Secure Cloud Storage Service with An Efficient *DOKS* Protocol

ZhengTao Jiang
Communication University of China
z.t.jiang@163.com

Ling Liu
Georgia Institute of Technology
lingliu@cc.gatech.edu

***Abstract*—Storage services based on public clouds provide customers with elastic storage and on-demand accessibility. However, moving data to remote cloud storage also raises privacy concerns. Cryptographic cloud storage and search over encrypted data have attracted attentions from both industry and academics. In this paper, we present a new approach to constructing efficient oblivious keyword search (*OKS*) protocol, which permits fast search (i.e., sub-linear time) and relatively short ciphertext, while providing provably strong privacy for both users and cloud storage service providers. Previous *OKS* protocols have ciphertext size linear in the number of keywords, which consume much storage space and relatively long searching time. We formally define a Disjunctively Oblivious Keyword Search (*DOKS*) protocol realizing oblivious keyword search with the ciphertext size constant in size of keywords, which is significantly less than that of previous *OKS* protocols. Our approach improves both the privacy and efficiency of existing *OKS* protocols. With *DOKS*, adversary cannot distinguish two search keywords submitted by users, and cannot know the relations between ciphertext of documents and search keywords. A search keyword cannot be reused by adversaries. Users can get the matching documents without revealing statistical information on search keywords.**

Keywords-Cloud storage; searchable encryption; privacy; provable security; oblivious keyword search; DOKS

I. INTRODUCTION

Cloud computing and its pay per use elastic pricing and utility model has made outsourcing storage and computing needs more attractive than ever. By moving computing and storage needs to the cloud, users can avoid the high cost of storage and computing infrastructure ownership and achieve availability and reliability at a relatively low cost. However, outsourcing storage and computing to a public cloud infrastructure also faces some new challenges since users and cloud storage providers (either IaaS or SaaS like databases) are not located in the same trust domain. Thus, both data privacy and access privacy must be maintained as a part of service level agreement (SLA) with high level of guarantee. This makes security and privacy of outsourced data and private information retrieval one of the biggest challenges for outsourcing to cloud storage services.

A. Requirements for Secure Outsoutcing

There are two important challenges in secure outsourcing. First, the stored data must be protected against unauthorized access. Second, both the data and the access to data need to be protected from cloud storage service providers (e.g., cloud system administrators). In these scenarios, relying on password and other access control mechanisms is insufficient. Cryptographic encryption mechanisms are typically employed. However, simply having encryption and decryption implemented in the cloud database systems is insufficient. In order to support both challenges, data should be encrypted first by users before it is outsourced to a remote cloud storage service and both data security and data access privacy should be protected such that cloud storage service providers have no abilities to decrypt the data, and when the user wants to search some parts of the whole data, the cloud storage system will provide the accessibility without knowing what the portion of the encrypted data returned to the user is about.

In summary, a cloud storage service should meet the following three security and privacy requirements:

(a) General data security: The data should be securely stored in database hosted by the cloud storage service such that any unauthorized users cannot access it;

(b) Database security: A user is allowed to retrieve some data by keyword search techniques, but the user cannot get more content than the searching result;

(c) User query privacy: The user's query preference may be sensitive, and the cloud storage provider and its database server should not learn any useful information about which search keyword was submitted by the user and which data has been obtained by the user.

In addition to meeting the security and privacy requirements outlined above, the cloud storage service should continue to honor the generally accepted service level agreements (SLAs). That is, the cloud storage service should provide high computation and communication efficiency and support query-based access to allow users to selectively and privately retrieve any desired segment of the whole data on demand. Finding a good security-functionality tradeoff for outsourcing is a challenging research problem, which has received a great deal of attention recently [1,2].

Cryptographic storage techniques are widely recognized as an approach that holds the potential to meet the above requirements. The main advantage of cryptographic storage services is that its security properties are derived from cryptography, as opposed to legal mechanisms, physical security or access control, and can be proved in a formal manner. A simple solution for secure cloud storage is to encrypt the whole data and then store it in a database. To query any part of the data, one must download the whole encrypted data for decryption. Its computation and communication complexity is high, and it fails to meet the database security and user query privacy requirements [3].

Searchable encryption schemes are designed to efficiently solve security problems for remote cryptographic storage while enabling search for the expected contents corresponding to an encrypted keyword securely. The area of

searchable encryption has been identified by DARPA as one of the technical advances that can be used to balance the need for both privacy and national security in information aggregation systems. It also provides value-added features to many business services, such as Google Desktop with the ability of searching a client's data across several computers without sacrificing the client's privacy.

B. Searchable Encryption Techniques for Cloud Storage

Searchable encryption techniques are commonly used to efficiently meet the above requirements. There are several types of searchable encryption schemes in the literature, each of which is appropriate to a particular application scenario.

Symmetric searchable encryption (SSE) scheme introduced in [4] is suitable for the setting where a party searching over the data is also the one who generates it. Such scenario is referred to as single writer and single reader (SW/SR) [6].

Asymmetric searchable encryption (ASE) is designed for the scenario where a party searching over the data can be different from the party who generates it [5]. Such scenario is referred to as many writers and single reader (MW/SR) [6]. Since writers and readers can be different, ASE schemes are more suitable for the setting with a larger number of users.

Both SSE and ASE protocols did not completely solve the problem that one can *privately* retrieve segments of encrypted data from remote databases. Since the database server can learn by passive logging with statistical inference which encrypted keyword matches the submitted search keyword and which encrypted document is retrieved.

