Privacy-Preserving Multiple Keyword Search on Outsourced Data in the Clouds*

Tarik Moataz[1,2], Benjamin Justus[2], Indrakshi Ray[1], Nora Cuppens-Boulahia[2], Frédéric Cuppens[2], and Indrajit Ray[1]

[1] Computer Science Department, Colorado State University, Fort Collins, USA
{tmoataz,indrajit,iray}@cs.colostate.edu
[2] Institut Mines-Télécom, Télécom Bretagne, Cesson Sévigné, France
{benjamin.justus,nora.cuppens,frederic.cuppens}@telecom-bretagne.eu

Abstract. Honest but curious cloud servers can make inferences about the stored encrypted documents and the profile of a user once it knows the keywords queried by her and the keywords contained in the documents. We propose two progressively refined privacy-preserving conjunctive symmetric searchable encryption (PCSSE) schemes that allow cloud servers to perform conjunctive keyword searches on encrypted documents with different privacy assurances. Our scheme generates randomized search queries that prevent the server from detecting if the same set of keywords are being searched by different queries. It is also able to hide the number of keywords in a query as well as the number of keywords contained in an encrypted document. Our searchable encryption scheme is efficient and at the same time it is secure against the adaptive chosen keywords attack.

1 Introduction

Data is often stored in an encrypted form in the clouds for security and privacy reasons. If the volume of the stored data stored is large, it may be infeasible for the client to download the encrypted data, decrypt them locally, and search for the relevant documents. Consequently, researchers have proposed searchable encryption schemes to perform searches on encrypted documents stored in the clouds. Such schemes allow cloud servers to retrieve multiple encrypted documents in response to a client's queries which may be keywords search or numerical range queries. Efficiency of such techniques, which impact the query response time, is critical. Moreover, protecting the privacy of the client against honest but curious servers is also important.

Researchers have proposed schemes that permit exact keyword search on encrypted documents [10,11] as well as conjunctive keyword search (please see Section 2). However, existing conjunctive keyword search schemes do not provide adequate levels of privacy. Often times, the server is aware of the number

* This work was partially supported by the U.S. National Science Foundation under Grant No. 0905232.

V. Atluri and G. Pernul (Eds.): DBSec 2014, LNCS 8566, pp. 66–81, 2014.

of keywords contained in a document and the number of keywords in a query. Moreover, in these schemes, search on identical set of keywords result in the same encrypted query. An honest but curious cloud server may become aware of the client's search information and gain knowledge about her profile. Such information when correlated with the additional knowledge possessed by the server may constitute a serious privacy leakage. We propose a privacy-preserving approach that protects against such information leakage without increasing the search and storage complexities but at the cost of two rounds of protocol.

We motivate our approach by using a simple example based on exact keyword search. Each document stored in the server is associated with a bit vector whose size depends on the number of keywords in the dictionary. The index i on the vector corresponds to the i^{th} keyword in the dictionary. A value of "1" in index position i signifies that the i^{th} keyword is present in the document. The search query is associated with a similar bit vector. An `and` operation of the bit vectors corresponding to the document and the query reveals the existence or non-existence of the keyword. Such a simple scheme, however, does not provide adequate privacy protection. The server is now aware of the number of keywords in the document and the query. Moreover, it can detect if the user is submitting the same query multiple times. This, together with some background knowledge possessed by the server, can cause serious privacy leakage. We propose a more privacy preserving approach by introducing noise that serves to hide the number and the content of keywords and also is able to randomize the queries. We call this augmented approach Privacy-Preserving Symmetric Searchable Encryption (PCSSE) scheme. We introduce in this article first a rudimentary scheme that we refer to as PCSSE-1 that introduces the concept of noise insertion. However, this scheme is vulnerable to two types of inference attacks. We address these attacks in our second scheme that we refer to as PCSSE-2. One major technical challenge in implementing the PCSSE scheme is how to introduce the noise such that both privacy and correctness of the queries are preserved. The location of noises should be random or else an adaptive adversary can observe the search history and infer the keyword information. The PCSSE-2 scheme is secure against such an adaptive adversary based on the security of the pseudo-random permutation primitives as well as the randomness of the noise generation.

Our approach preserves the privacy of the client. First, the server does not have any information about the number of keywords contained in a document. Second, the scheme hides the number of keywords contained in a query. Last but not the least, PCSSE-2 provides query randomization that generates different encrypted queries even when the client is searching for the same set of keywords. This protects the client against revealing his search pattern to the server. Note that, there are some techniques that have better search complexity and search expressiveness, but they leak the search pattern information.

We have implemented a proof-of-concept for the PCSSE-2 scheme to evaluate its performance. Our approach is efficient with respect to storage and network communication costs. Specifically, the PCSSE-2 scheme has a storage complexity linear in the size of documents and the size of the indexes on the server side. The

query phase of the scheme requires two rounds of client-server communication and has a communication complexity linear in the dictionary size. Moreover, the server-client communication during the query verification stage can be carried out efficiently as well since the size of the query, represented in the form of a binary vector, is at most 4 KB.

The rest of the paper is organized as follows. Section 2 discusses the state-of-the-art on searchable encryption. Section 3 presents an overview of our approach and contains the PCSSE algorithm and security definitions. PCSSE-1 construction is carried out in Section 4 while PCSSE-2 is detailed in Section 5. Section 6 evaluates the security and the performance of our scheme. Section 7 concludes the paper.

