2012 32nd International Conference on Distributed Computing Systems Workshops

Phrase Search over Encrypted Data with Symmetric Encryption Scheme

Yinqi Tang *, Dawu Gu*, Ning Ding* and Haining Lu*
*Lab of Cryptography and Computer Security
Shanghai Jiao Tong University, Shanghai, China 30332–0250
Email: {tangyinqi, dwgu, dingning, hnlu}@sjtu.edu.cn

***Abstract*—We study the case of searching over encrypted data from a remote server. In order to retrieve the encrypted documents that satisfy a client's criteria , a special index must be built and sent by the client together with encrypted documents. A trapdoor will also be produced to offer the privilege to search on the index. In the area of searchable encryption, many works mainly focused on search criteria consisting of a single keyword or conjunctive keywords. Up until now, searching of the exact documents that contain a phrase, or consecutive keywords still remains an unsolved problem.**

We first define the model of phrase search over encrypted data with symmetric encryption and its security definition based on the latest security definition raised by R. Curtmola. Then we propose a construction for phrase search with symmetric encryption(PSSE), which meets the functionality of searching a phrase over encrypted documents securely and efficiently. The computing complexity of our scheme when performing a query is linear in the size of the phrase, and at a moderate communication cost between server and client as well. In addition, we prove that our scheme achieves non-adaptive security.

***Index Terms*—phrase search; searchable encryption; symmetric encryption scheme**

I. Introduction

As cloud computing prevails, more and more clients choose to outsource data to an untrusted remote server. Hence, all files have to be encrypted before transmission. However, encryption prohibits client users from directly searching the encrypted data. To address this dilemma, a lot of advanced techniques have been proposed to ensure both confidentiality and searchability can be simultaneously satisfied. This is achieved by establishing a special index for the set of documents in question, over which all the following queries will be performed, to get the correct set of documents based on the search criteria. The Bloom Filter is an example of a structure designed for the secure index first raised in 1976[9]. [1] [8] proposed schemes for search single keyword based on secure index.

It is well known that searching by phrases over plaintext enjoys a wide range of applications. In many cases the ability of searching by a single keyword or a boolean combination of keywords over encrypted documents is good enough. However, oftentimes, it can be more efficient to search by phrases instead of discrete keywords in many cases. Most existing proposals for multi-keyword search fail to offer the ability to judge whether or not the keywords occurring in the document are consecutive. The reason being that most secure indexes are established based on the distinct word sets contained within the documents. The foundational scheme introduced in [1] offers the functionality to search by phrases but suffers from extremely high computational complexity that is linear to the number of words in the whole document . As far as multi-keyword search is concerned, all the works up until now mainly focused on conjunctive keyword search.

Our contribution: Some previous schemes were proposed for conjunctive keyword search based on asymmetric searchable encryption schemes which provide better functionality but suffer from computing efficiency. In this paper, we first proposed a provable secure-phrase-search scheme with symmetric searchable symmetric encryption, while maintaining a moderate computational cost in searching and at a moderate communication cost between the client and server, both linear in the phrase size. Additionally, we prove that our construction satisfies non-adaptive security, the latest security definition proposed by R. Curtmola[4].

The rest of the paper is organized as follows: Section II introduces related work on symmetric searchable encryption . Section III provides a definition of phrase based search with symmetric encryption. And in Section IV and V we describe the security definition of phrase search with symmetric encryption. Section VI explains the construction of phrase search with symmetric encryption and Section VII defines the security proof given in Section IV and V. Section VIII analyzes the performance and evaluates the update function. Section IX contains our conclusion.

II. Related Work

The concept of searchable encryption was first introduced by Song, Wagner, and Perrig in[1]. It can be classified into two categories according to the entities and key number involved: symmetric searchable encryption (SSE) and asymmetric searchable encryption(ASE).

Symmetric searchable encryption(SSE): SSE is a scheme that only those who possess the shared key to the encrypted files are able to search over the encrypted files. The functionality of SSE can be achieved by using general techniques like oblivious RAMs[10] and fully homomorphic encryption[11], but the communication and computation complexity is too large to be practical. Song, Wagner, and Perrig[1] first proposed a specific construction by means of PRGs, PRFs, and PRPs to search by a single keyword, conjunctive keywords, or even phrases at a cost linear to the size of the document needed to search a single keyword. [1] lacks an adequate security definition (IND-CPA) and leaks too much information to the server. E.g. , a server is able to determine whether two documents contain the same word after one query, and whether two documents are statistically similar after running many queries. [3] first proposed the notion of a "secure index" and the security definition of IND2-CKA (Indistinguishability against Chosen-Keyword Attacks), a stronger security definition than in [1] and [2]. And [3] offered an efficient construction whose computational complexity is constant for single-keyword based search although it is a probabilistic scheme which may return false positives. [2] proposed a simulation-based security definition and offered two deterministic constructions which are based on PRFs and PRPs. Its computational complexity is constant for single-keyword based search, identical to [3] and better than[1]. Therefore the computation time increases linearly with the size of the database to search the whole server. The latest security definition is proposed by [4]. It also describes two other efficient constructions for SSE which lower the computational complexity to constant O(1) when searching by a single keyword over the whole server. They also claim to achieve multi-user SSE function by means of broadcast encryption, however, considering all schemes proposed up until now, phrase based search with high efficiency still remains an open problem.

Asymmetric searchable encryption(ASE): ASE is a scheme that enables other parties besides the data owner to make queries to the server, as long as having access to the owner's public key(e.g:[5][6][7][8]). [5] first introduced the concept of public keyword encryption and described a construction of PEKS based on bilinear maps. ASE supports the scenarios of multi-user data sharing and collaboration, which SSE fails to offer. However, ASE suffers from low computing efficiency.

III. Definition For Phrase Search With Symmetric Encryption Scheme

The model of phrase search with symmetric encryption scheme(PSSE) involves two entities:

a) Sever: is responsible for storing client's data, searching for what the client requests and returning the correct results.

b) Client: is responsible for establishing the structured index of document against which the following query is performed, sending queries, and calculating the intermediate results.

A. The Phrase Search with Symmetric Encryption Scheme

The PSSE Scheme consists of 6 parts: Keygen(s), BuildIndex(), two Trapdoor() in two interactions, Search(), and TEST ().

(1) SK←Keygens(s): is a probabilistic key generation algorithm run by the client to setup the scheme. It takes as input a security system parameter with which it returns a secret key **SK**.

(2) ($\mathbf{I_D}$, **C**)←BuildIndex(**D**, SK):is a probabilistic algorithm run by the user to encrypt the document collection. It takes as input the secret key **SK** and a document collection $\mathbf{D} = (D_1, D_2, ..., D_n)$, and outputs a secret index $\mathbf{I_D}$ and a sequence of ciphertexts $\mathbf{c} = (c_1, c_2, ..., c_n)$. $\mathbf{I_D}$ consists of two parts: a binary matrix M and look-up table A, whose detailed definition is introduced in Section VI. Sometimes we write it as $(\mathbf{I_D}, \mathbf{C}) \leftarrow BuildIndex_{SK}(\mathbf{D})$.

(3) ($\mathbf{T_a}$, $\mathbf{T_b}$)←Trapdoor(**SK**, phrase):is a deterministic algorithm that run by the client to generate the trapdoor of the phrase that he wants to search. It takes as input **SK**, and a phrase to search by, and generates the trapdoor for phrase with **SK**. $\mathbf{T_a}$ is sent to the server and $\mathbf{T_b}$ is kept at the client side. Detailed definition of $\mathbf{T_a}$ and $\mathbf{T_b}$ is introduced Section VI.

