Search Pattern Leakage in Searchable Encryption: Attacks and New Constructions

Chang Liu
School of Computer Science,
Beijing Institute of Technology
Beijing, 100081, China
changliu.bit@gmail.com

Liehuang Zhu
School of Computer Science,
Beijing Institute of Technology
Beijing, 100081, China
liehuangz@bit.edu.cn

Mingzhong Wang
School of Computer Science,
Beijing Institute of Technology
Beijing, 100081, China
wangmz@bit.edu.cn

Yu-an Tan
School of Computer Science,
Beijing Institute of Technology
Beijing, 100081, China
victortan@yeah.net

ABSTRACT

Searching on remote encrypted data (commonly known as *searchable encryption*) is becoming an important technique in secure data outsourcing, since it allows users to outsource encrypted data to the third party and maintains the keyword searching on the data at the same time.

It has been widely accepted in the literature that searchable encryption techniques should leak as little information as possible to the third party. An early classical method called oblivious RAM hides all information at the cost of poly-logarithmic computation and communication overheads, which turns out to be impractical in the real world applications (e.g., cloud computing). A number of efficient searchable encryption schemes have been proposed under weaker security guarantees afterwards, however, such schemes leak statistical information about the user's search pattern.

In this paper, we show that the search pattern leakage can result in non-trivial risks. As pioneer work, we present two concrete attack models exploiting user's search pattern and some auxiliary background knowledge aiming to disclose the underlying keywords of user's queries. To resist these attacks, we develop two new searchable encryption constructions that hide the search pattern. Our constructions are designed to be independent from the underlying searchable encryption scheme. Our experiments, which are based on the real world dataset, demonstrate the effectiveness and efficiency of proposed attack models and new constructions.

Keywords

search pattern, searchable encryption, index, fake query

1. INTRODUCTION

Cloud computing has been increasingly employed to outsource data by cloud tenants to diminish the high overhead of data storage and management. In many cases, cloud tenants need to encrypt data before outsourcing them to prevent cloud administrator from accessing sensitive information, such as government documents, hospital records and personal emails. However, data encryption invalidates many data query functions, which will lead to inefficient data utilization. For instance, the cloud service provider cannot execute keyword search query over encrypted data. The purpose of searchable encryption is allowing a user to outsource data to a third party in a secure manner, and then retrieve documents containing the queried keywords. Therefore, searchable encryption can play an important role in today's cloud computing scenario (see [12] for a discussion).

Index, an auxiliary structure that accelerates the search process, has been widely studied in information retrieval fields. In general, each entry of the index is formed as a <keyword, document identifiers> tuple, so that all the documents containing the queried keyword can be easily located by searching the indexed document identifiers. We describe the general scenario of searchable encryption as follows: a data owner (e.g., Alice) has a set of documents to outsource to a third party (e.g., Carol). Alice first builds an index of all the keywords contained in the documents, and then encrypts both the documents and the index. After that, she outsources the encrypted documents and index to Carol. An authorized data user (e.g., Bob) has a secret key, so that he is able to generate keyword search queries (a.k.a. trapdoors or tokens) by calling a trapdoor function. Once Carol receives a query from Bob, she can search in the index and return the (encrypted) search result[1] to Bob. Then Bob is able to decrypt the search results and retrieve needed documents.

It has been widely accepted in the literature that neither the outsourced data (i.e., documents and index) nor the search query should leak as little information as possible to the third party. We note that searchable encryption can be achieved through a classical method called oblivious

[1]The search result is a collection of document identifiers whose corresponding documents contain the queried keyword.

RAM (e.g., [15, 10]), which attains the optimal security (i.e., nothing leaked to the third party). However, this kind of approaches is unpractical due to the poly-logarithmic computation and communication overheads. Therefore a number of searchable symmetric encryption (SSE) schemes (e.g., [16, 9, 7, 8, 17, 14, 13]) have been proposed under weaker security guarantees to achieve efficiency. Nevertheless, as it is also noted in [13], a limitation of most known SSE schemes is the leakage of access and search patterns. In fact, this limitation also exists in searchable asymmetric encryption schemes (e.g., [6, 4]). Informally, the access pattern is the information about which documents contain the queried keyword (i.e., the search result) for each of the user queries. The search pattern is the information about whether any two queries are generated from the same keyword or not. Thus the occurrence frequency of each query can be deduced from the search pattern. In recent studies, only [11] have discussed a concrete attack exploiting the access pattern to disclose user's queried keywords. The potential risks of the search pattern leakage are rarely studied in the literature, which motivates us to investigate on this issue and develop searchable encryption constructions under more rigorous security requirements.

Intuitions. Let's begin with analyzing why search pattern is leaked in SSE schemes. We can note that the query algorithms of all SSE schemes in the literature are deterministic, which means the same keyword will always generate the same query. In this sense, an adversary can easily judge whether any two queries are generated from the same keyword or not, so as to obtain the user's search pattern. One may apply probabilistic query algorithms to solve this problem. However, simply making use of probabilistic query algorithms still cannot hide the search pattern, because the entry touched in each search process discloses the search pattern as well. In other words, for the same keyword, its entry in the index must be the same, so that the adversary can disclose user's search pattern just by observing the entries touched in the index during the search process. For this reason, the search pattern is also leaked in searchable asymmetric encryption (a.k.a. public key encryption with keyword search, PEKS), whose query algorithms are probabilistic (see e.g., [6, 4]). It is also worthwhile to point out that the access pattern may leak search pattern in the same way (i.e., the same search result might mean the same queried keyword). We refer the reader to [11] for the details on how to hide the access pattern. The work of this paper only considers the search pattern leakage caused by the queries and index, and we leave the comprehensive search pattern hiding for the future work. A delicate approach to hide the search pattern is launching several fake queries along with the real query, so that the adversary cannot identify the real query. In this way, the search pattern is hidden simultaneously.