Oblivious keyword search (*OKS*) protocols are aiming at realizing the searching capabilities of searchable encryption (SSE, ASE) protocols while preserving privacy of both writers (requirement (b)) and readers (requirement (c)) in a strong version, which is not realized by SSE or ASE. The notion of *OKS* protocol was first introduced in [7], based on the assumption that remote storage service providers and users do not trust each other absolutely, and one party may try to learn sensitive information of the other party when conducting transactions. *OKS* hides statistical information on search keywords by not leaking keyword match results to databases or eavesdroppers.

However, to improve the practical applicability of *OKS* protocols, the following two issues need to be addressed: (i) *Inefficient communication, computation and storage.* In previous *OKS* protocols, each keyword is used to generate an encryption key, which is then used to encrypt the documents, so the number of ciphertexts and encryption keys to be maintained is equal to the number of keywords to be used for search over the encrypted documents. Usually the number of keywords is large, these protocols cost large storage space, high server computation time and high communication bandwidth. (ii) *Strong security guarantee*. To achieve a confident level of *OKS* security in a provable manner, proper formal security models of characterizing *OKS* attacks and for classifying common behaviors of adversaries are needed.

C. Scope and Contribution of the Paper

In this paper, we attempt to address the above two challenges by designing a new approach to constructing efficient oblivious keyword search (*OKS*) protocol, which permits fast search (i.e., sub-linear time) and relatively short ciphertext, while providing provably strong privacy for both users and cloud storage service providers. The smaller-ciphertext-size property is achieved by corresponding one ciphertext to a keyword set. The provably-strong-privacy property is achieved by reducing adversary's ability of decrypting and distinguishing search keywords to discrete logarithm problem (*DLP*) and Diffie-Hellman (*DDH*) problem respectively.

Concretely, previous *OKS* protocols have ciphertext size linear in the number of keywords, which consume much storage space and relatively long searching time. We formally define a Disjunctively Oblivious Keyword Search (*DOKS*) protocol, which realizes oblivious keyword search with the ciphertext size constant in size of keywords, significantly less than that of previous *OKS* protocols. Our approach improves both the privacy of and efficiency of existing *OKS* protocols. We show that *DOKS* is provably secure against adaptive chosen keyword attack (CKA) in random oracle (RO) model, which overcomes security flaws occurred in previous *OKS* protocols. With *DOKS*, any adversary cannot know the relations between ciphertext of documents and search keywords. Furthermore, a search keyword can not be reused by an adversary and users can get the matching documents without revealing any statistical information on search keywords.

The *DOKS* protocol has many possible applications. For example, a user Alice wants to search for some documents provided by other users or organizations from which Alice obtained authorization. *DOKS* will ensure that the search preference of Alice and other unrelated documents are perfectly protected. This type of secure search over remote storage systems can be useful for electronic health records (EHR) systems, in which a patient (or a physician) wants to search for sensitive treatment information about the patient(s) for a particular disease diagnosis from EHR databases hosted in some third party storage service providers.

II. Overview & Problem Statement

In this section we give a brief overview of existing searchable encryption protocols and present the problem statement for the *DOKS* protocol development.

A. *Basic Searchable Encryption Protocols*

There are two basic types of searchable encryption protocols: one is based on symmetric encryption schemes, and the other is based on asymmetric (public key) encryption schemes.

1) *Symetric Searhable Encyrption (SSE)*

The first construction of SSE scheme is proposed in [4]. It works as follows. Given a set of documents to be outsourced to a cloud storage service, each document is modeled as a sequence of words, represented by (w_1, w_2,… w_i,…, w_l). *SSE* can be used by a user *U* to encrypt each

document word by word such that the ciphertext of all words are stored in the remote database server sequentially. *U* can search for those documents that contain a search word *w* if its ciphertext matches one of the stored cipher blocks for the documents in the remote document database. Below we describe how the encryption and search are performed: First, a user *U* generates an encryption key k'' for a keyword w_i ($i = 1, 2, \ldots, l$), and gets

$$X_i = E_{k''}(w_i) =< L_i, R_i >,$$

where X_i has *n* bits, L_i and R_i denote the first *n-m* bits and the last *m* bits of X_i respectively.

U generates an encryption key k' for matching verification, and computes $k_i = f_{k'}(L_i)$. Then *U* generates a random number S_i and computes $T_i = S_i \,||\, F_{k_i}(S_i)$. The final ciphertext for w_i is $C_i = X_i \oplus T_i$. C_i is stored in the database server *DS*.

To search for documents containing a keyword *w*, *U* computes $X = E_{k''}(w)$ and $k = f_{k'}(L)$, and sends <*X*, *k*> to *DS*. This allows *DS* to search for *w* without revealing *w* itself. Any ciphertext stored in the database server *DS* is composed of cipher blocks, C_i, ($i = 1, 2, \ldots N$). *DS* sequentially computes $S_i||F_i = C_i \oplus X_i$, and $F = F_k(S_i)$.

If $F = F_i$, it means that *w* is contained in this document with high probability.

To decrypt the document, *U* generates S_i using pseudorandom generator (since *U* knows the seed) sequentially, then he recovers L_i by XORing S_i with the first *n-m* bits of C_i. Knowing L_i allows *U* to compute k_i and get $X_i = E_{k''}(w_i)$. Finally, *U* gets w_i from $E_{k''}(w_i)$ based on k''.

Since *SSE* is based on symmetric encryption schemes, and encryption of the search words and the corresponding matching documents is deterministic under one encryption key. *SSE* has the following limitations:

(1) It is only suitable for single-user environment in which a user who generates the ciphertext for storage is the same user who submits a searching task.