(4) Document candidate set $\mathbf{id_D}$ ←Search($\mathbf{T_a}$, $\mathbf{T_b}$, **M**): is a deterministic algorithm that involves one round communication interaction between server and client. It takes as input the trapdoor of the phrase and the binary matrix M, which can be regarded as a bitmap of existence of words in documents, and returns the set of documents in which all the keywords of the phrase have occurred.

(5) $\mathbf{T'}_{phrase}$ ← Trapdoor'(**SK**, phrase, $\mathbf{id_D}$): is a deterministic algorithm that run by the client to generate the trapdoor of the phrase for the document candidate that he wants to test. It takes as input **SK**, phrase, and a set of candidate document identifiers, and generates the trapdoor for it.

(6) **D** ← TEST($\mathbf{T'}_{phrase}$, **A**): is a deterministic algorithm run by the server. It takes as input the trapdoor of the phrase and look-up table **A** for the document candidates, and returns the set of document in which the phrase occurs.

IV. Security Definition For Phrase Search With Symmetric Encryption Scheme

History: Let Δ be a dictionary and D$\subseteq 2^{\Delta}$ be a document collection over Δ. A q-query history over D is a tuple H = (**D**, **phrase**) that includes the document collection **D** and a vector of q queries: **phrase** $= (phrase_1, phrase_2, ..., phrase_q)$

Access Pattern: Let Δ be a dictionary and D$\subseteq 2^{\Delta}$ be a document collection over Δ. We denote $\mathbf{D}(phrase_i)$ as the document collection each of which contains the $phrase_i$, and $\mathbf{D'}(phrase_i)$ as the document collection each of which contains all words of the phrase. Assume that $|phrase_i|$≤k,1≤i≤q, suppose $phrase_i = \{w[1], w[...], w[k]\}$,$D_j \in \mathbf{D}(phrase_i)$, we introduce a $|\mathbf{D}| \times q$ matrix M here. Its each element: M[i][j] describes the situation of $phrase_i$ occurred in D_j. Actually, the element M[i][j] is an integer matrix $Integer_{k \times k}$, each element: Integer[u][v] representing the occurrence of sub-phrase$\{w[u], w[...], w[v]\}$ server has found while performing the "TEST" algorithm in document D_j. From the view of TEST algorithm, the occurrence of sub-phrase$\{w[u], w[...], w[v]\}$ is equal to the occurrence of $\{w[1], w[...], w[v]\}$. Note that all the occurrences of the sub-phrases mentioned in the following paper means what is defined in $\mathbf{M}_{|\mathbf{D}|\times q}$. Because that is what the server, or the potential adversary, could know during the search process. Therefore, the access pattern induced by a q-query history H=(**D**,**phrase**), is the tuple $\alpha(H) = \{(\mathbf{D}(phrase_1), \mathbf{D}(phrase_2), ..., \mathbf{D}(phrase_q)), (\mathbf{D'}(phrase_1), \mathbf{D'}(phrase_2), ..., \mathbf{D'}(phrase_q)), \mathbf{M}_{|\mathbf{D}|\times q}\}$

Search Pattern: Let Δ be a dictionary and D$\subseteq 2^{\Delta}$ be a document collection over Δ. Assume that $|phrase_i| \leq k, 1 \leq i \leq q$, we introduce a matrix here $T_{q\times k}$.Actually each element T[i][j] is a binary matrix $E_{q\times k}$, each element E[u][v] representing whether the j^{th} word of $phrase_i$ is the same with the v^{th} of $phrase_u$. Therefore, the search pattern induced by a q-query history $H = (\mathbf{D}, \mathbf{phrase})$, is the matrix $\sigma(H) = T$

Trace: Let Δ be a dictionary and D$\subseteq$ 2^{Δ} be a document collection over Δ. The trace induced by a q-query history H=($\mathbf{D}, \mathbf{phrase}$), is a sequence $\tau(H)$=($|D_1|, ..., |D_n|, \alpha(H), \sigma(H)$)comprised of the length of documents in $\overrightarrow{D}$, the access and search patterns induced by H.

Throughout this paper, we will assume that the dictionary Δ and the trace are such that all histories H over Δ are non-singular as defined below

Non-singular history: we say that a history H is non-singular if (1) there exists at least one history H'$\neq$H such that $\tau(H') = \tau(H)$; and if (2) such a history can be found in polynomial-time given $\tau(H)$.

Note that the existence of a second history with the same trace is a necessary assumption, otherwise the trace would immediately leak all information about the history.

V. Non-Adaptive Security Definition For Phrase Search With Symmetric Encryption

Non-Adaptive indistinguishability security:let PSSE = (Gen, Enc, Trpdr, Search, TEST, Dec) be an index-based PSSE scheme over a dictionary Δ, k$\in$N be the security parameter, and **A**=($\mathbf{A}_1, \mathbf{A}_2$) be a non-uniform adversary and consider the following probabilistic experiment $\mathbf{Ind}_{PSSE,A}(k)$:

```
$\mathbf{Ind}_{PSSE,A}(k)$
$K \leftarrow Gen(1^k)$
$(st_A, H_0, H_1) \leftarrow \mathbf{A}_0(1^k)$
$\beta \leftarrow \{0,1\}$
parse $H_\beta$ as $(\mathbf{D}_\beta, \mathbf{phrase}_\beta)$
$(\mathbf{M}_\beta, \mathbf{A}_\beta, \mathbf{C}_\beta) = (\mathbf{I}_\beta, \mathbf{C}_\beta) \leftarrow Enc_k(\mathbf{D}_\beta)$
for 2≤i≤q,
    $\mathbf{T}_{\beta,i} = (\mathbf{T}_{a,\beta,i}, \mathbf{T}_{a,\alpha,i}) \leftarrow Trpdr_k(phrase_{\beta,i})$
    $\mathbf{id_D} \leftarrow Search(\mathbf{T}_{a,\beta,i}, \mathbf{M}_\beta)$
    $\mathbf{T}'_{\beta,i} \leftarrow Trpdr'_k(phrase_{\beta,i}, \mathbf{id_D})$
let $\mathbf{t}_\beta = ((\mathbf{T}_{\beta,1}, \mathbf{T}'_{\beta,1}), ..., (\mathbf{T}_{\beta,q}, \mathbf{T}'_{\beta,q}))$
$\beta' \leftarrow A_2(st_A, \mathbf{I}_\beta, \mathbf{c}_\beta, \mathbf{t}_\beta)$
if $\beta' = \beta$,output 1
otherwise output 0
```

with restriction that $\tau(H_0 = \tau(H_1))$, and where st_A is a string that captures A's state. We say that our scheme is secure in the sense of non-adaptive indistinguishability if for all polynomial-size adversaries A=(A_1, A_2) such that q=poly(k)

$$Pr[Ind_{PSSE,A}(k) = 1] \leq \frac{1}{2} + negl(k),$$

Where the probability is over the choice of β,and the coins of Gen and Enc.