Our contributions. We outline the contributions of this paper as the following:

1. As pioneer work, we address the search pattern leakage issue and demonstrate its potential risks in the practical applications by giving two concrete attack models. In particular, an adversary who has obtained user's search pattern can effectively attack the underlying keywords of the user query under the help of some public available knowledge.
2. We put forward universal constructions for hiding the search pattern. Our constructions are designed to be independent of the underlying searchable encryption scheme, so that most existing index-based searchable encryption schemes (e.g., [9, 7, 8, 17, 14, 13]) can be used in our constructions.
3. We test the performance of the proposed attack models and constructions based on the real world dataset. The experiment results indicate the effectiveness of our attack models, and the security and practicality of our constructions.

Organization of the paper. The remainder of the paper is organized as follows: Section 2 briefly surveys the motivations of hiding the search pattern. Section 3 introduces some preliminaries. We formalize two attack models in Section 4. In Section 5 we will describe our constructions. Section 6 shows experimental studies for evaluating the performance of the proposed attack models and constructions. We review related work in Section 7 and conclude the paper in Section 8.

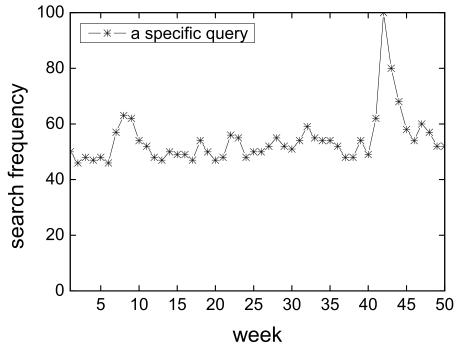
2. MOTIVATIONS

Throughout this paper, we treat the third party Carol as the adversary, which is consistent with most previous work. Carol behaves "honest-but-curious" in the searchable encryption protocol. On one hand, Carol follows the operations required by the protocol. On the other hand, Carol tries to deduce as much private information as possible by utilizing all kinds of attack methods. Carol has the ability of eavesdropping the network, so that she has access to all encrypted documents, index and queries. To make more effective attacks, Carol will draw support from some auxiliary knowledge, which can be legally obtained from other channels. Now we demonstrate that the search pattern leakage can lead to the keyword privacy leakage through several examples below.

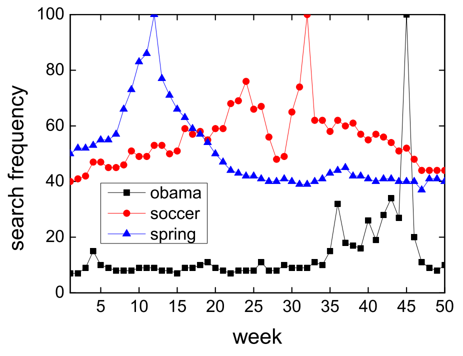
Once the search pattern is leaked to Carol, the occurrence frequency of each query is known to Carol. Intuitively, the most vulnerable keywords are of highest query frequencies, which we call hot keywords. For example, the keyword "Thanksgiving" is the hot keyword of Thanksgiving Day. Since Carol is able to identify the query with the highest occurrence frequency at Thanksgiving Day, she knows the underlying keyword of that query is "Thanksgiving".

To make a more general attack, Carol records the occurrence frequency of a specific query per day (week, month, etc.). As it is shown in Figure 1(a), Carol records the occurrence frequency of a specific query in each week of the year 2012. She also obtains auxiliary knowledge from a public web facility based on Google Search, called Google Trends [1] which shows how often a particular search-term is entered by users. For illustration, we show three sample keywords in Figure 1(b) and let the time interval be consistent with that recorded by Carol. What Carol needs to do is searching the best matched keyword by applying some methods of similarity measurement (e.g., Euclidean distance). The successful keyword attack above is based on the reasonable assumption that the users' search pattern are homologous in the Google Search and a general searchable encryption system.

In some cases, data users have specific background (such as IT, medicine, etc.), so the search habit of these users has



(a)



(b)

Figure 1: Search frequency[2]over time

discrepancy when compared with that of the general users. To attack the users with specific background, Carol needs to adjust the auxiliary knowledge accordingly. This is possible because Google Trends also offers statistics under variant categories, which can be treated as user backgrounds.

Currently, more and more results of analyses on users' search habits are open to the public, which indeed facilitates all variants of attacks on users' privacy. Consequently, the search pattern leakage will grievously threaten the keyword privacy. We formalize two keyword attack models in Section 4.

3. PRELIMINARIES

We first give a general definition of index-based searchable encryption, which covers both searchable symmetric encryption and searchable asymmetric encryption.

Definition 1 (Searchable Encryption). *An index-based Searchable Encryption (SE) scheme is a tuple of 6 algorithms* SE $=$ (KeyGen, BuildIndex, Encryption, Query, Search, Decryption)*:*

1. KeyGen(1^λ)*: The key generation algorithm takes a security parameter λ as input, it outputs K as a collection of secret keys.*
2. BuildIndex$(\mathcal{D})$*: The index building algorithm takes a document collection $\mathcal{D} = \{D_1, ..., D_n\}$ as input, it outputs an index $\mathcal{I}$.*
3. Encryption$(\mathcal{D}, \mathcal{I}, K)$*: The encryption algorithm takes a document collection $\mathcal{D}$, an index $\mathcal{I}$ and a secret key collection K as input, it outputs an encrypted document collection $\mathcal{C} = \{c_1, ..., c_n\}$ and a secure index $\mathcal{SI}$.*
4. Query(w, K)*: The query algorithm takes a keyword w as input, it outputs an encrypted query (a.k.a. trapdoor) q_w.*
5. Search$(q_w, \mathcal{SI})$*: The search algorithm takes a query q_w and a secure index $\mathcal{SI}$ as input, it outputs a collection of document identifiers whose corresponding data file containing the keyword w, which denoted as $\mathcal{R}(w) = \{\mathsf{id}(w, 1), ..., \mathsf{id}(w, p)\}$, where $\mathsf{id}(w, i)(1 \leq i \leq p)$ denotes the i-th identifier in $\mathcal{R}(w)$.*
6. Decryption(c_i, K)*: The decryption algorithm takes an encrypted data file $c_i \in \mathcal{C}$ and a secret collection K as input, it outputs D_i.*

Before we formally define the notion of search pattern, we first give the definition of *history*.