(2) Several secrete keys must be stored on the user side.

U must store at least three secret keys: (i) k'' for encrypting/decrypting words $w_1, \ldots, w_l$ and search word *w*; (ii) k' for calculating the verification function $f_{k'}(L)$; (iii) Seeds for generating random verification (and masking) values $S_1, \ldots, S_l$.

(3) Low privacy protection for users.

Since $X = E_{k''}(w)$ and for efficiency consideration, k'' is not changed according to each word of the documents to be encrypted, it cannot hide statistical information of words very well [4]. The *DS* server can get exact positions and frequency of search words in any document, even though he cannot learn the exact content. Furthermore, if a newly encrypted document is added to the database, the server can check whether it contains a particular keyword (as well as its exact positions, frequencies, etc.) that has been submitted by users in the previous queries.

(4) Low distinguishability guarantee for documents.

After receiving search words, *DS* can distinguish two documents by checking the inclusion or the positions of the encrypted data being queried by search words.

(5) Keyword encryption is of coarse granularity.

Since documents are encrypted word by word sequentially by stream cipher, if two words needs to be exchanged, some new words need to be added or some existing words need to be removed in a document, the whole document must be re-encrypted.

These limitations motivate the development of ASE.

2) Asymetric Searchable Encryption (ASE)

Motivated by the limitations of *SSE*, [5] introduced the asymetric searchable encryption (*ASE*) protocol using public key encryption. For example, Bob sends encrypted emails to Alice using Alice's public key. Both the email contents and the keywords are encrypted.The mail gateway cannot know the keywords and hence cannot make routing decisions or implement searching for customers. Their goal is to enable Alice to give the gateway the ability to test whether a keyword is included in the email, but the gateway should learn nothing else about the email. Another advantage for *ASE* is that it is suitable for the setting where the party search over encrypted data is different from the party who encrypts the data. Concretely, *ASE* works as follows.

(1) To send a message *M* to Alice with keywords $w_1, \ldots, w_m$, Bob sends

$$E_{A_{pub}}(M) \,||\, ASE(A_{pub}, w_1) \,||\, \cdots \,||\, ASE(A_{pub}, w_m)$$

to the gateway, where each keyword w_i is encrypted under Alice's public key A_{pub}.

(2) Alice sends the gateway a certain token T_{w^*} that enables the gateway to test whether one of the keywords associated with the message is equal to the search word *w* submitted by Alice. Namely, Given $ASE(A_{pub}, w)$ and T_{w^*} the gateway can test whether $w^* = w$.

ASE protocols are more suitable for multi-user setting than SSE.

Even though the encryption $ASE(A_{pub}, w_i)$ is probabilistic for w_i whereas the ciphertext of *SSE* is deterministic, *ASE* still has some security flaws for preserving privacy.

1) Leaking frequency of search words.

The token T_w is deterministic on *w* and the matching result is known to the gateway server. It knows the exact frequency of keywords queried by the users, and knows how many documents contain such keyword, which may suffer from dictionary attack.

2) Adversaries can check whether a new document contains a certain keyword queries before.

Without authorization, any adversary (including the server) can check whether a new document contains a certain keyword that has been queried.

Another limitation of *ASE* is that it only focuses on encryption of keywords and does not provide a concrete method on encrypting documents.

To overcome the limitations of both *SSE* and *ASE*, [6] designed *OKS* protocols by hiding statistical information and providing stronger privacy protection for both users and remote database servers [7,8]. *OKS* is applicable for the setting where one party uploads its encrypted data and many authorized users can download the portions of the data containing particular search keywords in an oblivious and yet more efficient manner. We refer to such scenario as single

writer/many reader (SW/MR), which differs from the setting of *SSE* (SW/SR) and ASE (MW/SR).

B. Oblivious Keywork Search (OKS) Protocols

In *OKS*, a database server possesses sensitive documents. It allows a user to search and retrieve documents containing some keywords chosen by users in an oblivious manner such that both user query privacy and the database security are guaranteed. By user privacy, we mean that both adversaries and database servers will learn nothing about the keywords submitted by users and which documents has been retrieved. By database security, we mean that a user can only get the documents that he has searched for, and cannot learn any more information on other documents. We below formally define these concepts.

Definition 1 (Correctness for *OKS*) Let $Search(w_i^*)$ denote the real search result which is a set of unencrypted documents in the database server *DS* contain a search word w_i^*. After running the *OKS* protocol on the user's input w_i^*, if the database *DS* also outputs the same search results as $Search(w_i^*)$, we say that the *OKS* is correct.

Definition 2 (User Privacy in *OKS*) An *OKS* protocol is secure for a user, if for any malicious provider *DS*, the view of *DS* for two keyword strings $(w_1^*,\ldots,w_k^*)$ and $(w_1',\ldots,w_k')$ is computationally indistinguishable when the following holds: $(w_1^*,\ldots,w_k^*) \neq (w_1',\ldots,w_k')$.

Definition 3 (Database Security in *OKS*) An *OKS* protocol is secure for a database, if the user can only get her searching result $Search(w_1^*),\ldots,Search(w_k^*)$, and cannot get any more useful information about other documents with non-negligible advantage.

Definition 4 (*OKS* Security) An *OKS* protocol is secure if it satisfies both user security and database security.

OKS protocols are two party protocols between a database and a user, which performs encryption and search over encrypted data with user privacy and database security guarantee through a two-phase process: commit phase and transfer phase [7].