Non-Adaptive semantic security. Let PSSE =(Gen, Enc, Trpdr, Search, Test, Dec) be an index-based SSE scheme, k$\in$N be the security parameter, A be an adversary such that q$\in$N and S be a simulator and consider the following probabilistic experiments $\mathbf{Real}_{PSSE,A}(k)$ and $\mathbf{Sim}_{PSSE,A}(k)$

```
$\mathbf{Real}_{PSSE,A}(k)$
$k \leftarrow Gen(1^k)$
$(st_A, H) \leftarrow A(1^k)$
parse H as $(\mathbf{D}, \mathbf{phrase})$
$(\mathbf{M}, \mathbf{A}, \mathbf{C}) = (\mathbf{I}, \mathbf{C}) \leftarrow Enc_k(\mathbf{D})$
for 1≤i≤q
    $\mathbf{T}_i = (\mathbf{T}_{a,i}, \mathbf{T}_{b,i}) \leftarrow Trpdr_k(phrase_i)$
    $\mathbf{id_D} \leftarrow Search(\mathbf{T}_{a,i}, \mathbf{M})$
    $\mathbf{T}'_i \leftarrow Trpdr'_k(phrase_i, \mathbf{id}_D)$
let $\mathbf{t} = ((\mathbf{T}_1, \mathbf{T}'_1), ..., (\mathbf{T}_q, \mathbf{T}'_q))$
output v=$(\mathbf{I}, \mathbf{C}, \mathbf{t})$and $st_A$

$\mathbf{Sim}_{PSSE,A}(k)$
$(H, st_A) \leftarrow A(1^k)$
$\mathbf{v}$=$S(\tau(H))$
output $\mathbf{v}$ and $st_A$
```

we said that our scheme is semantically secure if for all polynomial-size adversaries A, there exists a polynomial-size simulator S such that for all polynomial-size distinguishers D,

$$|Pr[D(v, st_A) = 1 : (v, st_A) \leftarrow \mathbf{Real}_{PSSE,A}(k)] - Pr[D(v, st_A) = 1 : (v, st_A) \leftarrow \mathbf{Sim}_{PSSE,A}(k)]| \leq negl(k)$$

Where the probabilities are over the coins of Gen and Enc.

We now prove that the two definitions of security for non-adaptive adversaries are equivalent.

Theorem: Non-adaptive indistinguishability security of PSSE is equivalent to non-adaptive semantic security of PSSE.

Proof. Let PSSE=(Gen, Enc, Trpdr, Search, TEST, Dec) be an index-based PPSE scheme. We make the following two claims, from which the theorem follows.

Claim. If PSSE is non-adaptively semantically secure for PSSE, then it is non-adaptively indistinguishable for SSE.

We show that if there exists a polynomial-size adversary A=(A_1, A_2) that succeeds in an $\mathbf{Ind}_{PSSE,A}(k)$ experiment with non-negligible probability over $\frac{1}{2}$. Then there exists a polynomial-size adversary B and polynomial-size distinguisher D such that for all polynomial-size simulator S, D distinguishes between the output of $\mathbf{Real}_{PSSE,B}(k)$ and $\mathbf{Sim}_{PSSE,B}(k)$.

Let B be the adversary that computes $(st_A, H_0, H_1) \leftarrow A_1(1^k)$;samples b←{0,1};and outputs the history H_b and state $st_B = (st_A, b)$. Let D be the distinguisher that, given v and st_B(which are either output by $\mathbf{Real}_{PSSE,B}(k)$ or $\mathbf{Sim}_{PSSE,B}(k)$), works as follows:

1. It parses st_B into (st_A, b) and v into (**I**,**c**,**t**)
2. It computes $b' \leftarrow A_2(st_A, \mathbf{I}, \mathbf{c}, \mathbf{t})$,
3. It outputs 1 if b'=b and 0 otherwise.

Clearly, B and D are polynomial-size since A_1 and A_2 are. So it remains to analyze D's success probability. First, notice that if the pair (**v**, st_B) are the output of $\mathbf{Real}_{PSSE,B}(k)$ then **v**=$(\mathbf{I}_b, \mathbf{c}_b, \mathbf{t}_b)$ and $st_B = (st_A, b)$. Therefore, D will output 1 if and only if $A_2(st_A, \mathbf{I}_b, \mathbf{c}_b, \mathbf{t}_b)$ succeeds in guessing b. Notice, however, that A_1's and A_2's views while being simulated by B and D, respectively, are identical to the views they would have during an $\mathbf{Ind}_{PSSE,A}(k)$ experiment. We therefore have that

$$Pr[D(v, st_B) = 1 : (v, st_B) \leftarrow \mathbf{Real}_{PSSE,B}(k)] = Pr[\mathbf{Ind}_{PSSE,A}(k) = 1] \geq \frac{1}{2} + \varepsilon(k),$$

Where $\varepsilon(k)$ is some non-negligible-function in k and the inequality follows from our original assumption about A.

Let S be an arbitrary polynomial-size simulator and consider what happens when the pair (v, st_B) is output by a $\mathbf{Sim}_{PSSE,B,S}(k)$ experiment. First, note that any v output by S will be independent of b since $\tau(H_b) = \tau(H_0) = \tau(H_1)$ (by the restriction imposed in $\mathbf{Ind}_{PSSE,A}(k)$). Also, note that the string st_A output by A_1 (while being simulated by B) is independent of b. It follows then that A_2 will guess b with probability at most 1/2 and that,

$$Pr[D(v, st_B) = 1 : (v, st_B) \leftarrow \mathbf{Sim}_{PSSE,B}(k)] \leq \frac{1}{2}$$

Combining the two previous Equations we get that,

$$|Pr[D(v, st_B) = 1 : (v, st_B) \leftarrow \mathbf{Real}_{PSSE,B}(k)] - Pr[D(v, st_B) = 1 : (v, st_B) \leftarrow \mathbf{Sim}_{PSSE,B}(k)]| \leq \varepsilon(k)$$

from which the claim follows.

Claim. If PSSE is non-adaptively indistinguishable, then it is non-adaptively semantically secure.

We show that if there exists a polynomial-size adversary A such that for all polynomial-size simulators S, there exists a polynomial-size distinguisher D that can distinguish between the outputs of $\mathbf{Real}_{PSSE,A}(k)$ and $\mathbf{Ind}_{PSSE,A,S}(k)$, then there exists a polynomial-size adversary B=(B_1, B_2) that succeeds in an $\mathbf{Ind}_{PSSE,B}(k)$ experiment with non-negligible probability over $\frac{1}{2}$.

Let H and st_A be the output of $A(1^k)$ and recall that H is non-singular so there exists at least one history H'≠H such that $\tau(H') = \tau(H)$ and, furthermore, such a H' can be found efficiently. Now consider the simulator S^* that works as follows:

1. It generates a key $K^* \leftarrow Gen(1^k)$,
2. Given $\tau(H)$ it finds some H' such that $\tau(H') = \tau(H)$,
3. It builds an index $\mathbf{I}^*$, a sequence of ciphertexts $\mathbf{c}^*$ and a sequence of trapdoors $\mathbf{t}^*$ from H' under key K^*.
4. It outputs **v**=$(\mathbf{I}^*, \mathbf{c}^*, \mathbf{t}^*)$ and $st^* = st_A$.

Let D^* be the polynomial-size distinguisher (which depends on S^*) guaranteed to exist by our initial assumption. Without loss of generality we assume D^* outputs 0 when given the output of a $\mathbf{Real}_{PSSE,A}(k)$ experiment. If this is not the case, then we consider the distinguisher that runs D^* and outputs its complement.