Definition 2 (History). *Let $\mathcal{D}$ be a document collection. A n-query history over $\mathcal{D}$ is a tuple $H = (\mathcal{D}, \boldsymbol{w})$ where $\boldsymbol{w} = \{w_1, w_2, ..., w_n\}$ is a vector of underlying keywords of the n queries.*

A *history* represents the interactions between the users and the third party, which contain a document collection and a list of keywords that the users have searched.

Definition 3 (Search Pattern). *The search pattern over a n-query history $H = (\mathcal{D}, \boldsymbol{w})$ is a $n \times n$ symmetric binary matrix τ_H such that for $1 \leq i, j \leq n$, $\tau_H[i][j] = 1$ if $w_i = w_j$, and 0 otherwise.*

Informally, a secure searchable encryption scheme should not leak any information of the history to the adversary. In existing security definitions of searchable encryption [7, 9, 8], the search pattern is assumed to leaked to the adversary. Thus their schemes might be vulnerable in the practical applications where the adversaries have knowledge on users' search habits.

4. TWO ATTACK MODELS

Now we formalize the keyword attack based on the search pattern leakage. Let q be a specific query that Carol wants to attack, Carol records the occurrence frequency of q in each period of time (e.g., day, week, month, etc.), which we denote as f_q^i for $1 \leq i \leq p$ where p is the total number of record items (e.g., $p = 50$ in Figure 1(a)). Thus Carol gets a frequency vector of q denoted as $V_q = (f_q^1, f_q^2, ..., f_q^p)$. Let D denote a dictionary of keywords and m denote the size of D. In our attack models, we assume Carol has auxiliary

[2]The search frequencies offered by Google Trends are normalized and displayed on a scale from 0 to 100, rather the absolute frequency numbers. We refer the reader to the site help of [1] for more details on how to normalize and scale data. In this paper, when we refer search frequencies (or occurrence frequencies) which are used for matching with the data in Google Trends, we assume these frequency numbers have been normalized and scaled properly.

knowledge about users' search habits (e.g., Google Trends [1]). Let $V_{w_i} = (f_{w_i}^1, f_{w_i}^2, ..., f_{w_i}^p)$ denote the auxiliary frequency vector of the keyword $w_i \in D(i \in \{1, ..., m\})$, where the time interval is consistent with V_q. Let $V = \{V_{w_1}, V_{w_2}, ..., V_{w_m}\}$ denote the set of all auxiliary frequency vectors. Let $Dist(V_q, V_{w_i})$ be a function measuring the similarity of V_q and V_{w_i} (e.g., Euclidean Distance, Cosine Distance and etc.).

Coral tries to identify the underlying keyword of q, which can be easily achieved by calling the following attack algorithm. We call this attack model *The General Attack* and denote it as $\mathsf{ATK}^{General}$:

- $\mathsf{ATK}^{General}(V_q, V)$:

```
    1. set $i^* = \arg\min_{i \in \{1,...,m\}} Dist(V_q, V_{w_i})$.
    2. return $w_{i^*}$.
```

We have mentioned that users sometimes have specific background, but Carol cannot discover the exact background directly. To deal with this situation, we present *The Adaptive Attack* (denoted as $\mathsf{ATK}^{Adaptive}$) which dynamically adjust the auxiliary knowledge based on the previous attacks. For example, if Carol has obtained five keywords "HIV", "leukocyte", "winter", "virus" and "glucose", she will guess the users are medical related and update the auxiliary frequency vector to an associated version. Here, we assume that the auxiliary knowledge offers statistics of users' search histories under different categories which refer to different user backgrounds. In addition, Carol needs to label all the keywords with these categories.

Let $C = \{c_1, c_2, ..., c_r\}$ be the set of all possible categories offered by the auxiliary knowledge. Let $V_{w_i,c_j} = (f_{w_i,c_j}^1, f_{w_i,c_j}^2, ..., f_{w_i,c_j}^p)(1 \le i \le m, 1 \le j \le r)$ be the auxiliary frequency vector of keyword w_i under the category c_j. Let $V_{c_j} = \{V_{w_1,c_j}, V_{w_2,c_j}, ..., V_{w_m,c_j}\}$ denote the set of all auxiliary frequency vectors under the category c_j. We set a weight value on each of the category (denoted as $v_j (1 \le j \le r)$), which are equal to the proportion of resulted keywords labeled with c_j[3] in the previous rounds of attacks. *The Adaptive Attack* works as follows:

- $\mathsf{ATK}^{Adaptive}(\{V_{q_i}\}_{i \in \mathbb{Z}}, \{V_{c_j}\}_{j \in \{1,...,r\}})$:

```
    1. for $1 \le j \le r$, set $v_j = 0$.
    2. randomly choose $V_{current}$ from $\{V_{c_j}\}_{j \in \{1,...,r\}}$.
    3. set $\mathsf{ctr} = 1$.
    4. let $w_{i^*} = \mathsf{ATK}^{General}(V_{q_{\mathsf{ctr}}}, V_{current})$.
    5. let $c_{j^*}$ be the category labeled on $w_{i^*}$.
    6. for $1 \le j \le r (j \ne j^*)$, let $v_j = \frac{v_j \cdot (\mathsf{ctr}-1)}{\mathsf{ctr}}$.
    7. let $v_{j^*} = \frac{v_{j^*} \cdot (\mathsf{ctr}-1)+1}{\mathsf{ctr}}$.
    8. set $j_{max} = \arg\max_{j \in \{1,...,r\}} v_j$.
    9. let $V_{current} = V_{c_{j_{max}}}$.
   10. let $\mathsf{ctr} = \mathsf{ctr} + 1$
   11. goto 4.
```

[3]It is not hard to verify that the sum of all the weight values is equal to 1.