In commit phase, the database server *DS* has n data blocks, $B_1, \ldots, B_n$ such that

$$B_i = (w_i, c_i),$$

where c_i denotes a document or data content to be encrypted and $w_i \in W$ (W is the keyword space) is the corresponding keyword that will be used to search over encrypted content. The remote database server *DS* commits the ciphertexts $C_1, \ldots, C_n$, where $C_i = Enc(k_i, c_i)$ and the encryption key k_i is generated from its corresponding keyword w_i.

At each subphase, U chooses a keyword $w \in W$ and then initiates a key generation protocol (KGP) with database server *DS*. U gets the decryption key for ciphertexts including w in an oblivious manner.

Based on KGP, U learns $Search(w)$, where $Search(w) = \{(i, c_i) \mid w_i = w\}$ is the set of all documents containing w as a keyword. However, U learns nothing more than $Search(w)$ and *DS* gains no information on w.

Presently, there are mainly two ways to realize *OKS*: One is based on blind signatures, and the other is based on oblivious polynomial evaluation (*OPE*).

Comparing with general keyword search encryption schemes, *OKS* protocols have advantages of preserving privacy for both user and database server. However *OKS* protocol also introduces disadvantages, such as high communication and computation cost, larger storage space to store ciphertext, since a large number of ciphertexts of documents are generated (see next subsection for detail.). One of the challenges for *OKS* is to reduce the size of ciphertexs and its implementation cost, while preserving provable privacy for both parties (user and database).

D. Problem Statement and Overview of DOKS

The idea of protecting privacy of user access and ensuring database security by using secure computation has been studied by many researchers. According to the types of cryptographic primitives utilized in literature, *OKS* fall into the following two categories: *OKS* constructed from *OPE* and *OKS* from *RSA*.

To illustrate the problems of existing *OKS* protocols and motivate the development of *DOKS*, we assume that the document database consists of a set of documents, and U wants to retrieve some of them according to particular keywords. Thus we below classify the problem of keyword search over documents into four scenarios:

Case 1 (1:1). Each document has only one keyword. Namely, there is one-to-one relationship between keywords and documents.

Case 2 (1:*n*). There are m keywords, and each keyword associates with n documents. The relationship between keywords and documents is one-to-many.

Case 3 (*m*:1). There are n documents, and each document includes m keywords. The relationship between keywords and documents is many-to-one.

Case 4 (*m*:*n*). There are n documents and m keywords, and each document includes m keywords, each keyword associates with n documents. The relationship between keywords and documents is many-to-many.

Previous *OKS* protocols are only suitable for Case 1 and Case 2, in which each keyword is used to generate one encryption key for the associated document [7]. If a document consists of m keywords (e. g., Case 3, Case 4), then m copies of cipherctexts must be generated by m encryption keys respectively. Therefore, the number of ciphertext for each document is equal to the number of keywords it contains. If the number of keywords is large, the size of ciphertex is also large.

In Case 3 and Case 4, each document has multiple keywords. For example, the document Doc_i has m keywords, represented by the keyword set $KSet(Doc_i) = \{KW_{i1}, .., KW_{mi}\}$. Using existing *OKS* protocols for Case 3 and Case 4, one must encrypt Doc_i m times, one per search word, with encryption keys generated by $KW_{i1}, .., KW_{mi}$ respectively. All the m ciphertexts for Doc_i must be computed, stored and delivered to the users who need to search over the encrypted document collection. Since the number of keywords is often

large, these protocols are highly prohibitive for Case 3 and Case 4. Here are a list of problems faced by existing *OKS*:

Problem 1. Computation explosion can be caused by large set of keywords. If each document has a large keyword set, say on the order of thousands or more, then a large number of keys and ciphertexts must be generated and the documents need to be encrypted multiple times. When a new keyword is added to a document, one more ciphertext corresponding to this keyword must be generated.

Probelem 2. Storage space explosion can be caused by large duplications of encrypted versions of the original documents. If a document contains *m* keywords, then it must be encrypted by *m* copies, using one of the *m* encryption keys, one per keyword, which consumes huge storage spaces and search cost when the number of documents is large and the number of search words per documents is large.

Problem 3. Communication explosion can be caused by large number of ciphertexts transferring between the remote database server *DS* and each user *U*. A large number of copies of ciphertexts need to be transferred from database to users for each query service request, which consumes high network I/O bandwidth.

Problem 4. It lacks of formal security models to characterize and manage *OKS* attacks for case 3 and case 4 scenarios.

E. DOKS: Design Ideas and Main Contributions

The design of our *DOKS* protocol aims at addressing the above challenges simultaneously by designing a new efficient *OKS* protocol.

DOKS is more efficient than existing *OKS* protocols, especially for the case 3 and case 4 scenarios. More specifically, *DOKS* needs only *n* ciphertexts to be transferred, while previous *OKS*s transfer $|KSet_1| + \ldots + |KSet_n|$ ($O(mn)$) ciphertexts, where $KSet_i$ is the set of keyword of document$_i$ ($i = 1, 2, \ldots, n$), *n* is the number of document, *m* is the average number of keywords contained in each document. Thus, *DOKS* is significantly more efficient in terms of storage, computation and communication performance compared with previous *OKS* protocols, especially for those case 3 and case 4, thanks to its small number of ciphertexts. *DOKS* is applicable to single writer and multiple reader (SR/MR) environments, unlike *SEE*, which is constrained to only the SR/SW environment, and unlike *ASE*, which works for MW/SR environment (case 1 and case 2). To the best of our knowledge, this *DOKS* development is among the first endeavors on developing efficient and *DLP* based secure *OKS* protocols.