B_1 is the adversary that computes $(H, st_A) \leftarrow A(1^k)$, uses $\tau(H)$ to find H'(as the simulator does) and returns (H, H',st_A)as its output. B_2 is the adversary that, given st_A and $(\mathbf{I}_b, \mathbf{c}_b, \mathbf{t}_b)$, sets **v**=$(\mathbf{I}_b, \mathbf{c}_b, \mathbf{t}_b)$ and outputs the bit b obtained by running $D^*(\mathbf{v}, st_A)$.

It remains to analyze B's success probability. Since b is chosen uniformly at random,

$$Pr[\mathbf{Ind}_{PSSE,B}(k) = 1] = \frac{1}{2}(Pr[\mathbf{Ind}_{PSSE,B}(k) = 1|b = 0] + Pr[\mathbf{Ind}_{PSSE,B}(k) = 1|b = 1]) \quad (1)$$

If b=0 occurs then B succeeds if and only if $D^*(v, st_A)$ outputs 0. Notice, however, that v and st_A are generated as in a $Real_{PSSE,A}(k)$ experiment so it follows that,

$$Pr[\mathbf{Ind}_{PSSE,B}(k) = 1|b = 0] = Pr[D^*(v, st_A) = 0 : (v, st_A) \leftarrow \mathbf{Real}_{PSSE,A}(k)] \quad (2)$$

On the other hand, if b=1 then B succeeds if and only if $D^*(v, st_A)$ outputs 1. In this case, v and st_A are constructed as in a Sim_{PSSE,A,S^*} experiment so we have,

$$Pr[\mathbf{Ind}_{PSSE,B}(k) = 1|b = 1] = Pr[D^*(v, st_A) = 1 : (v, st_A) \leftarrow \mathbf{Sim}_{PSSE,A,S^*}(k)] \quad (3)$$

Combining Equations (2) and (3) with Equation(1) we get

$$\begin{aligned} &Pr[\mathbf{Ind}_{PSSE,B}(k) = 1] \\ &= \frac{1}{2}(1 - Pr[D^*(v, st_A) = 1 : (v, st_A) \leftarrow \mathbf{Real}_{PSSE,A}(k)] \\ &+ Pr[D^*(v, st_A) = 1 : (v, st_A) \leftarrow \mathbf{Sim}_{PSSE,A,S^*}(k)]) \\ &= \frac{1}{2} + \frac{1}{2}(Pr[D^*(v, st_A) = 1 : (v, st_A) \leftarrow \mathbf{Sim}_{PSSE,A,S^*}(k)] \\ &- Pr[D^*(v, st_A) = 1 : (v, st_A) \leftarrow \mathbf{Real}_{PSSE,A}(k)]) \\ &\geq \frac{1}{2} + \varepsilon(k) \end{aligned}$$

Where $\varepsilon(k)$ is non-negligible function in k, and where the inequality follows from our original assumption about A.

VI. Our Construction

In this section we present our construction of SSE. First, we introduce some notations and some data structures with which we establish the index.

We divide each m documents into one group . We introduce 2 additional data structures here before we continue, a binary matrix M and a look-up table A. Given a binary matrix: M, we refer to M[i] as the i^{th} row of the M, which is an m-bit bit-string. As to matrix A, we refer to the element at i^{th} row and j^{th} column s A[i][j], and the i^{th} row as A[i]. We will denote "a||b" as a is concatenated by b.

Now we offer an overview of the construction of our scheme. We store a dictionary at the client side in which all words to be queried are stored. Each document will be encrypted with a symmetric encryption scheme by the client before being sent to the server side. Two data structures are involved to play the role as index: a binary matrix M for every m documents and a look-up table A for each document. They will be established during the BuildIndex() procedure. To search for a phrase, two rounds of communication between the client and server are needed for each query. During the first round of interaction, the server performs the search over the index of the binary matrix with the trapdoor sent from the client and returns the document candidates which contain every word of the phrase. In the second interaction, another trapdoor will be generated and sent from the client to help the server test whether each document candidate contains all words occurring as a phrase.

Before we step into details , we introduce the notations which are useful to construction.

A. Notation

We denote t,x,y,z, as four λ-bit secret keys of the client.

Besides, the following four polynomial time algorithm will be used during construction:

1) f_1:$\{0,1\}^\lambda \times \{0,1\}^d \to \{0,1\}^d$, a pseudo-random permutation. d is the bit-length of identifier for each word stored in client's dictionary, and λ is the bit length of client's secret key.

2) $f_2 : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$, a pseudo-random function. and λ is the bit length of client's secret key.

3) $g : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^m$, a pseudo-random function. and λ is the bit length of client's secret key, and m is the column number of matrix M.

4) $\pi : \{0,1\}^* \times \{0,1\}^{\lg d} \to \{0,1\}^{\lg d}$, a pseudo-random permutation. (d+1) is the column number of look-up table A.

5) $h : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^u$, a pseudo-random function. λ is the bit length of client's secret key, and u is the bit length of the output value.

6) $\Psi : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^{u+v}$, a pseudo-random function. λ is the bit length of client's secret key, and u is the bit length of the function h's output value, v is the bit length of the random number r_i.

B. Index establishment

Binary matrix M: We establish a $2^d \times m$ binary matrix for every m documents, where 2^d is donated as the dictionary size stored at the client side and m is the group size of documents that we build an index for at the same time. Each row of M indicates a word in the dictionary and each column indicates a document identifier. Firstly, each element, M[i][j], is initialized as zero. During the establishment, we map each word in the dictionary to a row in the matrix M by means of a pseudo-random permutation: $f_{1_x}(id_{word})$, here id_{word} is the identifier of a word stored in the local dictionary and x is the client's secret key. When scanning a document whose identifier is id module m, we set M[$f_{1_x}(id_{word})$][id] to 1 for the occurring *word*, otherwise let it remain zero.

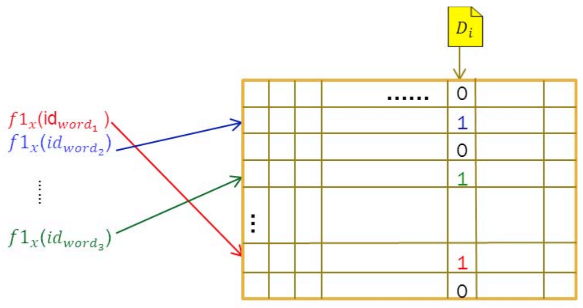

Fig. 1. binary Matrix M

Before sending the binary matrix M to the server side, we use a pseudo-random function $g_y(w[i])$, (here y is client's secret key), with the result of which $w[i]$'s corresponding row in M will make an XOR operation: M[$f_{1_x}(id_{word})$]$\bigoplus g_y(w[i])$

Look-up Table A for each document D: to record the position of each word of document D, we use a unique random number r_i to represent the i^{th} word in document D. We establish a $wcount \times (d+1)$ look-up table A for each document, here d is the estimated highest frequency of the word occurring in any document and wcount is the distinct word number of D. Each element of A is initialized as zero . We store the pseudo-random function value of each word concatenated by the document identifier: $\Psi_z(w[i]||id(D))$ in the first column of each row. In the end, the rows of look up table A will be sorted according to what is stored in the first column of each row. The remaining elements of A consist of 2 parts: output value of pseudo-random function of last random number concatenated by current random number:$h_s(r_{i-1})||r_i$. As for the first word of each document, $r^*||r_1$ is stored. r^* is just any v-bit random number. Note that the secret key s of pseudo-random function h is distinct for every two coherent words. It is generated by the pseudo-random function with $(w[i-1]||w[i]||id(D))$ as input parameter: $f_{2_t}(w[i-1]||w[i]||id(D))$, where t is the secret key of the client. The remaining elements of A which have not been fulfilled will be padded a (u+v)-bit random number. Before everything is finished, we permute each row from the 2^{nd} element until the last element of the same row with the pseudo-random permutation: $\pi_{w[i]}(currentcolumn-1)+1$, (with $w[i]$ as the parameter), to disorder the elements.