2 Related Work

Song et al. [20] presented the first symmetric searchable encryption scheme. Subsequently, many works have focused on enhancing the search and storage complexities of the scheme, as well on the strength of the security models [11,8,10,16,15]. The study of asymmetric searchable encryption started with the work of Boneh et al. [4]. Some of the later works focused on providing techniques that improve on the search complexity of Boneh's scheme [1,3,9]. All these constructions deal with exact keyword search, and do not have the capability of performing conjunctive keyword search.

The existing exact keyword searchable encryption schemes are not suitable for a conjunctive search. They disclose sensitive meta-information, and at the same time induce an exponential computation overhead on the server side. Golle et al. [13] introduced the first conjunctive scheme in a symmetric setting. Golle's scheme associates a searchable index with each document. The server performs a matching test on the document index with the client submitted query. Golle presented two constructions: the security of the first construction is based on the hardness of the Decisional Diffie-Hellman (DDH) problem. The first construction has a search complexity linear in the number of stored documents. The second construction has a search complexity linear in the number of dictionary keywords. Both constructions use exponentiations and pairings in the search phase and test phase. Parker et al. [18] presented a similar scheme which handles the asymmetric setting. Ballard et al. [2] later enhanced the scheme's communication and search complexities on the server side. Ryu et al. [19] presented another symmetric solution which additionally reduces the complexity of the encryption phase by diminishing the number of pairing operations. On the other hand, some works deals with multiple keyword search in the cloud by enabling some enhanced privacy preserving techniques [6,21]. Moreover, some works introducing boolean search over encrypted data [17,7] still disclose too many information to the outsourced servers. Specifically, these works fail to hide the search pattern.

Boneh's work [5] extended the search options in a public setting that allow conjunctive subsets, ranges and exact keywords searches. Hwang et al. [14] presented an enhancement for the cipher-text size with a comparable computation

complexity. All the above conjunctive searchable encryption schemes leak the following information: the number of keywords contained in each document, and the number of keywords that are in the client's query. These meta-information leakage causes a breach of the client's privacy. Recent work by Wang et al. [22] allows clients to hide this information. Wang's scheme however is not deterministic in the sense that there are false positives associated with each of the client's query. Furthermore in order to minimize the query's false positive rate, the client can store at most 16 keywords in each document, and cannot have more than 4 keywords in a given query.

3 Overview of the Approach

3.1 Problem Statement

Let $\mathcal{D} = \{D_1, ..., D_n\}$ be a collection of documents, and $\mathcal{W} = \{w_1, ..., w_l\}$ be a dictionary of keywords. The collection $\mathcal{D}$ is encrypted using a private-key CPA-secure encryption scheme $\mathcal{E}$. The encrypted collection $\mathcal{E}(\mathcal{D})$ is outsourced and stored on external cloud servers. The dictionary $\mathcal{W}$ is a finite set whose size depends on the underlying language.

Let $\{w_k\}_{k \in I_C}$ be a list of keywords contained in the dictionary $\mathcal{W}$ where I_C contains the keyword indexes. The search query contains a conjunction of keywords, denoted by, $\bigwedge_{k \in I_C} w_k$. The objective of the conjunctive searchable encryption is to retrieve the set of encrypted documents that contain the set of keywords from the encrypted collection $\mathcal{E}(\mathcal{D})$ in an efficient manner, so as to preserve the privacy of the client to the extent possible. In response to the query, the server sends the set of encrypted documents to the client.

3.2 Notations

The notation $x \xleftarrow{R} \mathcal{S}$ means uniform sampling the string x from the set S. In this paper, the strings x is a binary string (i.e. $x \xleftarrow{R} \{0,1\}^*$). The size of string x is denoted by $|x|$. Let x and y be two binary strings, then $x \parallel y$ is the concatenation of x and y. Let A be an array, $A[i]$ is the value of the array at i^{th} index. The transpose of array A is represented by A^T. If $\mathcal{AL}$ is an algorithm then $x \leftarrow \mathcal{AL}(...)$ represents the result of applying the algorithm $\mathcal{AL}$ with given arguments. The parameter k is used to denote a security parameter.

Let $\mathcal{D} = \{D_1, ..., D_n\}$ be a set of documents, and $\mathcal{W} = \{w_1, ..., w_l\}$ a dictionary of keywords. Each document D_i is associated a set of keywords. The set of keywords in D_i is represented by the set $M_i = \{r_{i,1}, r_{i,2}, \ldots, r_{i,|M_i|}\}$, where $|M_i|$ is the number of keywords in D_i and each $r_{i,j}$, where $1 \leq j \leq |M_i|$, gives the position of the keyword in the dictionary, that is, $1 \leq r_{i,j} \leq l$. Moreover, the set of all keywords in all the n documents are given by $\mathcal{M}$, where $\mathcal{M} = \{M_1, ..., M_n\}$. The set of keywords that a client searches for is given by I_C, and the conjunction of these keywords is denoted by $C(\mathsf{W}) = \bigwedge_{k \in I_C} w_k$.

Definition 1 ***Hamming Weight:*** *The Hamming weight $H(x)$ of a binary vector x is defined to be the number of "1"s in the vector x.*

Consider the inner scalar product ϕ on an inner product space: $\phi : \{0,1\}^n \times \{0,1\}^n \to [\![0, n]\!]$. The value $\phi(x, x)$ is a non-negative integer, and it coincides with the Hamming weight of x. Note that, $\phi(x, y) = \sum_{k=1}^{n} x_k y_k$, where x and y are vectors of size n.