In *DOKS* we address the privacy property guarantee by using a strong formal security model. We show that *DOKS* is provably secure against adaptive chosen keyword attack (CKA) in RO model, which overcomes security flaws occurred in previous *OKS* protocols. The provably-strong privacy property is achieved by reducing adversary's attacking ability to *DLP* and *DDH* problem respectively. Generally speaking, cryptographic settings are deployed to support two types of encryption schemes: integer factorization problem (IFP) based RSA schemes or *DLP* based Diffie-Hellman schemes. To the best of our knowledge, all previous *OKS* protocols are constructed from RSA or OPE schemes. Thus it is valuable to design *DLP* based *OKS* protocols, which is suitable for cryptographic setting of discrete logarithm based encryption.

To prove the security guarantee of *DOKS*, the formal chosen keyword attack (CKA) model is introduced to characterize *OKS* attackers. *DOKS* provides strong provable privacy for both users and database service providers against CKA in RO model. This CKA model can also be extended to analyze security of other previous *OKS* protocols.

III. *DOKS* PROTOCOL BASED ON DL-ENCRYPTION

In this section we first introduce the basic definitions and security model for the *DOKS* protocol and then describe two phases of the *DOKS* protocol: encryption and upload phase, download and decryption phase.

A. Definitions and Security Models

The *DOKS* protocol is constructed using discrete logarithm problem (*DLP*) based Diffie-Hellman scheme. Thus before we introduce the *DOKS* protocol, we provide definition of some basic concepts.

Definition 5 (The discrete logarithm problem *DLP*) Given (g, X, Y, Z), where $g, y \in Z_p$, to find x such that

$$g^x = y \bmod p,$$

is called discrete logarithm problem (*DLP*).

The *DLP* assumes that there does not exist any polynomial-time algorithm that can solve *DLP* with non-negligible advantage.

Definition 6 (The Computational Diffie-Hellman *CDH*) Given (g, X, Y, Z), where $X = g^x \bmod p$ and $Y = g^y \bmod p$, without knowing x and y, to compute

$$Z = g^{xy} \bmod p,$$

is called computational Diffie-Hellman (*CDH*) promlem.

Similarly, *CDH* assumes that there exist no polynomial-time algorithms that can solve *CDH* problem with non-negligible advantage.

Definition 7 (The decisional Diffie-Hellman *DDH*) Given (g, X, Y, Z), where $g \in Z_p$, $X = g^x \bmod p$ and $Y = g^y \bmod p$, to decide whether

$$Z = g^{xy} \bmod p,$$

is called decisional Diffie-Hellman (*DDH*) problem.

DDH also assumes that there exist no polynomial-time algorithms that can solve *DDH* problem with non-negligible advantage.

Definition 8 (*DOKS*) A disjunctive oblivious keyword search protocol based on *DLP*, denoted by *DOKS*, consists of the following polynomial time algorithms:

- ***KeyGen*(1^K):** Generates system parameters $p, q \in Z_p$, such that $q \mid p - 1$, and g is of order q. User U's public/private key pair is pk/sk.
- ***Encode*(*KSet*):** Generates a random polynomial $P(x)$ in commitment phase, such that $P(h_w) = t$, where $h_w = H(w)$ is a hash function and $w \in KSet$, t is a random number, and *KSet* is the set of keyword associated with a document. Documents are encrypted based on the key material t in commitment phase.
- ***Blind*(w^*):** Blinds a search word w^* with U's public key.

- ***Encrypt***(*Blind*(w^*))**:** Insert decryption key materials into *Blind*(w^*), and get $c = Enc(Blind(w^*), pk)$.
- ***Decrypt*(*c*, *sk*)**:** Decrypts *c* with *sk* to get a decryption key according to w^*.
- ***Decode*(*E*)**:** Based on *Decrypt*(*c*, *sk*), decodes data generated in commitment phase.

The *DOKS* protocol relies on the following steps to encrypt data before uploading it to the remote cloud storage and then to enable search over encrypted data by keywords:

(1) The database server *S* encodes a set of keywords of each document to be a secret polynomial $P(x)$; then the document is encrypted based on parameters of $P(x)$.
(2) *U* submits a search word *w* to search for documents containing *w*.
(3) *U* initiates a blind key generation protocol and gets the decryption key.
(4) If a document contains *w* as a keyword, *U* can decrypt the corresponding ciphertext.

We have mentioned that *DOKS* protocols are secure against CKA in RO model. It provides strong provable privacy of both users and database service providers. Below we define the chosen keyword attack (CKA) model.

Definition 9 (CKA) Given a *DOKS* protocol, *DOKS* = (*KeyGen*, *Encode*, *Blind*, *Encrypt*, *Decrypt*, *Decode*), and the set of public parameters generated by *KeyGen*, the model of chosen keyword attack (CKA) works as follows.

In the attack game, the adversary *U´* interacts with the simulator *Sim*, who simulates database server, through queries in RO model.

(1) *Sim* generates *n* strings $E_1, ..., E_n$ in the commitment phase.

(2) *U´* queries *Sim* with the search words, and *Sim* simulates the database server as in the real world.

(3) *Sim* generates a challenging ciphertext *E*.

(4) *U´* and *Sim* repeat (2).

(5) At the end of the attack game, *U´* outputs the plaintext *c* of ciphertext *E*.