In the end, the look-up table A for each document and the binary matrix M of every m documents will be sent to the server side together with the secure indexes over which the upcoming queries will be performed.

C. How to search

The search procedure involves 2 rounds of communication: during the 1^{st} round, the client is able to get a set of candidate document identifiers who have all the words occurring in the target phrases. In the 2^{st} round, the server will test whether the words occur as a phrase in each candidate.

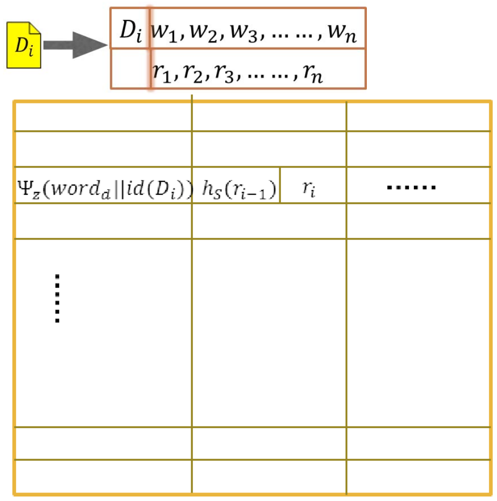
D_i
D_i $w_1, w_2, w_3, \dots\dots, w_n$
$r_1, r_2, r_3, \dots\dots, r_n$
$\Psi_z(word_d||id(D_i))$
$h_S(r_{i-1})$
r_i
......
⋮

Fig. 2. logic structure for document D and its look-up table A

We assume the phrase to be queried is $\{w[1], w[...], w[k]\}$. To simplify the following demonstration and without a loss of generalization, we suppose phrase=$\{w[1], w[2]\}$ is our search criteria..

The client looks up the words' identifiers in the client dictionary by means of binary search. During the first round, the client generates the trapdoor for the phrase=$\{w[1], w[2]\}$: $f_{1_x}(id_{w[1]}), f_{1_x}(id_{w[2]}), g_y(w[1]), g_y(w[2])$ with the secret keys x,y. The first half:$f_{1_x}(id_{w[1]}), f_{1[x]}(id_{w[2]})$, which indicates the corresponding row number of the binary matrix M will be sent to the server with which the server is able to find the corresponding row of w_1 and w_2 in the binary matrix M and return the binary string M[$f_{1_x}(id_{w[1]})$] and M[$f_{1_x}(id_{w[2]})$] to the client. Once the binary strings M[$f_{1_x}(id_{w[1]})$] and M[$f_{1_x}(id_{w[2]})$]are returned, the client is able to make an XOR operation with them by $g_y(w_1), g_y(w[2])$ respectively to get the plaintext of the two m-bit binary strings. Finally, the two binary strings undergo an "AND" operation to gain the word's occurrence information in the documents belonging to the current group. For example: if M[$f_{1_x}(id_{w[1]})$][5]=1, then the word $w[1]$ has occurred in a document whose identifier is 5. By carrying out the above steps, the client gains a set of document candidates **D**=$\{D_1, D_2, ..., D_\iota\}$ who have all words in the phrase occurring simultaneously.

During the second round of communication, the server will test whether each document candidate contains all of the words occurring as in the phrase being queried. Again the client generates the trapdoor for each document whose identifier is id: $\Psi_z(w[1]||id), \Psi_z(w[2]||id)$ and $f_{2_t}(w[i-1]||w[i]||id)$, here z, t is the client's secret key. Then the trapdoor is sent to the server side. Because the $\Psi_z(w[1]||id), \Psi_z(w[2]||id)$ is the first element of some row and each look-up table is sorted by the first element of each row, the server can search the corresponding row by binary searching the first column with the help of "wcount", the row number of the look-up table A.

With the entry of the corresponding row found, for example: k and l for the row of $w[1]$ and $w[2]$. The server is able to test whether the two words are located coherently by carrying out the following steps. As each element consists of (u+v) bit, we denote the right v bit as r_i(2≤i≤d+1), and the left u bit as β_j(2≤j≤d+1),the pseudo-random function's output value to be compared with. Then the server calculates the $h_{f_{2_t}(w[i-1]||w[i]||id)}(r_i)$(2≤i≤d+1)of A[k], and compares it to each β_j(2≤j≤d+1) of A[l]. If there is any equality existed, the server returns 1 to the client; otherwise 0.

If there are more than 2 words in the phrase, the above steps will be repeated for every 2 coherent words of the queried phrase except that in the second round we do not calculate every $h_{f_{2_t}(w[i-1]||w[i]||id)}(r_i)$, just those whose corresponding β_i have passed the test in the last round.

D. Construction in detail

- $Keygen(1^k)$
  - Generate the secret key **SK**: $K_1, K_2, K_3, K_4 \leftarrow \{0,1\}^k, K_5 \leftarrow SKE.GEN$

- ($\mathbf{I_D}$, C)←BuildIndex(D, SK):

  1) Initialization:
  - Generate a $2^d \times m$ binary matrix M, and initialize M[i][j]($1 \leq i \leq 2^d$,$1 \leq j \leq m$)as zero.

  2) Building the binary matrix:
  - For each document D_j(whose identifier is j), construct a distinct word set $\delta(D_j) = (w[1], w[2], ...)$
    - ∗ Set wcount = $|\delta(D_j)|$
    - ∗ For each word $w \in D_j$
      - · Look up for the id of w to the local dictionary stored at the client side.
      - · Calculate the corresponding row number for w by means of pseudo-random permutation:$f_{1_{K_1}}$, set M[$f_{1_{K_1}}$][j] to be 1.
  - For each word w stored in local dictionary at the client side(assume its identifier is id)
    - ∗ Calculate the $g_{K_2}(w)$
    - ∗ Make XOR operation: M[$f_{1_{K_1}}$]$\bigoplus g_{K_2}(w)$