Our construction is based on several well known cryptographic primitives. These include private-key CPA-secure encryption scheme (e.g. AES), pseudorandom permutation. We refer the reader to [12] for details.

3.3 PCSSE Protocol

Our PCSSE protocol over a set of documents $\mathcal{D}$ and a dictionary space $\mathcal{W}$ consists of applying five polynomial-time algorithms, namely, *KeyGen*, *Enc*, *Query*, *Response*, *Dec*, each of which is briefly enumerated below.

Key Generation $(K_1, K_2) \leftarrow KeyGen(k)$: $KeyGen$ takes the security parameter k as input and outputs two secret keys K_1 and K_2.

Encryption $(\mathcal{L}, \mathcal{C}) \leftarrow Enc(K_1, K_2, \mathcal{D}, \mathcal{M}, \mathcal{W})$: *Enc* takes as inputs the secret keys K_1, K_2, the collection of documents $\mathcal{D}$, the set of keyword index $\mathcal{M}$, and the dictionary $\mathcal{W}$. It outputs the encrypted collection of documents $\mathcal{C} = \{C_1, ..., C_n\}$ and the associated set of labels $\mathcal{L} = \{L_1, ..., L_n\}$.

Randomized Query Generation $Q \leftarrow Query(K_2, C(\mathsf{W}))$: *Query* is a probabilistic algorithm that takes as inputs the secret key K_2 and the conjunction of keywords $C(\mathsf{W})$. It outputs a randomized query Q .

Query Response $\mathcal{X} \leftarrow Response(Q, \mathcal{L})$: *Response* is a deterministic algorithm that takes as inputs the query Q and the set of labels $\mathcal{L}$ associated with the encrypted documents. The output consists of a set of encrypted documents $\mathcal{X}$ that match the query Q.

Decryption $\mathcal{D}_i \leftarrow Dec(K_1, \mathcal{C}_i)$: *Dec* is a deterministic algorithm that takes as inputs the key K_1 and a ciphertext C_i and outputs the unencrypted document D_i.

Definition 2 ***Correctness:*** *Let $C(\mathsf{W})$ be a conjunctive keywords query, and $\mathcal{C}$ the set of encrypted documents that match the conjunctive query $C(\mathsf{W})$. Let Q be the result of applying the probabilistic algorithm Query on $C(\mathsf{W})$. We say that PCSSE is correct if:*

$$Response(Q, \mathcal{L}) = \mathcal{C}.$$

3.4 PCSSE Security Definition

In our PCSSE scheme, we are interested in the adaptive security model such as the one introduced in [10], namely, secure against chosen keywords attack CKA-2. In this security model, no pseudorandom polynomial-time adversary, who

is given encrypted labels, encrypted documents and encrypted search queries, can learn any information about the content of the documents and the content of search queries other than the *search pattern* and *access pattern*. The access pattern contains information about the identifiers of encrypted documents that match a search query. The search pattern contains the history of all the search queries. We refer the reader to [10] for the formal definitions of search and access patterns. The search pattern, access pattern, the number of documents and the size of documents returned in a query response, are the possible information leakages in an adaptive security model. Note that, the number of keywords in a document is also a possible source of information leakage in existing symmetric searchable encryption schemes; our PCSSE scheme keeps the number of keywords secret (see section 5).

The important characteristic of randomized queries is that no polynomial time adversary is able to discern whether a search query is repeated or not. This is because the query submitted by a client is different each time, even when searching for the same conjunction of keywords. We formally define the notion of randomized query as follows:

Definition 3 ***Randomized query:*** *Let $\{Q_i\}_{1\leq i\leq t}$ be the t queries generated with the $Query(.)$ algorithm with the same key K_2 and the conjunctive keyword expression $\{C(\mathsf{W}_i)\}_{1\leq i\leq t}$. Let Q_i and Q_j two queries for the same conjunction of keywords. We say that the scheme satisfies randomized query if no pseudorandom polynomial time adversary can, with high probability, associate these queries to the same conjunction.*

As a consequence of using randomized queries, the information leakage is now limited to the access pattern, the number and the size of encrypted documents. However, we want to point out that as we will see later on in the security analysis section, the access pattern is inherently related to the search pattern. In other words, knowing the access pattern can infer as well some information about the user search behavior.

Moreover, with randomized queries the number of keywords sent in each query is completely hidden from the server. Following upon the definition 3, we classify the information leakage into two categories. The *leakage* $\rho_1(\mathcal{L},\mathcal{C})$ represents a statistical leakage known by the server before receiving any search queries. The second *leakage* $\rho_2(Q_{i\in\{1,\cdots,t\}},\mathcal{L},\mathcal{C})$ deals with access patterns disclosed by the search queries results. These are defined below.

Definition 4 ***Leakage*** $\rho_1(\mathcal{L},\mathcal{C})$*: The leakage consists of the following information: the number of encrypted documents, the size of each encrypted document, and the identifier of each encrypted document stored on the server.*

Definition 5 ***Leakage*** $\rho_2(Q_{i\in\{1,\cdots,t\}},\mathcal{L},\mathcal{C})$ *(Access pattern): If each Q_i is a randomized query, the leakage caused after submitting Q_i consists of the identifiers of encrypted documents that match the queries $Q_{i\in\{1,\cdots,t\}}$.*

In the following, we define security of our PCSSE scheme using a simulation based definition.