The Random-Selecting-Based Construction

```
1. $\mathsf{KeyGen}(1^\lambda)$: output $K \leftarrow \mathsf{SE.KenGen}(1^\lambda)$
2. $\mathsf{BuildIndex}(\mathcal{D})$: output $\mathcal{I} \leftarrow \mathsf{SE.BuildIndex}(\mathcal{D})$
3. $\mathsf{Encryption}(\mathcal{D}, \mathcal{I}, K)$:
   output $(\mathcal{C}, \mathcal{SI}) \leftarrow \mathsf{SE.Encryption}(\mathcal{D}, \mathcal{I}, K)$
4. $\mathsf{Query}(k, w, K)$:
   (a) randomly choose $k-1$ keywords:
       $w_{i_0}, ..., w_{i_{k-2}} \in \mathcal{W} \backslash w$
   (b) randomly choose $b \in \{0, ..., k-1\}$
   (c) for $0 \le j \le b-1$:
       let $sq_j \leftarrow \mathsf{SE.Query}(w_{i_j}, K)$
   (d) let $sq_b \leftarrow \mathsf{SE.Query}(w, K)$
   (e) for $b+1 \le j \le k-1$:
       let $sq_j \leftarrow \mathsf{SE.Query}(w_{i_{j-1}}, K)$
   (f) output $(b, \mathcal{Q} = \{sq_0, ..., sq_{k-1}\})$
5. $\mathsf{Search}(\mathcal{Q}, \mathcal{SI})$:
   (a) for each $sq_i$ in $\mathcal{Q}$:
       let $\mathcal{R}_i \leftarrow \mathsf{SE.Search}(sq_i, \mathcal{SI})$
   (b) output $\mathcal{R} = \{\mathcal{R}_0, ..., \mathcal{R}_{k-1}\}$
6. $\mathsf{Extract}(\mathcal{R}, b)$: output $\mathcal{R}_b$
7. $\mathsf{Decryption}(c_i, K)$:
   output $D_i \leftarrow \mathsf{SE.Decryption}(c_i, K')$
```

Figure 2: The random-selecting-based construction (RSBC)

5. OUR CONSTRUCTIONS

In this section, we construct two searchable encryption constructions based on a index-based searchable encryption scheme defined in Section 3.

5.1 The random-selecting-based construction (RSBC)

We first introduce a straightforward construction, which we call the random-selecting-based construction (RSBC). Before describing RSBC in detail, we introduce some additional notations. Let $\mathcal{W}$ be the list of all distinct keywords contained in document collection $\mathcal{D}$ in alphabetical order and $|\mathcal{W}|$ be its size. Assume w_i denotes the i-th keyword in $\mathcal{W}$.

In RSBC, the query generated by Bob is a collection of k sub-queries, which includes one sub-query of the real keyword and $k-1$ sub-queries of randomly selected keywords. We call k the confusion parameter. To prevent Carol from identifying the sub-query of the real keyword, Bob needs to place this sub-query at a random position in the query structure. When Carol received a query (i.e., a collection of k sub-queries) from Bob, for each sub-query, she calls the search algorithm to get the sub-result (assumed to be encrypted). Then she gets the final search result in the form of k sub-results with the consistent order of the sub-queries, and sends it to Bob. Since Bob knows the correct position of the real sub-query, he can extract the sub-result of the real sub-query and deletes other sub-results. Bob then uses his secret key to decrypt the sub-result and finishes the query process.

Let SE be a specific searchable encryption scheme. We use the Extract algorithm to represent the process of extracting the real sub-result from the search result structure. The details of RSBC are shown in Figure 2.

Analysis. The main idea of RSBC is blending the sub-query of the real keyword with several sub-queries of fake keywords which are randomly selected in $\mathcal{W}$. In this case, Carol cannot identify the real sub-query of each Bob's query, thus cannot obtain Bob's search pattern. However, we observe that RSBC is not robust in some cases. It is possible that some particular keywords are queried much more times than other keywords. In this case, Carol is able to figure out the real sub-queries of these keywords by performing set-intersection operation. We show this using the following example: assume Bob has queried 5 keywords $(w_1, w_2, w_1, w_1, w_1)$ and the confusion parameter $k = 3$, the corresponding queries received by Carol are, say $\mathcal{Q}_1 = (sq(w_7), sq(w_1), sq(w_{38}))$, $\mathcal{Q}_2 = (sq(w_{34}), sq(w_{91}), sq(w_2))$, $\mathcal{Q}_3 = (sq(w_1), sq(w_{51}), sq(w_{67}))$, $\mathcal{Q}_4 = (sq(w_8), sq(w_{77}), sq(w_1))$, $\mathcal{Q}_5 = (sq(w_1), sq(w_{12}), sq(w_{83}))$. Through set-intersection operation, Carol can easily observe that $sq(w_1)$ appears four times among the five queries, so she believes that the real sub-query of the 1-st, 3-rd, 4-th and 5-th query is $sq(w_1)$. Thus the view of Carol can be presented as $View = \{\mathcal{Q}_1 = (sq(w_1)), \mathcal{Q}_2 = (sq(w_{34}), sq(w_{91}), sq(w_2)), \mathcal{Q}_3 = (sq(w_1)), \mathcal{Q}_4 = (sq(w_1)), \mathcal{Q}_5 = (sq(w_1))\}$, which leaks Bob's search pattern to a certain degree. One may think that $sq(w_1)$ can also appear in a query when it is selected as a fake sub-query. However, such a case occurs with very low probability if (1) the keyword set size is large, or (2) k value is small.

5.2 The random-dividing-based construction (RDBC)

To strengthen security, we propose an improved construction which we call the random-dividing-based construction (RDBC). In RDBC, we add an additional algorithm named Dividing, which divides $\mathcal{W}$ into $|\mathcal{W}|/k$ subsets $\{S_1, ..., S_{|\mathcal{W}|/k}\}$, where k is the confusion parameter. Here, we assume $|\mathcal{W}|$ is an integral multiple of k. Once a keyword $w \in \mathcal{W}$ is queried, each keyword that appears in the same subset with w will also be queried, so that the final query also contains one real sub-query and $k-1$ fake sub-queries.

In addition, We make use of a pseudo-random permutation π with the following parameters:

$$\pi : \{0,1\}^{\lambda} \times \mathbb{Z}_{|\mathcal{W}|} \to \mathbb{Z}_{|\mathcal{W}|}$$

where λ is a security parameter denotes the length of secret key. To randomly divide $\mathcal{W}$ into $|\mathcal{W}|/k$ subsets, we first use π permuting $\mathcal{W}$ and then divide the permuted $\mathcal{W}$. Here, only authorized users have secret key to calculate the permutation.