  3) Building look-up table A: For each document D_q (whose identifier is q),
  - Establish a $wcount \times (d+1)$ look-up table A, wcount will be stored with A together. A[i][j]($1 \leq i \leq wcount, 1 \leq j \leq (d+1)$) is 0 string of length (u+v).
  - Generates a distinct random number r_i of length v to represent i^{th} word in D_q
  - Set m=0.
  - For i^{th} word $w[i]$ in $D_q(2 \leq i)$
    - ∗ Set $sk \leftarrow f_{2_{K_3}}(w[i-1]||w[i]||q)$
    - ∗ search it in the first column of A for: $\Psi_{K_4}(w[i]||q)$
      - · if it exists some row of A,assume it to be γ^{th} row, then find the first element which is not zero in A[γ], set it to be $H_{sk}(r_{i-1})||r_i$
      - · else if it doesn't exist

```
            a)A[m][0]= $\Psi_{K_4}(w[i]||q)$
            b)A[m][1]= $H_{sk}(r_{i-1})||r_i$
            c)$m \leftarrow (m+1)$
    – For each element in the look-up table A which is left to be zero, set it to a random number
    – Sort the rows according to the first element of each row
    – For each A[i]($1 \le i \le wcount$)
        * For A[i][j]($2 \le j \le (d+1)$)
            · j'=$\pi_{A[i][1]}(j-1)$
            · Swap(A[i][j'], A[i][j])
  4) The output($I_D$,C):
    – $\mathbf{I_D}$=(M, A)
    – For $1 \le i \le m$, set $c_i$=$SKE.ENC(K_5, D_i)$
```

```
• ($\mathbf{T_a}$,$\mathbf{T_b}$)←Trapdoor(SK, $phrase$):
    – Suppose the $phrase$= $\{w[1], w[...], w[k]\}$
    – Search each keyword in the local dictionary to find their identifier:$id_1, ..., id_k$
    – Calculate $f_{1_{K_1}}(id_1), ..., f_{1_{K_1}}(id_k), g_{K_2}(w[1]), ..., g_{K_2}(w[k])$
    – Set $T_a = f_{1_{K_1}}(id_1), ..., f_{1_{K_1}}(id_k)$
      $T_b = g_{K_2}(w[1]), ..., g_{K_2}(w[k])$
```

```
• Document candidate set $\mathbf{id_D}$←Search($T_a$,$T_b$,M):
    – parse $T_a$ as $(\gamma_1, ..., \gamma_k)$
    – Server sends back M[$\gamma_1$],...,M[$\gamma_k$] to the client
    – parse $T_b$ as $(\eta_1, ..., \eta_k)$
    – For $1 \le i \le k$
        * Set $\theta_i = M[\gamma_i] \bigoplus \eta_i$
    – Let $\theta = \theta_1$
    – For $1 \le i \le (k-1)$
        * $\theta = \theta \wedge \theta_i$
    – the m-bit binary string $\theta$ gives a document candidates set $\mathbf{id_D}$
```

```
• $T'_{phrase}$ ← Trapdoor'(SK,phrase,$\mathbf{id_D}$):
    – set $\mathbf{T}_a$=$(f_{2_{K_3}}(w[1]||w[2]), ..., f_{2_{K_3}}(w[k-1]||w[k]))$
    – for each document candidate whose identifier is q:
        * set $\mathbf{T}_{b,q} = (\Psi_{K_4}(w[1]||q), ..., \Psi_{K_4}(w[k]||q))$
    – set $T'_{phrase} = \{\mathbf{T}_a, \mathbf{T}_{b,q(q \in \mathbf{id_D})}\}$
```

```
• D ← TEST($T'_{phrase}$, A)
    – parse $T'_{phrase}$ as $\{\mathbf{T}_a, \mathbf{T}_{b,i(i \in \mathbf{id_D})}\}\}$,
      $\{T_a = (\zeta_1, ..., \zeta_{k-1})$,$T_{b,i} = (\delta_{1,i}, ..., \delta_{k,i})(i \in \mathbf{id_D})$
    – for each $i \in \mathbf{id_D}$, find its corresponding look-up table A
        * TEST($\mathbf{T}_a$, $\mathbf{T}_{b,i}$, A)
        * If the returned result is 0, the $i^{th}$ document is deleted from the document candidate set.
    – With the left document candidate identifier set, return the corresponding encrypted document to the client.
```

```
• TEST($\mathbf{T}_a, \mathbf{T}_b, A$)
    – parse $\mathbf{T}_a$ as $(\zeta_1, ..., \zeta_{k-1})$, $\mathbf{T}_b$ as $(\delta_1, ..., \delta_k)$
    – Binary search $\delta_1$ in the first the column of A, set the row number to be $\alpha_1$
    – set ctr=d,
    – for $1 \le i \le ctr$
        * flg[i]=i+1
    – for $1 \le i \le k$
        * Binary search $\delta_i$ in the first the column of A, set the row number to be $\alpha_1$
        * set count=0
        * for $1 \le j \le ctr$
            · Let r be the right v-bit of A[$\alpha_{i-1}$][flg[j]]
            · Let h be the pseudo-random function's output value:$h_{\zeta_{i-1}}(r)$
            · for $2 \le k \le d+1$
              Test:
              if h is equal to the left u-bit of A[$\alpha_i$][k],then
                a) count = count +1
                b) flg[count]=k
                c) break for
              else do nothing
        * set ctr = count
        * if ctr==0 break for; return 0
  return 1;
```

VII. Security Proof

In this section we state the proof of the security of PSSE in terms of the definition stated in section 4.

Theorem. As long as f_1 and π are pseudo-random permutations, and f_2, g, h, and Ψ are pseudo-random function, is PCPA-security and PSSE is non-adaptive secure.

Proof. We describe a polynomial-size simulator S such that for all polynomial-size adversaries A, the outputs of $Real_{SSE,A}k$ and $Sim_{SSE,A,S}k$ are indistinguishable. Consider the simulator S that given the trace of a history H, generates a string $\mathbf{v}^* = (\mathbf{I}^*, \mathbf{c}^*, \mathbf{t}^*) = ((\mathbf{M}^*, \mathbf{A}^*), (c_1^*, ..., c_n^*), (t_1^*, ..., t_q^*))$ as follows:

A. Simulating M^*:

above all, S establishes a $2^d \times m$ binary matrix M^*, and initializes M^*[i][j] as 0;

- If q=0, set the M^*[i][j] to be a random bit.
- Else if $q \ge 1$. Suppose **phrase** $= (phrase_1, ..., phrase_q)$. From the trace history, the simulator is able to get the information of $\mathbf{D}'(phrase_i)$, the documents that contain the all the words$\{w[1], ..., w[k]\}$ of $phrase_i$
- Assume $phrase_1$ is composed of the distinct words $\{w[1], ..., w[k_i]\}$. For $1 \le j \le k_1$, S generates pairs (α_j^*, β_j^*) such that the α_j^* are distinct strings of length 2^d chosen uniformly at random, and the β_j^* are distinct strings of length m chosen uniformly at random. For $\forall b, D_b \in \overrightarrow{D}'(phrase_1)$, sets$M^*[\alpha_j^*][b] = 1$. And

(α_j^*, β_j^*) is a trapdoor of $w[j]$, therefore $T_{phrase_1} = (T_a, T_b), T_a = (\alpha_1^*, ..., \alpha_{k_1}^*), T_b = (\beta_1^*, ..., \beta_{k_1}^*)$.

- Assume $phrase_l (2 \leq l)$ is composed of distinct words $\{w[1], ..., w[k_l]\}$. Note that simulator S is able to know whether some word has occurred before from the trace history. For $1 \leq j \leq k_l$, if $w[j]$ has occurred in the phrase before, S retrieves the trapdoor of $w[j] = (\alpha_j^*, \beta_j^*)$. if $w[j]$ didn't occur, S generates pairs (α_j^*, β_j^*) the same way as S did for $phrase_1$. And for $\forall b, D_b \in \mathbf{D}'(phrase_l)$ sets$M^*[\alpha_j^*][b] = 1$. And (α_j^*, β_j^*) is a trapdoor of $w[j]$, therefore $T_{phrase_l} = (T_a, T_b), T_a = (\alpha_1^*, ..., \alpha_{k_l}^*), T_b = (\beta_1^*, ..., \beta_{k_l}^*)$.
- In the end, for $\forall w[j]$ occurred in **phrase**, simulator S sets $M^*[\alpha_j^*] \bigoplus \beta_j^*$, where (α_j^*, β_j^*) such that the α_j^* is the trapdoor of $w[j]$. As to the rest of the row, simulator S generates distinct strings of length m chosen uniformly at random, with which S XORs the left rows respectively.