Definition 6 ***Adaptive security against chosen keyword attack CKA2:*** *Let us consider the five algorithms that defines our PCSSE scheme, namely,* KeyGen, Enc, Query, Response, Dec *as described in Section 3.3. We present the following game based on two experiences* $\mathbf{Real}_{\mathcal{A}}$ *and* $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}$ *where we consider a stateful adversary* $\mathcal{A}$*, a stateful simulator* $\mathcal{S}$*, the leakages* ρ_1*,* ρ_2*, and the security parameter* k*:*

> $\mathbf{Real}_{\mathcal{A}}(k)$*: The challenger (user) runs the* $KeyGen(k)$ *algorithm and outputs the key* (K_1, K_2)*.* $\mathcal{A}$ *sends the tuple* $(\mathcal{D}, \mathcal{M}, \mathcal{W})$ *and receives* $(\mathcal{L}, \mathcal{C}) \leftarrow Enc(K_1, K_2, \mathcal{D}, \mathcal{M}, \mathcal{W})$ *from the challenger.* $\mathcal{A}$ *makes a polynomial number of adaptive queries* $\{C(\mathsf{W}_i)\}_{1 \leq i \leq t}$ *and sends them to the challenger. The adversary receives the search queries generated by the challenger such that* $Q_i \leftarrow Query(K_2, C(\mathsf{W_i}))$*.* $\mathcal{A}$ *returns one if his queries return the expected result, otherwise zero is returned.*
> $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(k)$*: The adversary outputs the tuple* $(\mathcal{D}, \mathcal{M}, \mathcal{W})$ *and sends it to the simulator. Given the leakage* ρ_1*,* $\mathcal{S}$ *will generate the labels as well as the encrypted documents* $(\mathcal{L}, \mathcal{C})$ *and sends them to the adversary.* $\mathcal{A}$ *makes a polynomial number of adaptive queries* $\{C(\mathsf{W}_i)\}_{1 \leq i \leq t}$ *and sends them to the simulator. Given the leakage* ρ_2 *(containing the access pattern or previous queries), the simulator sends the appropriate search queries to the adversary. Finally,* $\mathcal{A}$ *returns a zero or one depending on whether or not the responses are accurate.*

We say that PCSSE is adaptively secure against chosen keyword attack if for all polynomial time adversary $\mathcal{A}$*, there exists a non-uniform polynomial-size simulator* $\mathcal{S}$ *such that:*

$$|\Pr[\mathbf{Real}_{\mathcal{A}}(k) = 1] - \Pr[\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(k) = 1]| \leq \epsilon(k)$$

4 PCSSE-1 Construction

In this section, we present the construction of the first scheme PCSSE-1. This scheme represents a fundamental basis for the second scheme where we introduce globally all our techniques. PCSSE-1 suffers as will be described in the subsequent section 5 from some notable attacks that will lead us to present the enhancement version of it.

The scheme consists of two phases, namely, the *setup phase* and the *search phase*. The setup phase is done once by the client when the encrypted documents and the labels are uploaded on the server. In this phase, the client uses the algorithms *KeyGen* and *Enc* to construct the labels and create the encrypted documents. The search phase is performed every time a query is submitted. In this phase, the client generates the randomized query using the algorithm *Query* and the the server generates the response using the algorithm *Response*. Since the PCSSE algorithm is a two-rounds protocol, the *Response* algorithm includes one additional interaction with the user before outputting the results.

4.1 Setup Phase

The client uses the $KeyGen(k)$ algorithm with the security parameter k to generate the keys K_1, K_2 such that $K_i \in \{0,1\}^k$.

In the following we will detail the $Enc(.)$ algorithm. In order to encrypt document D_i and generate the label L_i associated with D_i, the client performs the following steps.

[Step 1:] Create an array L_i of size $3l$ where l is the size of the dictionary and initialize it with all zeros.

[Step 2:] Choose randomly a subset P in $[\![1, 3l]\!]$ such that $|P| = l$. Apply the permutation function $\pi : \{0,1\}^k \times \{0,1\}^l \rightarrow P$ to the keyword index $M_i = \{r_{i,1}, ..., r_{i,|M_i|}\}$ associated with the document D_i to obtain a permuted index set $\pi_{K_2}(M_i) = \{\pi_{K_2}(r_{i,1}), ..., \pi_{K_2}(r_{i,|M_i|})\}$. For each $\pi_{K_2}(r_{i,j}) \in \pi_{K_2}(M_i)$, where $1 \leq j \leq |M_i|$, set $L_i[\pi_{K_2}(r_{i,j})] = 1$.

[Step 3:] Choose randomly a subset R in $[\![1, 3l]\!] \backslash P$ such that $|R| = l$. Sample randomly two binary vectors a_i and b_i such that $a_i \parallel b_i \xleftarrow{R} \{0,1\}^{|R|}$. Fill L_i in such a way that L_i restricted to the positions R is $a_i \parallel b_i$, and L_i restricted to the positions $Z = [\![1, 3l]\!] \backslash \{P \bigcup R\}$ is $a_i \parallel \overline{b_i}$.

[Step 4:] Client encrypts each document D_i using a private-key CPA-secure encryption scheme $\mathcal{E}_{K_1}$ to produce the encrypted document $\mathcal{E}_{K_1}(D_i)$. Client sends the server the encrypted document along with the label, denoted by $(\mathcal{E}_{K_1}(D_i), L_i)$. The user stores in his side $h(a_i)$, the hamming value of the vector a_i.

4.2 Search Phase

The search step includes both the $Query(.)$ and the $Response(.)$ algorithm.