The details of RDBC are shown in Figure 3. Unlike RSBC, which randomly selects fake sub-queries from the whole $\mathcal{W}$, RDBC divides $\mathcal{W}$ into a number of subsets so that the generated query will always be the same when the queried keyword belongs to the same subset. This property avoid Carol's attack by performing set-intersection operation. To testify this, let's suppose $k = 3$, and suppose w_1 and w_2 belongs to the subset (w_{33}, w_1, w_{74}) and

The Random-Dividing-Based Construction

```
1. KeyGen($1^{\lambda}, 1^{\lambda'}$):
    (a) pick a random $s \in \{0,1\}^{\lambda}$
    (b) let $K' \leftarrow$ SE.KenGen($1^{\lambda'}$)
    (c) output $K = \{s, K'\}$
2. BuildIndex($\mathcal{D}$): output $\mathcal{I} \leftarrow$ SE.BuildIndex($\mathcal{D}$)
3. Encryption($\mathcal{D}, \mathcal{I}, K$):
   output $(\mathcal{C}, \mathcal{SI}) \leftarrow$ SE.Encryption($\mathcal{D}, \mathcal{I}, K'$)
4. Dividing($k, \mathcal{W}, K$):
    (a) calculate a permutation $P$ on $\{0, ..., |\mathcal{W}|-1\}$
        as follows:
        let $P[i]$ be the $i$-th element of $P$, for $0 \le i \le$
        $|\mathcal{W}|-1$:
        set $P[i] = \pi_s(i)$
    (b) divide $\mathcal{W}$ into $|\mathcal{W}|/k$ subsets as follows:
        for $0 \le i \le |\mathcal{W}|/k - 1$:
        set $S_i = \{w_{P[i*k]}, ..., w_{P[i*k+k-1]}\}$
    (c) output $\{S_i\}_{i \in \{0,...,|\mathcal{W}|/k-1\}}$
5. Query($k, w_i, K$):
    (a) set $x = \pi_s(i)$
    (b) calculate $(a, b)$ satisfy $x = ak + b$, where $0 \le$
        $b \le k-1$
    (c) let $S_a[j]$ be the $j$-th element of $S_a$, calculate
        sub-queries as follows:
        for $0 \le j \le k-1$:
        let $sq_j \leftarrow$ SE.Query($S_a[j], K'$)
    (d) output $(b, \mathcal{Q} = \{sq_0, ..., sq_{k-1}\})$
6. Search($\mathcal{Q}, \mathcal{SI}$):
    (a) for each $sq_i$ in $\mathcal{Q}$:
        let $\mathcal{R}_i \leftarrow$ SE.Search($sq_i, \mathcal{SI}$)
    (b) output $\mathcal{R} = \{\mathcal{R}_0, ..., \mathcal{R}_{k-1}\}$
7. Extract($\mathcal{R}, b$): output $\mathcal{R}_b$
8. Decryption($c_i, K$):
   output $D_i \leftarrow$ SE.Decryption($c_i, K'$)
```

Figure 3: The random-dividing-based construction (RDBC)

(w_{85}, w_{41}, w_2) respectively. Let's consider the following view of Carol: $View = \{\mathcal{Q}_1 = (sq(w_{33}), sq(w_1), sq(w_{74})), \mathcal{Q}_2 = (sq(w_{85}), sq(w_{41}), sq(w_2)), \mathcal{Q}_3 = (sq(w_{33}), sq(w_1), sq(w_{74})), \mathcal{Q}_4 = (sq(w_{33}), sq(w_1), sq(w_{74})), \mathcal{Q}_5 = (sq(w_{33}), sq(w_1), sq(w_{74}))\}$. Although $\mathcal{Q}_1$, $\mathcal{Q}_3$, $\mathcal{Q}_4$ and $\mathcal{Q}_5$ are identical, it is not rational for Carol to guess the real sub-query of the 1-st, 3-rd, 4-th and 5-th query are (1) $sq(w_1)$, because $sq(w_{33})$ and $sq(w_{74})$ also satisfy this situation, or (2) identical, because different queried keywords will generate the same query if these keywords are from the same subset.

It is easy to find that $k = 1$ implies an unchanged searchable encryption scheme, and $k = \mathcal{W}$ means each query contains sub-queries of all the keywords. The larger k is, the stronger security attained, but the higher overhead required. Therefore, it is significantly important to choose k properly according to the practical applications.

6. EXPERIMENTS

In this section, we test the attack models and our constructions presented in this paper. Our test programs were implemented by Python-2.7.3. All experiments were performed on a PC with Intel Core i5-2400 CPU 3.10GHz and 4.00GB RAM. The underlying operation system was Windows 7 Professional. Each data point presented in the experiment results was the mean of 10 executions.

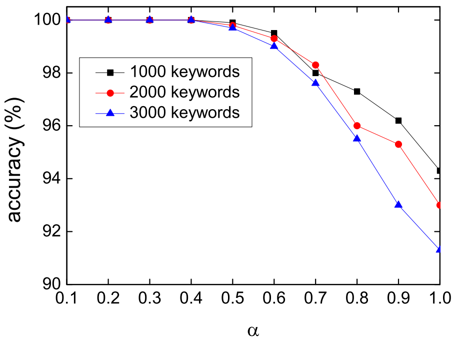
In our experiments, all keywords were selected from Enron email dataset[3], which is a real world dataset containing a total of about 500000 messages of about 150 users. We selected a subset of the emails as our corpus. It contains 96107 messages from the "Sent Mail" directories. The total number of distinct words in the corpus is 122426. We ranked these words with occurrence frequency and chose the top 3000 words as our keyword set. Here, all English stopwords[2] such as "a", "the" and "about" had been filtered away.