B. Simulating $\mathbf{A}^*$:

First, as the whole look-up table A is stored with the wcount together, and s is a system parameter, simulator S establishes a wcount×(d+1)look-up table A, and initializes A[i][j] as a (u+v)-bit 0.

- a) If q=0, for each M^*[i][j], simulator S selects a distinct string of length (u+v) chosen uniformly at random, and sets M^*[i][j] to be this value.
- b) Else if $q \geq 1$. Assume **phrase** $= (phrase_1, ..., phrase_q)$. From the trace history, the simulator is able to get the information of $\mathbf{D}'(phrase_i)$, the documents that contain all the words$\{w[1], ..., w[k]\}$ of $phrase_i$
- c) Assume $phrase_l (2 \leq l) = \{w[1], ..., w[k_l]\}. \forall D \in \mathbf{D}'(phrase_1)$, for distinct $w[j] \in phrase_1$, simulator S generates pairs (γ_j^*, θ_j^*) such that γ_j^* are distinct strings of length (u+v) chosen uniformly at random, and γ_j^* are distinct strings of length log(wcount). Simulator S sets A[θ_j^*][1]=γ_j^*. From the trace history, the simulator S is able to have an idea of how many times each combination:$\{w[i], w[i+1]\} (1 \leq i \leq (k_1 - 1))$ has appeared in D $\in \mathbf{D}'(phrase_1)$ either contributed by $phrase_1$ alone or by all the phrases. Assume that $phrase_1$ contributed t_i times of the combination $\{w[i], w[i+1]\}$ in D, and all the phrases together contribute t_i' times. Note that from the trace history, $t_1 \geq t_2 \geq ... \geq t_{k_1}$, but the relation among $t_1', t_2', ..., t_{k_1}'$ cannot be simply determined as $t_1, t_2, ..., t_{k_1}$ do. For some D $\in \mathbf{D}'(phrase_1)$ and D $\notin \mathbf{D}(phrase_1)$, there must exist some j($1 < j < k_1$)that $t_j = t_{j+1} = ... = t_{k_1} = 0$. Assume it's ρ from which t_ρ begins to be equal to 0. Firstly, for each combination in $phrase_1$: $\{w[i], w[i+1]\} (1 \leq i \leq (k_1 - 1))$, Simulator S generates a random string $\eta_i (1 \leq i \leq (k_1 - 1))$ of length λ chosen uniformly at random, as the secret key of pseudo-random function. Secondly, for the first combination, simulator S generates t_1 pairs $(r_{1_1}, r_{2_1}), (r_{1_2}, r_{2_2}), ..., (r_{1_{t_1}}, r_{2_{t_1}})$ such that r_* are distinct strings of length v chosen uniformly at random. Then for $1 \leq j \leq t_1$, simulator S sets $A^*[\theta_1^*][j+1] = r^* || r_{1_j}$, where r^* is a distinct string of length u, and simulator S calculates $h_{\eta_1}(r_{1_j})$ and sets $A^*[\theta_2^*][j+1] = h_{\eta_1}(r_{1_j}) || r_{2_j}$. Thirdly,
  - For $\forall D \in \mathbf{D}(phrase_i)$: for the following each of the combinations in $phrase_i : \{w[i], w[i]\} (3 \leq i \leq k_1 + 1)$, simulator S generates t_{i-1} random numbers: $(r_{i_1}, r_{i_2}, ..., r_{i_{t_1}})$ for each combination such that r^* are distinct strings of length v chosen uniformly at random. Then for each j:$1 \leq j \leq t_{i-1}$, simulator S calculates $h_{\eta_{i-1}}(r_{(i-1)_j})$, and sets $A^*[\theta_i^*][j+1] = h_{\eta_{i-1}}(r_{(i-1)_j}) || r_{i_j}$
  - For $\forall D \in \mathbf{D}'(phrase_1) and D \notin \mathbf{D}(phrase_1)$, simulator S performs the same steps as is described above for the following combination in $phrase_i$: $\{w[i], w[i]\}$ except that $(3 \leq i \leq \rho + 1)$

  The left trapdoor for $phrase_1$ is $T'_{phrase_1} = \{(\gamma_1^*, ..., (\gamma_{k_1}^*), (\eta_1, ..., \eta_{k_1 - 1})\}$
- d) For $2 \leq i \leq q$, Assume $phrase_i = \{w[i], ..., w[k_i]\}$. For D $\in \mathbf{D}'(phrase_i)$, if $D \notin \{\mathbf{D}'(phrase_1), ..., \mathbf{D}'(phrase_{i-1})\}$, then note that from the trace history, simulator S is able to know whether $(w[1], ..., w[k_i])$ is a sub-phrase of some previously queried phrase appearing in D.
  - If it is, obviously the A contains any sub-phrase of the previously occurring phrases in D. Therefore simulator S does nothing.
  - If $phrase_i$ is not a sub-phrase of any previously queried phrase appearing in D , then it contains such a phrase. For simplicity and without a loss of generality, we assume:
    - 1. From the trace history, simulator S is able to know both sub-phrases' occurrence. If the occurrence of sub-phrases of the previous one is consistent with its counterpart in $phrase_i$, we assume that the previous phrase actually is part of $phrase_i$. Thus when fulfilling the look-up table A, simulator S performs the same steps as is in c) except that S does not generate any new random numbers or secret keys using the pseudo-random function for the words in the sub-phrase which is identical to the previously queried phrase.
    - 2. Otherwise, we assume that they are two independent phrases sharing the same sub-phrase, hence simulator S performs the same steps as is in c), generating distinct string η_* of length λ chosen uniformly at random for each combination of secret key from the pseudo-random function and distinct random number r_* of length v chosen uniformly at random for each word to further fulfill the look-up table A of document D.
  - Else if neither $phrase_i$ is a sub-phrase of any previously queried phrases appearing in document D nor is any previously queried phrase a sub-phrase of $phrase_i$,then there exists an interaction which plays

the role as the head and tail of each respectively. From the trace history, the simulator is able to know whether the occurrence of each sub-phrase in the interaction is identical to its counterpart.

  * 1. If it is, we assume that they actually share the same sub-phrase as head and tail respectively. Without a loss of generality, we assume that the same sub-phrase is the $phrase_i$'s tail. (Reverse is symmetric.) Simulator S performs the same steps as is in c) except that S does not generate any new random numbers or secret keys from the pseudo-random function for the words in the sub-phrase.
  * 2. Else if it is not, we assume that they are two independent phrases sharing the same sub-phrase, hence simulator S performs the same steps as is in c), generating distinct string η_* of length λ chosen uniformly at random for each combination as secret key of pseudo-random function and distinct random number r_* of length v chosen uniformly at random for each word to further fulfill the look-up table A of document D.

- Else if by no means we can view the two phrases as part of each other from the trace history, we assume that they are two independent phrases. Hence simulator S performs the same steps as is in c), generating distinct string η_* of length λ chosen uniformly at random for each of the two combinations of secret key from the pseudo-random function and distinct random number r_* of length v chosen uniformly at random for each word to further fulfill the look-up table A.

The left trapdoor for $phrase_i$ is $T'_{phrase_i} = \{(\gamma_1^*, ..., (\gamma_{k_i}^*), (\eta_1, ..., \eta_{k_i-1})\}$

- e) Before the end, the simulator S performs additional steps: firstly S sets all elements that are left to be zero in A to be distinct strings of length (u+v) chosen uniformly at random, and makes a permutation of each row from the second column until the last one. Secondly, S sorts all the rows of each look-up table A by comparing it to the element in the first column.
- f) As to those A unchanged at all, just set each zero element to be a distinct string of length (u+v) chosen uniformly at random.