Let $C(\mathsf{W}) = \bigwedge_{k \in I_C} w_k$ be the conjunction of keywords that the client searches for, where I_C is the set of keywords positions in the dictionary such $I_C = \{r_1, ..., r_{|I_C|}\}$. The client performs the three steps of the $Query(.)$ and sends the randomized query to the server such that:

[Step 1:] Create an array Q of size $3l$ where l is the size of the dictionary and initialize it with all zeros.

[Step 2:] Fill the array Q such that $Q[\pi_{K_2}(r_j)] = 1$ for $1 \leq j \leq |I_C|$.

[Step 3:] Randomly sample two binary vectors c and d such that $c \parallel d \xleftarrow{R} \{0,1\}^{|R|}$. Fill Q in such a way that Q restricted to the positions R is $c \parallel d$, and Q restricted to the positions Z is $\overline{c} \parallel d$. The user sends the query Q to the server and retains the value $h(d) + |I_C|$.

Once the server receives the query Q, it performs the following steps for the *Response* algorithm:

[Step 1:] The server does the following computation: for each label L_i such that $Token_i = Q.L_i^T$, the server sends the $Token_i$ to the user.

[Step 2:] The user performs the following verification: if $Token_i = h(a_i) + h(d) + |I_C|$, the user sends a bit b_i equal to 1, otherwise the bit will be equal to 0.

[**Step 3:**] If the server receives a bit b_i equal to 1, the server sends the encrypted document $\mathcal{E}_{K_1}(D_i)$ to the user.

Finally the user invokes the algorithm $Dec(\cdot)$ in order to decrypt the received documents.

Theorem 1. *The PCSSE-1 construction is correct as per Definition 2.*

5 PCSSE-2 Construction

PCSSE-1 as constructed in the previous section is vulnerable to two critical attacks, referred to as *Attack 1* and *Attack 2*, that can leak the position of the noise and therefore the position of the keywords. We describe first the attacks and then present PCSSE-2 that is going to address these deficiencies.

***Attack 1* description.** The positions of the noise defined by the set R and Z represent a vital information that must be kept secret. Thus, in the PCSSE-1 construction, the hamming values of both binary vectors a_i and d are not shared with the server. Note that if the verification was done in the server side, it can infer directly the value $|I_c|$ (by knowing the values of $h(a_i)$, $h(d)$ and $Token_i$). However, even if the critical part of the verification is done on the client side, the server will finally know all the matching documents (by receiving a bit equal to 1 or 0 for each label). The server then has the knowledge of all labels matching the corresponding query. Let us denote by $(L_1, \cdots, L_q)$ the set of labels matching a query Q. The server knows that all these labels have all the positions of searched for keywords equal to 1. The idea is to reduce the number of ones caused by noise insertions. In fact since the noise is randomly inserted in these fixed positions, some labels will contains '0' and other will contain '1' in the same position. The server then performs a binary 'AND' operation between all these labels such that:

$$L = L_1 \wedge L_2 \wedge \cdots \wedge L_q$$

The resulting L is a binary vector containing fewer 1 values. The most important consequence is that the server now has the certainty that all positions that turn from 1 to 0 are actually a noise position while some of the positions keeping a '1' value correspond to keywords' positions. After executing a number of queries, the server will build an exact knowledge of the noise and keywords' positions.

***Attack 2* description.** The first attack is adaptive in the sense that the adversary needs the result of the search to build an extra-knowledge of the labels' construction. The second attack is more destructive in the sense that the adversary can infer noise positions defined by the sets R and Z *passively*. In fact, the determinism of noise position is the key of this adversary's attack. Even if the noise is randomly inserted independently for each label it does not change anything to its fixed position which is similar to all labels. The noise vector a_i is inserted twice in the label L_i. This redundancy is the origin of the problem (the

same for the vector b_i and its complementary). As an instance, if the first bit of a_i is inserted in $R[1]$, then the same bit will be inserted in $Z[1]$, this applies to all labels. Consequently, an attacker can follow this strategy: first select a random bit in the first label and search for a position containing the same bit value. Secondly, check the second label in both positions whether they have the same bits. If so, go to the third label and continue the same test, if not, go back to the first label and choose a different position and go through the process recursively. The attack outputs the noise positions with high probability, which, in turn, reveals the keywords' positions *passively*. Note that, this attack is computationally expensive.

In the following we present PCSSE-2. We detail the noise insertion in the setup phase and the search phase. We generate the keys with the $KeyGen(k)$.

Setup Phase. Steps 1, 2 and 4 are similar to the PCSSE-1 construction. We will detail here the new strategy of noise insertions. Subsequently, we use the term *translation by a nonnegative integer* q applied to a set P. Here we give an example showing this concept. Let us take the ordered set $S = \{2, 5, 10, 20, 35\}$. The translation by a scalar value 2 applied on S will output the new set $\{20, 35, 2, 5, 10\}$. The translation will shift all elements by q modulus the size of the set.

1. Choose randomly a subset R in $[\![1, 3l]\!] \backslash P$ such that $|R| = l$. The positions in R are ordered. The same applies for Z.
2. Sample randomly two binary vectors a_i and b_i such that $a_i, b_i \xleftarrow{R} \{0,1\}^l$.
3. Choose randomly two nonnegative integers l_i^1 and l_i^2 smaller than l.
4. Apply respectively a translation to the positions in R and Z by l_i^1 and l_i^2 and output the new ordered set of positions R_i and Z_i.
5. Fill L_i in such a way that L_i restricted to the positions R_i is $a_i \parallel a_i$, and L_i restricted to the positions Z_i is $b_i \parallel \overline{b_i}$.