U´ wins the attack game if *c* is a valid plaintext of *E*. The advantage of an adversary is defined as the probability it wins the game. An adversary is said to be (ε, *t*, q_w)-attacker, if it has advantage at least ε in the above game, runs in time at most *t*, and make at most q_w hash queries, where $\varepsilon \in [0, 1]$ is a real number.

Definition 10 (CKA security) An *OKS* scheme is said to be (ε, *t*, q_w)-secure in the sense of chosen keyword attack (CKA), if no (ε, *t*, q_w)-attacker exists.

B. DOKS Two Phase Protocol

DOKS protocol is a two-phase protocol with upload phase and download phase. The upload phase is also called commit phase. The data owner (writer) will first encode and encrypt all the documents before uploading them to the cloud storage. A commitment processor (CP) is used to carry out this task by following the phase I of the protocol. The second phase is called download phase or transfer phase, in which a reader *U* who is authorized by the writer can query over the encrypted data hosted in the cloud database server by having a download processor (DP) sending blinded search word(s) to the remote database server *DS*. *U* can download and decrypts the matching ciphertexts. DP will carry out this task by following the phase II of the protocol, which gets decryption keys for particular documents obliviously from CP based on a blinded search word. DP also generates public/secret key pairs for user *U*.

Let $D = \{(w_{11}, w_{12}, \ldots, w_{1m}; c_1), \ldots, (w_{n1}, w_{n2}, \ldots, w_{nm}; c_n)\}$ denote the collection of *n* documents to be uploaded to the cloud database, where *m* is the rank of keyword field, c_i is the data to be searched for, and $w_{i1}, w_{i2}, \ldots, w_{im}$ are the *m* keywords corresponding to c_i ($1 \le i \le n$). Thus *D* contains a total of *mn* search words.

The concrete *DOKS* protocol works as follows:

System public parameter (KeyGen): Generate two large prime integers *p* and *q*, such that $q \mid p\text{-}1$. $g \in Z_q^*$ is of order *q*. *G* is a pseudo-random generator. $Y = g^{\mu} \bmod p$ is *U*'s *public* key, and μ is *U*'s private key.

Input: CP: $D = \{(KSet_i; c_i)\}_{i \in [n]}$, where $KSet_i = (w_{i1}, w_{i2}, \ldots, w_{im})$, and w_{ij} are not necessarily distinct ($1 \le i \le n$, $1 \le j \le m$), $[n]$ denotes the set $\{1, 2, \ldots, n\}$; DP: a search word *w*.

Output: DP: c_i, if $w \in KSet_i$; nothing otherwise.

(1) Upload and Data Encryption

At Upload Phase (Commit Phase), CP encodes keywords, encrypts corresponding documents as ciphertexts and attaches some metadata (e.g., keywords, types, access models, etc), and commits ciphertext of documents and metadata to the cloud storage server. CP encodes documents as polynomials using a pseudorandom generator.

Commit Phase (Encode):

The CP performs three tasks. First, CP chooses two random numbers r_i, $t_i \in Z_q^*$ for each document $D_i = (w_{i1}, w_{i2}, \ldots, w_{im}; c_i)$, $i=1,\ldots,n$; and then computes hash value $H(w_{ij}) = x_{ij}$, then D_i is encoded to be a polynomial:

$$P_i(x) = r_i \prod_{j=1}^{m} (x - x_{ij}) + t_i = \sum_{j=0}^{m} a_{ij} x^j .$$

Therefore, each keyword is one root of $P_i(x) - t_i = 0$, and can be used as a token to generate the decryption key.

Second, CP uses a pseudorandom function *G* to encrypt the document content ($c_i \| 0^l$) in an XOR way:

$$E_i = G(T_i \| i) \oplus (c_i \| 0^l),$$

where $T_i = g^{t_i} \bmod p$, $\|$ denotes concatenation, 0^l is a *l*-bit string. The 0-sequence checks validity of the content c_i.

Third, CP uploads the ciphertexts of the documents, $E_1, E_2, \ldots, E_n$ to the cloud database server *DS*. Without knowing the key T_i, any adversary cannot get c_i.

(2) Data Download and Decryption

At Download Phase (Transfer Phase), user *U* can get sensitive documents containing keyword *w*, without letting *DS* know what he is downloading.

U (DP) blinds *w* with his public key, ***Blind***(*w*), and sends it to the server, who knows nothing about *w*.

DP generates *Enc*(***Blind***(*w*)) by inserting decryption key information into ***Blind***(*w*).

On getting *Enc*(***Blind***(*w*)), Alice unblinds *Enc*(***Blind***(*w*)) and gets the corresponding key to *w*. Then Alice can decrypt documents what he wanted.

ElGamal encryption mechanism is used as the blinding function. The probabilistic encryption and homomorphism property of ElGamal scheme will be used to ensure semantic security for search word *w* in the following form:

$$Enc = y^k g^{h_w} \bmod p,$$

in which *k* is random and *y* is the public key of user *U*, and $h_w = H(w)$.

Transfer phase:

U wants to search for documents associating with a keyword set, *Kset*, containing the keyword *w*. He invokes *DP*. *DP* first computes $h_w = H(w)$ and then carries out the search, download and decryption in the following 4 steps.

Step 1 (Blind). DP chooses *m* random integers $k_j \in Z_q^*$ ($1 \le j \le m$), and computes

$$K_j = g^{k_j} \bmod p,\ Enc_j = y^{k_j} g^{h_w^j} \bmod p.$$

Then sends $(K_1, Enc_1), \ldots, (K_m, Enc_m)$ to CP.