C. Simulating t_i^*:

For $phrase_i$, trapdoor=$(\mathbf{T}_a, \mathbf{T}_b, \mathbf{T}'_{phrase_i})$,where $\mathbf{T}_a = (\alpha_1^*, ..., \alpha_{k_i}^*)$, $\mathbf{T}_b = (\beta_1^*, ..., \beta_{k_i}^*)$, and $\mathbf{T}'_{phrase_i} = \{(\gamma_1^*, ..., \gamma_{k_i}^*), (\eta_1, ..., \eta_{k_i-1})\}$

D. Simulating c_1^*:

It sets c_i^* to a $|D_i|$-bit string uniformly at random. Note that simulator S is able to know $|D_i|$ from the trace history.

It follows by construction that by searching over $(\mathbf{M}^*, \mathbf{A}^*)$ with trapdoor $(t_1^*, ..., t_i^*)$, we can get the same results.

Let V be the outcome of a $Real_{PSSE,A}(k)$experiment. We now claim that no polynomial-size distinguisher D that receives st_A is able to distinguish between the distribution V and V^* otherwise D is able to distinguish between at least one of the elements of V and its corresponding element in V^*.

We now prove that it is impossible by showing that each element of V^* is indistinguishable from its corresponding element in V to a distinguisher D that is given st_A.

1) (M and M^*): Recall that each row of M was XOR-ed with the output of function g, and each row of M^* was XOR-ed with a random string β_j^* of length m. With all but negligible probability, st_A does not include the PRF key K_2 for function g, and therefore the pseudo-randomness of g guarantees that each element M is indistinguishable from its counterpart in M^*.

2) (A and A^*): Recall that the elements of the first column in each A are strings of length (u+v) generated by pseudo-random function f_2 and its counterparts in A^* are random strings γ_j^* of length (+). With all but negligible probability,st_A does not include the PRF key K_3 for function f_2, and therefore the pseudo-randomness of f_2 guarantees that each element of A's first column is indistinguishable from its counterpart in A^*. Besides, the element from the 2^{nd} column until the last one in A was fulfilled with a u-bit pseudo-random function's output value concatenated by a u-bit pseudo-random function's output value concatenated by a (v+u)-bit random number. So it is with A^* that each such element was indistinguishable from a random number.

3) (t_i and t_i^*): Recall that each t_i was composed of PRP f_1 PRF f_2, g, Ψ, with all but negligible probability st_A will not contain the secret key random number or a K_3, K_2, K_4, so the pseudo-randomness of f_1, f_2, g, Ψ will guarantee that each t_i will be indistinguishable from the counterpart of t_i^*

4) c_i and c_i^*): Recall that c_i is $SKE.ENC$ encryption. Therefore, with all but negligible probability, st_A will not contain the secret key k_5, the PCPA-security[4] of $SKE.ENC$ will guarantee that each c_i is indistinguishable from its counterpart c_i^*.

VIII. Performance Analysis

Assume that *k* is the word count of a phrase, *d* is the column number of table A, and ζ is the size of the dictionary stored at the client side.

What the client has to store locally is a dictionary on which all of the following phrase searches have to cover, and 5 λ bit secret keys.

Things become a little complicated as far as the server is concerned. We look into the computing complexity firstly, and then come to the storage cost.

During the first round of interaction, it takes O(k) for the client to calculate all the trapdoors and handle the results for a k-word phrase in addition to the time dealing with an m-bit binary string.

In the second round of interaction, the server has to test every document candidate. When testing a document candidate, it takes O($\lg(wcount)$) for the server to search for every word's corresponding row in look-up table A, where wcount is the word count in each document, and O(d^2) to find an equation between two rows. Therefore, it takes O((k-1)$\times d^2$) to test a document candidate. Assuming there are $|\mathbf{D}|$ document candidates at all, the server's computing complexity when searching is O($|\mathbf{D}| \times \{(k-1) \times d^2 + k \times \lg(wcount_{id})\}$), (id$\in$ $\mathbf{id_D}$). As d is a system parameter and $\lg(wcount_{id})$ is a constant once the indexes are established, the computing complexity of searching at the server side is linear to the size of the number of words in a phrase.

The server stores the encrypted documents and their secure indexes: matrix M and look-up table A. As far as matrix M is concerned, it takes ζ bits for the server to store each document. While considering the look-up table A, assume the highest occurrence of a word in all documents is ζ, which is represented by *d* here. And assume the average occurrence of a word in all documents is σ. If each word occupies 5 characters, or 40 bits on average, while each element of the look-up table is allocated for (u+v) bits. Then it takes the server $\frac{(d+1)\times(u+v)}{40\sigma}$ times the size of the original document collection to store the look-up table As.

IX. Conclusions And Future Work

In this paper, we have further studied the problem of searchable encryption, which solves the dilemma of maintaining the confidentiality of data and the ability for a client to search. We first introduce the model of phrase search with symmetric encryption and its security definition, then propose a construction and its security proof. Lastly, we analyze our scheme and evaluate how it performs when updating.

Our scheme achieves non-adaptive security under the security definition we gave in section 4. But it is not the strongest, at least it does not meet the criteria of adaptive security, which is raised by R. Curtmola[4]. A construction, which satisfies the criteria of adaptive security, still remains an open problem.

Acknowledgment

This research work was sponsored by Morgan Stanley - Shanghai Jiao Tong University Innovation Center of Computing in Financial Services (No. CIP-A20110324-1), by Major Program of Shanghai Science and Technology Commission(No. 10DZ1500202) and National Natural Science Foundation of China (61100209). After this paper, we will still focus on the field of searchable encryption and other security issues in cloud computing environment.

References

[1] D. Song, D. Wagner, and A. Perrig, *Practical techniques for searching on encrypted data*, In IEEE Symposium on Research in Security and Privacy, pages 44-55. IEEE Computer Society, 2000.

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

[3] E-J. Goh, *Secure indexes*, Technical Report 2003/216, IACR ePrint Cryptography Archive, 2003. See http://eprint.iacr.org/2003/216.

[4] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky,*Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions*, In ACM Conference on Computer and Communications Security (CCS '06), pages 79-88. ACM, 2006.

[5] D. Boneh et al, *Public key encryption with keyword search*, Advances in Cryptology-Eurocrypt'04, LNCS 3027, pages 506-522, 2004.

[6] J. Baek, R. Safavi-Naini, at el, *Public key encryption with keyword search Revisited*, In International conference on Computational Science and Its Applications, pages 1249-1259. Springer-Verlag, 2008.

[7] P. Golle, J. Staddon, and B. Waters, *Secure conjunctive keyword search over encrypted data*, AIn M. Jakobsson, M. Yung, and J. Zhou, editors, Applied Cryptography and Network Security Conference (ACNS '04), volume 3089 of Lecture Notes in Computer Science, Springer, 2004.

[8] D. Boneh and B. Waters, *Conjunctive, subset, and range queries on encrypted data*, In Theory of Cryptography Conference (TCC '07), volume 4392 of Lecture Notes in Computer Science, Springer, 2007.

[9] Burton H. Bloom, *Space/Time Trade-offs in Hash Coding with Allowable Errors*, Communication of the ACM, 1976.

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

[11] C. Gentry, *Fully homomorphic encryption using ideal lattices*, In ACM Symposium on Theory of Computing (STOC '09), pages 169-178. ACM Press, 2009.