Figure 1 describes the process of translation applied to the first label.

Search Phase. For the $Query(.)$ algorithm, Steps 1 and 2 are similar to those of the PCSSE-1 construction. Here we detail the new strategy for the noise insertion in the query:

1. Sample randomly two binary vectors c and d such that $c, d \xleftarrow{R} \{0,1\}^l$.
2. Choose randomly two nonnegative integers[1] l^1 and l^2 smaller than l.
3. Apply respectively a translation to the positions in R and Z by l^1 and l^2 and output the new ordered set of positions R' and Z'.
4. Fill Q in such a way that Q restricted to the positions R' is $c \parallel \overline{c}$, and Q restricted to the positions Z' is $d \parallel d$.

Once the server receives the query Q, it performs the following steps for the *Response* algorithm:

[Step 1:] The server performs the following computation: for each label L_i such that: $Token_i = Q.L_i^T$, the server sends the $(Token_1, \cdots, Token_n)$ to the user.

[1] These integers are different for each newly generated query.

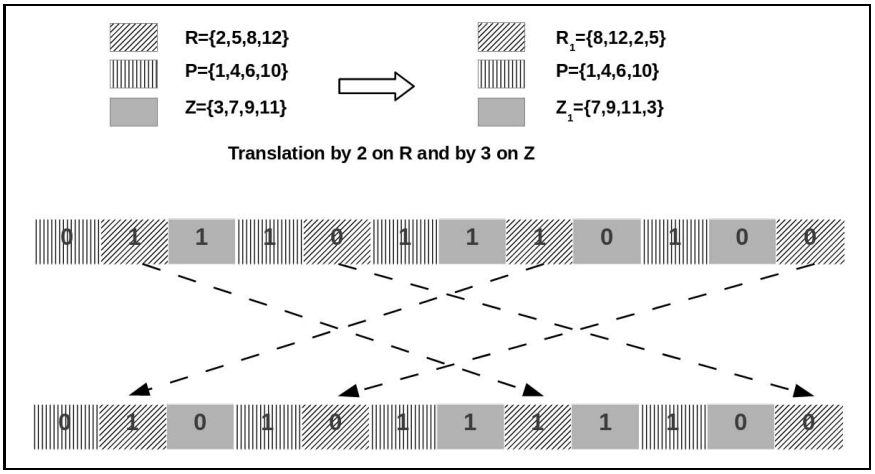

Fig. 1. Translation process applied to a label where $a_i = (1, 0)$ and $b_i = (1, 1)$ and the size of the dictionary l equal to 4

[**Step 2:**] The user performs the following verification: if $Token_i = h(a_i) + h(d) + |I_C|$, the user outputs a bit b_i equal to 1, otherwise the bit will be equal to 0. Finally the user outputs $\mathbf{b} = (b_1, \cdots, b_n)$.

[**Step 3:**] The user randomly selects r (if available) bits equal to 0 from $\mathbf{b}$ and turns them to 1. The user will then output the new version $\widetilde{\mathbf{b}}$.

[**Step 4:**] If the server receives a bit $\widetilde{b_i}$ equal to 1, the server sends the encrypted document $\mathcal{E}_{K_1}(D_i)$ to the user.

Finally, the user invokes the algorithm $Dec(\cdot)$ in order to decrypt the received documents. The choice of the value r can be limited such that the exact number of matching labels plus r will not exceed a given value. This will give the user a control on the communication complexity.

Theorem 2. *The PCSSE-2 construction is correct as per Definition 2.*

Discussion. The ideas introduced in PCSSE-2 solve both attacks described earlier. Including false bits in the user answer, will prevent the server from computing the exact value of $L = L_1 \wedge L_2 \wedge \cdots \wedge L_q$ after each query. In fact, this idea will mislead the server by including some noisy labels that do not contain the query. This solution will not create any false positive but will increase the communication complexity (not the number of interactions).

The second countermeasure taken into account in the PCSSE-2 construction phase keeps the position of noise fixed, but introduces a translation to change the position of the noise vector and its redundant value or its complimentary value for each label. This translation is done randomly for each label and is independent from that done to another label. The position of a noise bit and its complement in one label has no relation with the corresponding positions in the second label. The same reasoning applies also to the query since we are changing the scalars of the translations from a query to another.

6 Security Analysis and Evaluation

6.1 Security Analysis

The privacy guarantees of our PCSSE-2 scheme can be expressed in terms of the following lemma and theorem. For lack of space, we omit the proofs but offer an intuition instead.

Lemma 1. *Let Q_i be the outputs of the algorithm $Query(K_2, C(\mathsf{W}_i))$ for $1 \leq i \leq t$. The PCSSE-2 primitive has randomized queries such that:*

$$\forall i, j \in [\![1, t]\!] \text{ s.t. } i \neq j,\ \Pr(Associate(Q_i, Q_j)) \precsim \frac{1}{\binom{h(Q_i \vee Q_j)}{|I_C|}},$$

where $h(Q_i)$ is the Hamming value of the query Q_i with known number of conjunction I_C.