Step 2 (Encrypt). CP uses the homomorphism property of ElGamal encryption to compute

$$Enc(P_i(h_w)) = g^{a_{i0}} \prod_{j=1}^{m} (Enc_j)^{a_{ij}} = y^{\sum_{j=1}^{m} a_{ij} k_j} g^{P_i(h_w)} \bmod p,$$

and computes $A_{ij} = g^{a_{ij}} \bmod p$, $j = 0, 1, \ldots, m$.

CP sends $Enc(P_1(h_w)), \ldots, Enc(P_n(h_w))$ and A_{ij} ($1 \le i \le n$, $0 \le j \le m$) to DP. This step aims at hiding statistical information on search words to ensure user access privacy.

Step 3 (Decrypt). DP has *U*'s private key μ and knows k_j, $j = 1, 2, .., m$, so it can use ElGamal decryption method to recover $T_i' = g^{P_i(h_w)} \bmod p$, $i = 1, 2, \ldots, n$.

Step 4 (Decode). Initially, let $T = \emptyset$. For $i = 1, 2, \ldots, n$, *U* computes $(a_i \| b_i) = E_i \oplus G(T_i \| i)$.

If $b_i = 0^l$, then DP succeeds and adds (w, a_i) to *T*. Otherwise, DP outputs a failure message.

Finally, DP has $T = Search(w)$ as the searching result, in which the keyword set of each document containing *w*.

IV. DOKS Feasibility Analysis

We have presented the design of *DOKS* protocol. One of the important developments of *DOKS* protocol is to formally prove its correctness, user security and database security. However, due to the space constraint, in this section we give an informal discussion on the *DOKS* feasibility analysis and we refer readers to our technical report [3] for further detail.

For correctness analysis, we want to prove that there is a high probability that *DOKS* will return accurate and complete set of the results of a keyword search. Suppose that there are *n* ciphertexts and *l* denote the number of zero in the 0-sequence which checks the validity of the decoding process in step 4. After running *DOKS* protocol, the probability that one can get the final searching result $Search(w)$ is at least $1 - n2^{-l}$.

Consider two cases: (i) If the search word *w* submitted by a user *U* is equal to a keyword belonging to the keyword set $KSet_i$ ($1 \le j \le m$), then *U* can get the correct key information and the correct decryption key for documents. (ii) In the case that the search word *w* is not equal to any keyword of the keyword set $KSet_i$ ($1 \le j \le m$), then the probability that *U* can get the correct decryption key for documents is no higher than 2^{-l}.

In addition to correctness, we also need to formally prove that *DOKS* preserves desired user's security and database security.

To achieve User's security, *DOKS* protocol should prevent any adversary, including database server *S* from getting any useful information on the keyword hidden in the ciphertext. Concretely, in step 1 of *DOKS* protocol, if an adversary wants to distinguish two keywords, he will run into the problem of semantic security of ElGamal encryption scheme, which is intractable.

We next analyze the database's security in RO model by assuming that *DLP* is hard.

Suppose that there is a game between the simulator *Sim* and an adversary. *Sim* simulates the encryption ability of the server by encrypting keyword indexes and generating ciphertexts in RO model. Based on the intractability of *DLP*, it can prove that the probability that an adversary outputs the correct plaintext is negligible.

Based on the assumption of *DLP*, a malicious user cannot get any extra useful information on other documents in the proposed *DOKS* protocol. In fact, by assuming that *DLP* is hard, for a *DLP*-challenge instance, the ability of recovering other plaintext will be reduced to guess the output value of random oracle, which is negligible.

At Query Phase 1:

– *Sim* acts as *S* to "encode" keywords as polynomials;

– *U*′ submits a polynomial number of search words for querying;

– *Sim* trains *U*′'s attack ability by answering the queries correctly in RO model.

At Challenge Phase 2:

– *U*′ is given a challenging ciphertext to extract plaintext indexed by a keywords set.

At Query Phase 3:

– *U*′ and *Sim* repeat Query Phase 1.

At Query Phase 4:

U′ outputs the plaintext of the challenging ciphertext in Phase 2.

Furthermore, based on the assumption that *DLP* is hard, the *U*′'s attack ability is reduced to guessing a random value, which is negligible. □

Since a set of keywords is encoded into only one encryption key in *DOKS*, *DOKS* only needs to maintain *n* ciphertexts for the same dataset to be outsourced to a cloud storage, instead of generating and storing *mn* ciphertexts, which is expensive in terms of storage, computation and communication, especially when the number of keyword (*m*) and the number of ciphertexts (*n*) are large. Furthermore, users do not have to download, decrypt and verify *mn* ciphertexts. Therefore, even though the per encryption cost in *DOKS* is slightly higher than existing *OKS* protocols, with the reduction on the number of encryption/ decryptions needed from *mn* to *n*, *DOKS* protocol consumes significantly

less computation and transmission for both encrypting and uploading the outsourced datasets as well as downloading and decrypting query results, while offering higher user access privacy and database security.

In short, *DOKS* needs to transfer $m(n+2)|p|$ bits, and the total computation is about $(2n+6m+1)$ module exponentiations (E) and $m(n+1)$ module multiplications (M), while the *OKS* protocol in [7] needs more than $(mn+2)$ E; and the *OKS* protocol in [9] needs more than $(2mn+n+1)$ E. *DOKS* needs to store with n ciphertexts, while [7], [8] and [9] stores mn ciphertexts.

In the application of remote storage, some new keywords may need to be added to the database and some keywords may to be deleted from the database.