6.1 Performance of our attack models

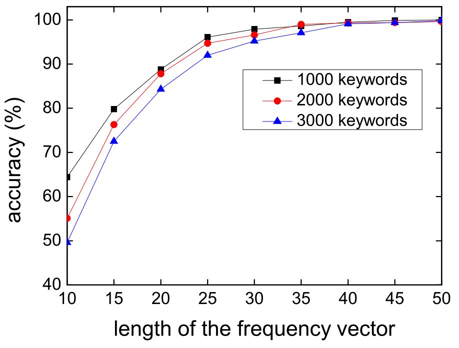
To demonstrate the feasibility and effectiveness of our attack models, we tested their attack accuracy[4] under different factor settings. We assumed that Carol had recorded the occurrence frequency of each query launched by users for consecutive weeks in the year 2011, thus she had obtained the frequency vector of each query. The length of the frequency vector was equal to the number of recorded weeks. Carol's auxiliary knowledge on statistics of users' search histories were extracted from Google Trends [1], which contains the frequency vectors of all the keywords in the keyword set. It is obvious that the attack accuracy will be 1 if the frequency vectors of users' queries perfectly match the auxiliary knowledge. However, in a real application, the scale of users may be so smaller than Google's that the real users' queries will be inevitably diverse from the auxiliary knowledge. Considering the significant differences between particular users' query behaviors noted by [11], and no real world query set on the Enron dataset has been published to the best of our observation, we simulate user query of a particular keyword through adding Gaussian noise $\mathcal{N}(0, \alpha \cdot \sigma^2)$ to the frequency vector of this keyword in the auxiliary knowledge. Here, σ^2 is the variance of the frequency vector and α[5] is a constant representing the noise level.

Figure 4(a) shows the attack accuracy of *The General Attack* under different keyword set sizes and α values. Here, the length of the frequency vector was fixed at 52 (i.e., 52 weeks in the year 2011), and the keyword set size was chosen to be 1000, 2000 and 3000 respectively. We can see that the attack accuracy is almost 100% when $\alpha \leq 0.5$ and starts to decrease when $\alpha > 0.5$. The experiment results showed that the proposed attack model was quite accurate if the data users' searches (in the experiment) were well consisted with Google's statistic. The attack was also successful when the data users' searches had large deviation to Google's statistic

[4]The attack accuracy represents the percentage of keywords that are successfully attacked by the adversary.
[5]The larger α value implies the larger noise energy.



(a)



(b)

Figure 4: Attack accuracy of *The General Attack* for different choices of keyword set size.

since the accuracy was approximately 92% even when $\alpha = 1$. With the increase of keyword set size, the attack accuracy slightly declined. This is because the larger size of keyword set inevitably incurs the higher probability of mismatch.

Figure 4(b) shows the attack accuracy of *The General Attack* under different keyword set sizes and lengths of the frequency vector. In this experiment, the α value was set at 0.5 and the keyword set size was chosen to be 1000, 2000 and 3000 respectively. We can see that the attack accuracy is growing with the increase of length of the frequency vector and reaches 90% when the vector length comes to 25. The attack accuracy is higher than 50% even when the vector length is as short as 10. Same as previous experiment, the attack accuracy slightly declines when keyword set size increases. As we have mentioned in Section 2, the period of records in the frequency vector in not limited as a week. Other kinds of period such as a day, a month and a year are also feasible.

To test the attack accuracy of *The Adaptive Attack*, we labeled 1000 keywords with 6 categories, which were "science", "health", "games", "sports", "food" and "non-classified". We extracted the associated frequency vectors under each of the categories from Google Trends, which formed Carol's auxiliary knowledge. In this experiment, data users were assumed to be of a specific category. To simulate the users' searches, we assumed that users of a specific category searched the keywords of this very category with higher probability than

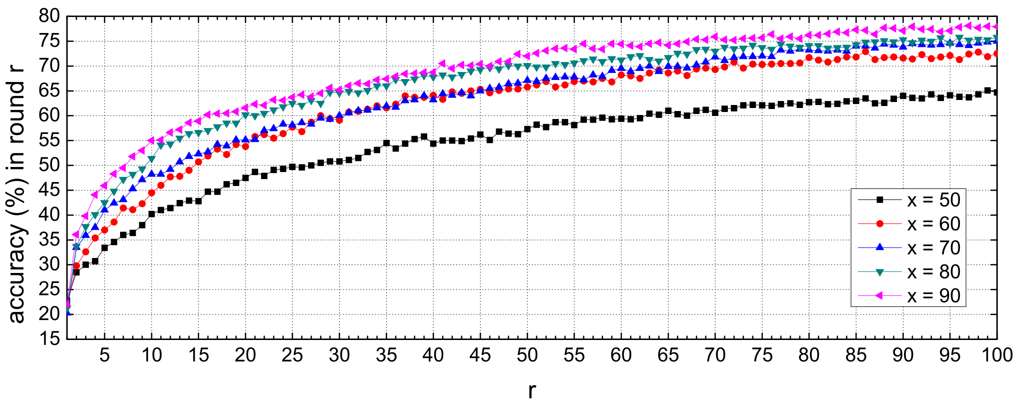


Figure 5: Attack accuracy of *The Adaptive Attack* for different choices of x.

other keywords. Thus we generated a sequence of 100 keywords, where x of them were labeled with the same category as the users. As it is shown in Figure 5, x is set to be 50, 60, 70, 80 and 90 respectively. The larger x implies the higher speciality of the user searches. According to *The Adaptive Attack* we have described in Section 4, during each round of the attack process Carol adjusts the utilized frequency vectors based on the previous rounds. In the experiment we fixed the noise parameter α to be 0.5 and the length of frequency vector to be 52. From Figure 5 we can see that the attack accuracy in the first round is only about 20%, that is because Carol had no idea on the users' category in the first round, thus the version of the auxiliary knowledge she used was very likely to mismatch the users' category. With the increase of rounds, Carol has significant probability to figure out the users' exact category, so that the attack accuracy in the 100-th round comes to about 65% when $x = 50$ and near 80% when $x = 90$.

The experiment results above suggested significant distinctions exist among the frequency vectors of each keyword, thus again we emphasize that any searchable encryption scheme leaking users' search pattern is vulnerable.