Theorem 3. *Let $\mathcal{E}$ be a private-key CPA-secure encryption scheme, π a pseudo-random permutation. Then the PCSSE-2 scheme is adaptively secure against chosen keywords attack as defined in Definition 6 where ρ_1 and ρ_2 are the possible leakages, and the queries are randomized in the sense of Definition 3.*

Proof Sketch. The query randomization property is the key to the privacy of our PCSSE-2 scheme. The randomized query gives an attacker negligible chances of discerning the keywords searched by a client or whether the same/similar keyword searches have been performed in the past. As a direct consequence of this property, the adversary (server) cannot find out the number of keywords in a given query. Since the label construction is similar to the query construction, the adversary also cannot infer the number of keywords inside each document.

However, we also need to prove that the indexes and the encrypted documents are indistinguishable from simulated ones based on the leakages previously defined in section 3.4. Note that, the keywords positions are computed using a pseudo-random permutation (PRP) with a key that is secret to the user. The adversary, while in simulation, has to generate randomly a key and use it for the PRP. Consequently the adversary's generated keyword positions and the real positions are indistinguishable. The index is constructed as well by inserting random noise which makes every pair of indexes different even if they contain the same keywords. Thus, an adversary's simulated index and the real index are indistinguishable. The same applies for the generated query since it follows the same index construction. Moreover, the encrypted documents are indistinguishable from the real ones as well since the adversary has to simulate a random key for this purpose. Lastly, during the search phase, the adversary has an additional knowledge about the query results. However with a position of noise different from every two indexes, and a user's answer containing always extra documents, the adversary cannot define the keywords positions. Thus our scheme is secure against adaptive chosen keyword attacks.

6.2 Evaluation

In summary, Table 1 compares our PCSSE-2 scheme with other deterministic conjunctive symmetric schemes with respect to computation complexity, and query privacy properties. The notations n, m and $|C|$ denote respectively the number of documents, the number of keywords and the number of keywords within a given conjunction. We use **exp**, **pr** to designate the operations exponentiation and pairing[2]. p is a 128-bit prime. Finally we use **IP**, **MCG** and **PRF** to designate the inner product, the multiplication in a cyclic group, and a pseudo-random function respectively.

Table 1. Comparison of Conjunctive Symmetric Searchable Encryption Schemes

	Server side computation	Server side storage	Client side computation	Query computation	Query size	Randomized Query	Hide the number of keywords
GSW-1 [13]	n **exp** + n **PRF**	$(m+1)pn+$ $\|\mathcal{E}(\mathcal{D})\|$	$(m+1)n$ **exp** + nm **PRF**	n **exp** + $(\|C\|+n)$ **PRF**	$(n+1)p+$ $\|C\|$	No	No
GSW-2 [13]	$(2\|C\|+1)n$ **pr**	$(2m+1)pn+$ $\|\mathcal{E}(\mathcal{D})\|$	$(2m+1)n$ **exp** + nm **PRF**	3 **exp** + $\|C\|$ **PRF**	$3p+\|C\|$	No	No
BKM [2]	$2n$ **pr** + $n\|C\|$ **PRF**	$(m+1)pn+$ $\|\mathcal{E}(\mathcal{D})\|$	nm **PRF** $+(m+1)n$ **MCG**	$\|C\|$ **PRF**	$2p+\|C\|$	No	No
ET [19]	$2n$ **pr**	$(m+2)pn+$ $\|\mathcal{E}(\mathcal{D})\|$	$(m+1)n$ **exp** + n **pr** + mn **PRF**	2 **exp**+ $\|C\|$ **PRF**	$2p+\|C\|$	No	No
PCSSE (our approach)	n **IP**	$3ln+$ $\|\mathcal{E}(\mathcal{D})\|$	nm **PRF**	$\|C\|$ **PRF**	$3m$	Yes	Yes

We run experiments to evaluate the performance of the search phase of PCSSE-2 as this directly impacts the server's real-time response capabilities. We ignore the time required for query construction as this step can be carried out in constant time; instead, we focus on the query verification that is carried out on the server. Our experiments on the server query-verification stage investigates how PCSSE-2 performs asymptotically taking into account the number of keywords and the number of documents. In the experiment, we have considered up to 1000 file, not taking into account their types. A file can be a document, email, without a specific file type, i.e. it can be a document, email, media etc.). For each scenario, tests are performed on three different dictionary sizes: 1000, 5000, and 10000 keywords. The PCSSE-2 primitive is scripted and tested inside the open-source Scilab environment. The computations are performed on a dell laptop with 2.40GHz processors. The results are plotted in Fig. 2. The plot shows

[2] Computing one pairing is equal to 6 to 20 exponentiation.

that the performance is linear with respect to the number of documents n. Furthermore, the label size increase implies only a constant overhead added to the search time. The search time per-document is roughly $30\mu s$ for labels containing 1000 keywords, and $200\mu s$ for labels containing 10000 keywords. The plot gives us a clear idea how PCSSE-2 performs asymptotically.

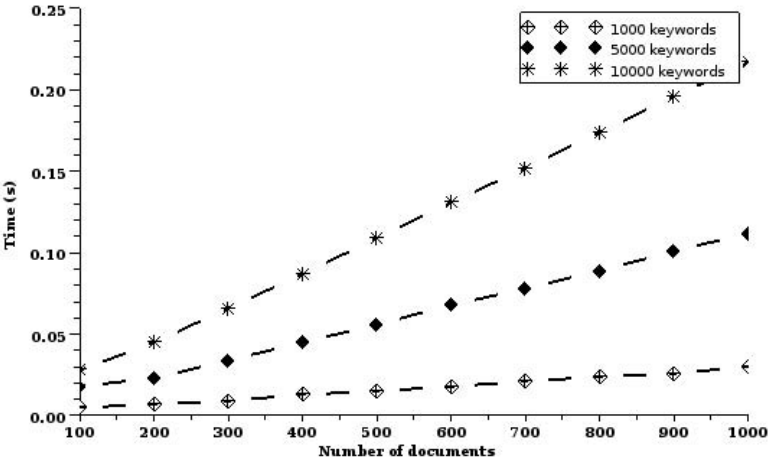

Fig. 2. Performance Evaluation for the Search Phase

7 Conclusion

We have presented a new deterministic privacy-preserving conjunctive symmetric searchable encryption scheme (PCSSE) that allows cloud servers to perform efficient conjunctive keywords searches on encrypted documents while protecting the privacy of clients by hiding the keywords in the query and also the keywords contained in the document.

