Merriam-Webster's Collegiate Dictionary, 11th edition*, Merriam-Webster, Inc., 2003.

[NR97] M. Naor and O. Reingold, "On the construction of pseudo-random permutations: Luby-Rackoff revisited (extended abstract)," in *Proceedings of ACM STOC'97*, pp. 189–199.

[SWP00] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceedings of IEEE Symposium on Security and Privacy 2000*, pp. 44–55.