In Figure 6 we evaluate the execution time for searching the best match among the auxiliary frequency vectors under different lengths of the frequency vector. The keyword set size was set to be 1000, 2000 and 3000 respectively. We can see though the time costs were linear with the length of the frequency vector and the number of keywords, the time costs are in millisecond level, which means our attack models are quite efficient.

6.2 Performance of our constructions

Now we evaluate the performance of our constructions. In our constructions, a user query is a set of $k-1$ fake sub-queries and one real sub-query. To apply the attack models described in Section 4, Carol has to first guess the real sub-query in each user query. We describe Carol's operation as follows:

Guessing the real sub-query in RSBC. For each query $\mathcal{Q}_i = \{sq_{i_1}, sq_{i_2}, ..., sq_{i_k}\}$ (here, k is the confusion parameter), Carol calculates set-intersection with every other query. In this way, she can figure out the sub-query which occurs the most times in the results of the set-intersection calculations. She regards this sub-query as the "real" sub-query of $\mathcal{Q}_i$.

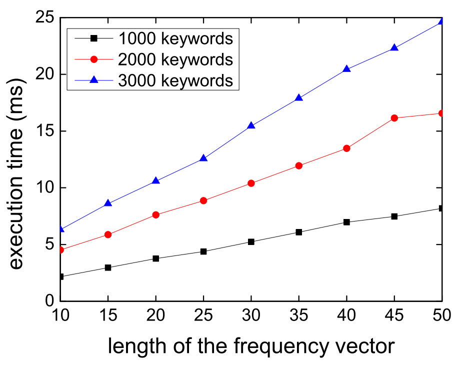


Figure 6: Execution time for search the best match.

Guessing the real sub-query in RDBC. We have analyzed in Section 5 that RDBC invalidates the set-intersection operation. In this case, for each query $\mathcal{Q}_i = \{sq_{i_1}, sq_{i_2}, ..., sq_{i_k}\}$, Carol randomly choose a sub-query as the "real" sub-query of $\mathcal{Q}_i$.

After Carol has obtained the "real" sub-query of all users' queries, she can calculate the frequency vector of each query and apply our attack models.

We evaluated the average accuracy of Carol hitting the real sub-query in Figure 7. In this experiment, the keyword set size, the length of the frequency vector and the noise parameter α were 3000, 52 and 0.5 respectively. Users' query sequence was generated according to the statistics of Google Trends. From Figure 7 we can see that the guessing accuracy in RDBC approximately equals to $1/k$ since Carol just randomly selects a sub-query from the total k sub-queries as the real sub-query. Carol has small advantage in guess-

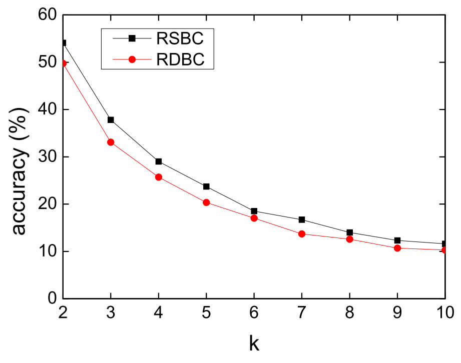


Figure 7: Under different choices of k, the accuracy of guessing the real sub-query.

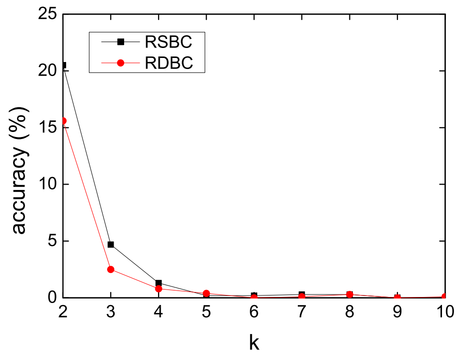
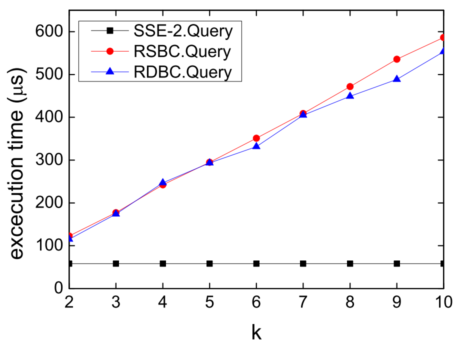
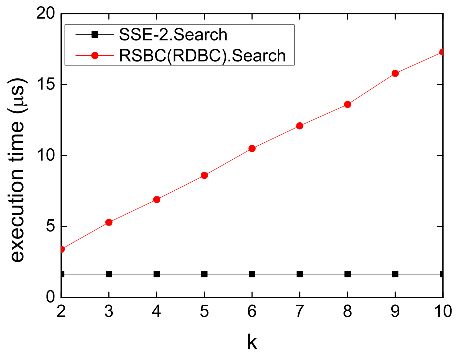


Figure 8: Under different choices of k, the attack accuracy of *The General Attack*

.



(a)



(b)

Figure 9: Average query and search execution time in SSE-2 and our constructions.

ing the real sub-query in RSBC, which is induced by the set-intersection operations.

Based on the results in Figure 7, we further evaluated how *The General Attack* performs in our constructions. Also, the keyword set size, the length of the frequency vector and the noise parameter α were set to be 3000, 52 and 0.5 respectively. From Figure 8 we can see that the attack accuracy is low as about 20% in RSBC and about 15% in RDBC even when $k = 2$. With a increase in k value, the attack accuracy decreases rapidly. For both constructions, the attack accuracy is near 0 when $k \geq 5$. Therefore, we suggest to choose k not less than 5 in the practical applications.

We also evaluated the computation overhead of our constructions. In our experiments, we chose SSE-2 in [8] as the underlying searchable encryption scheme in our constructions. Figure 9 reports the execution times for the Query and Search algorithms under different choices of k value. From the results in Figure 9(a) we can see that the execution time for the Query algorithms of our constructions are linear to the k value. In addition, they are about k times the execution time in SSE-2. Nevertheless, the Query algorithms of our constructions are quite time saving, since the execution time is less than 1ms even when k is chosen to be 10. In Figure 9(b), similarly, the execution times for the Search algorithms of our constructions are linear to the k value, and they are about k times the execution time in SSE-2. The Search algorithms of both constructions are exactly the same, so we didn't test them separately. In our implementations, indexes are built using hash table, which achieves $O(1)$ look-up time. Therefore the results of the search time in Figure 9(b) are very small.