We plan to extend this work along two dimensions. First, we would extend PCSSE scheme to include sub-match on keywords and also boolean expression searches. Second, we would like to make the scheme more dynamic that will allow efficient and privacy preserving queries even if the documents on the server side are updated.

References

1. Abdalla, M., Bellare, M., Catalano, D., Kiltz, E., Kohno, T., Lange, T., Malone-Lee, J., Neven, G., Paillier, P., Shi, H.: Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. Journal of Cryptology 21(3), 350–391 (2008)

2. Ballard, L., Kamara, S., Monrose, F.: Achieving efficient conjunctive keyword searches over encrypted data. In: Qing, S., Mao, W., López, J., Wang, G. (eds.) ICICS 2005. LNCS, vol. 3783, pp. 414–426. Springer, Heidelberg (2005)
3. Bellare, M., Boldyreva, A., O'Neill, A.: Deterministic and efficiently searchable encryption. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 535–552. Springer, Heidelberg (2007)
4. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer, Heidelberg (2004)
5. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 535–554. Springer, Heidelberg (2007)
6. Cao, N., Wang, C., Li, M., Ren, K., Lou, W.: Privacy-preserving multi-keyword ranked search over encrypted cloud data. In: Proceedings of the 30th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, Shanghai, China, pp. 829–837 (April 2011)
7. Cash, D., Jarecki, S., Jutla, C., Krawczyk, H., Roşu, M.-C., Steiner, M.: Highly-scalable searchable symmetric encryption with support for boolean queries. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 353–373. Springer, Heidelberg (2013)
8. Chang, Y.-C., Mitzenmacher, M.: Privacy preserving keyword searches on remote encrypted data. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 442–455. Springer, Heidelberg (2005)
9. Di Crescenzo, G., Saraswat, V.: Public key encryption with searchable keywords based on jacobi symbols. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) INDOCRYPT 2007. LNCS, vol. 4859, pp. 282–296. Springer, Heidelberg (2007)
10. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, Alexandria, Virginia, USA, pp. 79–88 (November 2006)
11. Goh, E.J.: Secure indexes. IACR Cryptology ePrint Archive 2003, 216 (2003)
12. Goldreich, O.: Foundations of Cryptography. Basic Applications, vol. 2. Cambridge University Press (2004)
13. Golle, P., Staddon, J., Waters, B.: Secure conjunctive keyword search over encrypted data. In: Jakobsson, M., Yung, M., Zhou, J. (eds.) ACNS 2004. LNCS, vol. 3089, pp. 31–45. Springer, Heidelberg (2004)
14. Hwang, Y.-H., Lee, P.J.: Public key encryption with conjunctive keyword search and its extension to a multi-user system. In: Takagi, T., Okamoto, T., Okamoto, E., Okamoto, T. (eds.) Pairing 2007. LNCS, vol. 4575, pp. 2–22. Springer, Heidelberg (2007)
15. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: Proceedings of the 19th ACM Conference on Computer and Communications Security, Raleigh, North Carolina, USA, pp. 965–976 (October 2012)
16. van Liesdonk, P., Sedghi, S., Doumen, J., Hartel, P., Jonker, W.: Computationally efficient searchable symmetric encryption. In: Jonker, W., Petković, M. (eds.) SDM 2010. LNCS, vol. 6358, pp. 87–100. Springer, Heidelberg (2010)
17. Moataz, T., Shikfa, A.: Boolean symmetric searchable encryption. In: Proceedings of the 8th ACM Symposium on Information, Computer and Communications Security, Hangzhou, China, pp. 265–276 (May 2013)

18. Park, D.J., Kim, K., Lee, P.J.: Public key encryption with conjunctive field keyword search. In: Lim, C.H., Yung, M. (eds.) WISA 2004. LNCS, vol. 3325, pp. 73–86. Springer, Heidelberg (2005)
19. Ryu, E.K., Takagi, T.: Efficient conjunctive keyword-searchable encryption. In: Proceedings of the 21st International Conference on Advanced Information Networking and Applications, Niagara Falls, Canada, pp. 409–414 (May 2007)
20. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: Proceedings of the 21st IEEE Symposium on Security and Privacy, Berkeley, California, USA, pp. 44–55 (May 2000)
21. Sun, W., Wang, B., Cao, N., Li, M., Lou, W., Hou, Y.T., Li, H.: Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. In: Proceedings of the 8th ACM Symposium on Information, Computer and Communications Security, Hangzhou, China, pp. 71–82 (May 2013)
22. Wang, P., Wang, H., Pieprzyk, J.: An efficient scheme of common secure indices for conjunctive keyword-based retrieval on encrypted data. In: Chung, K.-I., Sohn, K., Yung, M. (eds.) WISA 2008. LNCS, vol. 5379, pp. 145–159. Springer, Heidelberg (2009)