1) Keywords Addition

Suppose that a new keyword w is added to the document c_i, then its corresponding polynomial generated in commit phase should be changed into

$$P_i'(x) = r_i \prod_{j=1}^{m}(x - x_{ij})(x - h(w)_w) + t_i = \sum_{j=0}^{m+1} a_{ij}' x^j$$

where $h_w = h(w)$.

Since the parameters r_i, t_i are not changed, odd users do not need to initiates new implementation of the full *DOKS* protocol.

2) Keywords Deletion

When a keyword w needs to be deleted from a document c_i, the symmetric key $G(T_i||i)$ of $E_i = G(T_i||i) \oplus (c_i||0^l)$ must be changed. The database server S chooses new parameters r_i, t_i for $P_i(x)$, and the procedure of transfer phase is not changed.

3) Multi-User Setting

Since the generation of polynomial $P(x)$ and document encryption do not depend on any users' key, *DOKS* protocol supports multi-user settings, namely SW/MR, which is different from SW/SR (SSE) or MW/SR (ASE).

V. Related Work

The cryptographic storage services have gained active attention recently [6]. The main component for search over encrypted data includes the searchable encryption (SSE, ASE and Multi-user SSE), the attribute-based encryption, and proofs of storage. [10] has presented two solutions to design more efficient SSE, both of them offer more efficiency and stronger security (adaptive SSE security) in a multi-user setting. Their first construction is efficient non-adaptive SSE scheme in terms of computation on the server, and incurs a minimal cost for the user. Their second construction achieves adaptive security. As we discussed in the introduction section, both SSE and ASE have some limitations: while they are proven to be a secure encryption scheme, it is not proven to be a strongly secure searchable encryption scheme; the distribution of the underlying plaintexts is vulnerable to statistical attacks [5]. Recently a public-key encryption scheme is proposed [11] to hide the access patterns. However, it has an overhead in search time that is proportional to the square root of the database size, which is far less efficient than SSE[12]. Oblivious keyword search (*OKS*) protocols [6,8] present alternative approaches to address the privacy and security of access patterns. However, as analyzed in Section 2, we have shown the inefficiency and weaker security of existing *OKS* protocols.

VI. Conclusions and Future Works

This paper investigated new approaches for constructing an efficient *OKS* protocol from *DLP*. Since all previous *OKS* protocols are based on RSA or *OPE* problems, *DOKS* is suitable for new security parameter settings. Formal *DOKS* protocol and CKA model are initially defined to achieve better performance and provably strong privacy. The ciphertext size of *DOKS* is independent of the number of keywords, leading to better performance than previous *OKS* protocols in terms of the cost of communication, computation and storage space. Other significant advantages of *DOKS* include: semantic security for search words, full query isolation from documents, controlled search preventing search words from reusing, hiding statistical information on queries.

Acknowledgement. This work is partially sponsored by grants from NSF NetSE program, SaTC program, IBM faculty award and Intel ISTC on Cloud Computing. The first author thanks the support from NSF (61103199, 61003244, 61063041), BMNSF (4112052), Engineering Program Project of CUC, IERCPGP&ME (2012B091000060).

References

[1] C. Wang, K. Ren, Sh. Ch. Yu, et al. Achieving usable and privacy-assured similarity search over outsourced cloud data. INFOCOM, 2012.

[2] Y. Z. Tang, T. Wang, L. Liu, et al.. Privacy-preerving indexing for eHealth information network, Proceedings of 20th ACM CIKM, 2011.

[3] Z. T. Jiang, L. Liu. Practical *DOKS* protocols without ciphertext expansion for secure cloud storage. Technical Report, Feb. 2013, CERCS, Georgia Institute of Technology.

[4] D. X. Song, D. Wagner, A. Perrig. practical techniques for searches on encrypted data. Proceedings of the IEEE Symposium on Security and Privacy, 2000, pp. 44-55.

[5] D. Boneh, G. D. Crescenzo, R. Ostrovsky, G. Persiano. Public key encryption with keyword search. Advances in Cryptology-EUROCRYPT'04, 2004, LNCS 3027, Springer, pp. 506-522.

[6] S. Kamara, K. Lauter. Cryptographic cloud storage. The 14th international conference on Financial cryptograpy and data security, 2010, Springer-Verlag, pp. 136-149.

[7] W. Ogata, K. Kurosawa. Oblivious keyword search. Journal of Complexity, 2004, Vol. 20, Iss. 2-3, pp. 356-371.

[8] M. J. Freedman, Y. Ishai, B. Pinkas, et al. Keyword search and oblivious pseudorandom functions. Proceedings of the Second international conference on Theory of Cryptography- TCC'05, 2005, Springer-Verlag Berlin, pp. 303-324.

[9] H. S. Rhee, J. W. Byun, D. H. Lee, et al. Oblivious conjunctive keyword search. Proceedings of the 6th international conference on Information Security Applications-WISA'05, 2005, Springer-Verlag, Berlin, pp. 318-327.

[10] R. Curtmola, J. Garay, S. Kamara, et al. Searchable symmetric encryption: improved definitions and efficient constructions. Proceedings of the 13th ACM conference on Computer and communications security-CCS'06, 2006, pp. 79-88.

[11] D. Boneh, E. Kushilevitz, R. Ostrovsky, et al. Public-key encryption that allows PIR queries. Cryptology ePrint Archive: Report 2007/073.

[12] H. F. Zhu, F. Bao. Oblivious keyword search protocols in the public database model. ICC'07, 2007, pp. 1336-1341.