Besides the Query and the Search algorithms, the Dividing algorithm (in RDBC) and the Extract algorithms (in both RSBC and RDBC) also import computation overhead. However, the Dividing algorithm only needs to be executed once at the setup phase for each user, and the execution time of the Extract algorithms can be ignored. Therefore, the additional computation overheads induced by our constructions are very small.

7. RELATED WORK

The problems of searching on remote encrypted data have been widely studied in the literature, most of which focus on enhancing privacy guarantees and optimizing efficiency. The classical method proposed by Goldreich and Ostrovsky [15, 10], which called oblivious RAM, can resolve the problem without leaking any information to the third party. However, standing in the perspective of practical applications, such schemes are unacceptable due to their poly-logarithmic computation and communication overheads. A number of searchable encryption schemes (e.g., [16, 9, 6, 7, 8, 17, 14, 13]) have been proposed under weaker security guarantees to achieve efficiency. More concretely, Song et al. [16] gave a practical solution attaining search time that is linear to the data size. This construction is not secure against statistical analysis, since the adversary can obtain the distribution information of the underlying plaintext through statistic approaches. To formalize security, Goh [9] formulated a security model as semantic security against adaptive chosen keyword attack (IND-CKA) and a slightly stronger IND2-CKA. He also developed an IND-CKA secure index called Z-INX which utilizes Bloom filter [5] to build an index for each data file. The overhead of Z-INX for testing whether a keyword belongs to a data file is $O(1)$, thus searching on the whole file collection needs $O(n)$ time. The security definition in [7] is similar with IND2-CKA except for the requirement that trapdoors should not leak any information of the queried keywords. As further related work, Curtmola et al. [8] put forward stronger security definitions. Their security definition requires any function about the documents and the keywords that can be computed from the encrypted documents, the index and the trapdoors can be computed from the length of the documents, the identifiers of the documents, the access pattern and the search pattern. In other words, nothing is leaked more than the length of the documents, the identifiers of the documents, the access pattern and the search pattern. In addition, both non-adaptive and adaptive adversaries are considered in their work.

The approaches mentioned above are in the scope of searching on symmetric key encrypted data, which thus called searchable symmetric encryption (SSE). For application sake, another research field of searchable encryption focuses on the public key setting. As pioneer work, Bonel et al. [6] proposed a searchable encryption scheme called PEKS (i.e., Public-key Encryption with Keyword Search), where trapdoor function is probabilistic. This property seems to contribute to hiding the search pattern. Unfortunately, except for trapdoors, index and search outcomes also leak the search pattern, which we have discussed in Section 1.

With the exception of oblivious RAMs, all proposed searchable encryption schemes leak the search pattern. The trend of distributing searchable encryption schemes into cloud servers [12, 17, 14] highlights the potential risks of search pattern leakage (as well as access pattern leakage). That is because large amounts of data centralizing into the cloud servers affiliates effective statistical attacks. Moreover, such attacks can be launched well under the massive computing power of cloud servers. We are aware of a recent work of access pattern disclosure on searchable encryption proposed by Islam et al. [11], which is close to our work. In [11], the authors treated keyword distribution as a background knowledge of the adversary and formulated a concrete attack on the access pattern leakage. They also presented a mitigation approach to hide the access pattern. Their main idea is importing some false positives to make search outcomes turn into identical to a certain degree. Therefore, both the issue we focus and the principle of our approaches are quite different from theirs.

8. CONCLUSION

In this paper, we review searchable encryption schemes in the literature and point out the search pattern leakage issue. By giving two concrete attack models, we demonstrate that the search pattern can be utilized to attack the underlying queried keywords. Motivated by this, we present two constructions based on the idea of faking query. The security guarantees of both constructions are associated with the confusion parameter, which needs to be properly chosen according to the practical applications. To clarify the performance of proposed attack models and constructions, we give detailed experiments on the real world dataset.

9. ACKNOWLEDGEMENTS

The authors are grateful to Yuan Zhang for the early discussions of this work. This work is supported by National Natural Science Foundation of China No.61272512 and Beijing Municipal Natural Science Foundation No.4121001.

10. REFERENCES

[1] Google trends. http://www.google.com/trends/.

[2] Stopword list. http://www.ranks.nl/resources/stopwords.html.

[3] Enron email dataset, 2009. http://www.cs.cmu.edu/∼enron/.

[4] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi. Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. In *Advances in Cryptology - CRYPTO'05*, volume 3621 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2005.

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

[6] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Advances in Cryptology - EUROCRYPT'04*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.

[7] Y.-C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Applied Cryptography and Network Security (ACNS'05)*, volume 3531 of *Lecture Notes in Computer Science*, pages 391–421. Springer, 2005.

[8] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. *Journal of Computer Security*, 19(5):895–934, 2011.

[9] E.-J. Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. http://eprint.iacr.org/.

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

[11] M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *Network and Distributed System Security Symposium (NDSS'12)*, 2012.

[12] S. Kamara and K. Lauter. Cryptographic cloud storage. In *Financial Cryptography Workshops*, volume 6054 of *Lecture Notes in Computer Science*, pages 136–149. pringer, 2010.

[13] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *ACM Conference on Computer and Communications Security (CCS'12)*, pages 965–976, 2012.

[14] M. Kuzu, M. S. Islam, and M. Kantarcioglu. Efficient similarity search over encrypted data. In *IEEE International Conference on Data Engineering (ICDE'12)*, pages 1156–1167, 2012.

[15] R. Ostrovsky. Efficient computation on oblivious rams. In *ACM Symposium on Theory of Computing (STOC'90)*, pages 514–523, 1990.

[16] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searching on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55, 2000.

[17] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou. Secure ranked keyword search over encrypted cloud data. In *IEEE International Conference on Distributed Computing Systems (ICDCS'10)*, pages 253–262, 2010.