International Journal on Applications in Information and Communication Engineering
Volume 4: Issue 2: April 2018, pp 1 - 11. www.aetsjournal.com ISSN (Online) : 2394 - 6237

PRIVACY-PRESERVING STORAGE AND RETRIEVAL IN MULTIPLE CLOUDS

C. JAYASRI , S.JAYASURIYA , P.RAMYA , R.SANTHINI , SAVITHRI.R , C.ARUNKUMAR

Abstract— Cloud computing is growing exponentially, whereby there are now hundreds of cloud service providers (CSPs) of various sizes. While the cloud consumers may enjoy cheaper data storage and computation offered in this multi-cloud environment, they are also in face of more complicated reliability issues and privacy preservation problems of their outsourced data. Though searchable encryption allows users to encrypt their stored data while preserving some search capabilities, few efforts have sought to consider the reliability of the searchable encrypted data outsourced to the clouds.

In this paper, we propose a privacy-preserving STorage and REtrieval (STRE) mechanism that not only ensures security and privacy but also provides reliability guarantees for the outsourced searchable encrypted data. The STRE mechanism enables the cloud users to distribute and search their encrypted data across multiple independent clouds managed by different CSPs, and is robust even when a certain number of CSPs crash. Besides the reliability, STRE also offers the benefit of partially hidden search pattern. We evaluate the STRE mechanism on Amazon EC2 using a real world dataset and the results demonstrate both effectiveness and efficiency of our approach.

Keywords— searchability, privacy, and reliability

I. INTRODUCTION

Cloud computing is growing exponentially, whereby there are now hundreds of cloud service providers (CSPs) of various sizes [1]. A concept of a cloud-of-clouds (also called an intercloud) is proposed and studied in recent years [1], [2]. In a cloud-of-clouds, we disperse data, with a certain degree of redundancy, across multiple independent clouds managed by different vendors, such that the stored data can always be available even if a subset of clouds becomes inaccessible.

C. Jayasri , Final Year Cse, Meenakshi Ramaswamy Engineering College, Thathanur, Tamilnadu, India.
S.Jayasuriya , Final Year Cse, Meenakshi Ramaswamy Engineering College, Thathanur, Tamilnadu, India.
P.Ramya , Final Year Cse, Meenakshi Ramaswamy Engineering College, Thathanur, Tamilnadu, India.
R.Santhini , Final Year Cse, Meenakshi Ramaswamy Engineering College, Thathanur, Tamilnadu, India.
Savithri.R , Final Year Cse, Meenakshi Ramaswamy Engineering College, Thathanur, Tamilnadu, India.
C.Arunkumar , Assistant Professor, Meenakshi Ramaswamy Engineering College, Thathanur, Tamilnadu, India.

The multi-cloud environment [2] offers plenty of new opportunities and avenues to cloud consumers. Cloud consumers will be able to leverage not just one cloud provider, but many, to solve their diverse needs and switch providers if one ceases service. To promote the multiple clouds, IEEE has initiated Intercloud Testbed [1] that helps make interactions among multiple clouds a reality.

However, while cloud consumers may enjoy cheaper data storage and powerful computation capabilities offered by multiple clouds, consumers also face more complicated reliability issues and privacy preservation problems of their outsourced data. More specifically, as it is difficult to obtain clear guar- antees on the trustworthiness of each CSP [3], cloud con- sumers are typically suggested to adopt searchable encryption techniques [4] [5] to encrypt their outsourced data in a way that the encrypted data can be directly searched by the CSPs without decryption. Despite many efforts devoted to improving efficiency and security of the searchable encryption, there is little consideration on ensuring the reliability of the searchable encrypted data. Though cloud storage provides an on-demand remote backup solution, it inevitably raises dependability con- cerns related to having a single point of failure and to possible storage crash.

Existing reliability guarantees solely rely on each CSP's own backup solution, which however could be a single-point of failure. For instance, the crash of Amazon's elastic computing service in 2011 took some popular social media sites off- line for a day and one energy department collaboration site unavailable for nearly two days. More seriously, this crash has permanently destroyed many customers' data with serious consequences for some users [6]. It is worth noting that a com- prehensive solution to simultaneously ensuring *searchability, privacy, and reliability* on data outsourced to multiple clouds is not trivial to define. Simply replicating data at multiple CSPs is the most straightforward method, which however is the least cost-efficient approach. To the best of our knowledge, we are not aware of any existing work that addresses the three requirements in a comprehensive manner.

To address the aforementioned challenges, we propose a privacy-preserving STorage and REtrieval (STRE) mechanism that enables cloud users to distribute and search their encrypted data in CSPs residing in multiple clouds while obtaining reliability guarantees. We have designed efficient and secure multi-party protocols based on the secret sharing mechanism, to ensure that a user will be able to reconstruct the query results even if (n t) CSPs have been compromised, where *n* is the total number of CSPs storing the user's files and *t* is a threshold value predefined. Moreover, the STRE mechanism also offers better protection on the use's search pattern compared to existing works. Specifically, many existing works on searchable encryption would completely disclose the user's search pattern that indicates whether two searches are for the same query keyword or not [7][8]. In our STRE mechanism, this pattern leak risk is lowered because the search is conducted distributed and the search pattern will be revealed only if there are more than *t* CSPs collude. $\sum$

In this paper, we provide an in-depth security analysis, and present the results of our experimental study. We build $_{j=}$ a testbed in Amazon EC2 to simulate the multi-cloud environment and evaluate our mechanism using a real world dataset (i.e., the Enron dataset [9]). Our experimental results demonstrate the efficiency of our proposed approach. The rest of the paper is organized as follows. In Section II, we discuss some preliminary notions. In Section III, we present the system model and design goals. The proposed STRE mechanism is provided in Section IV. Security analysis and experimental results are respectively shown in Section V and Section VI. Section VII reviews the related works. Finally, Section VIII draws the conclusion of this paper.

II. PRELIMINARIES

For better understanding, we first give a brief review of the idea of searchable encryption, and then introduce the concept of secret sharing that forms our approach.

A. Searchable Encryption

Searchable encryption is a cryptographic primitive, which allows users to execute keyword-based search directly on encrypted data without decryption. Some scheme [4] implements the *searchability* via a special ciphertext that allows searching, while most other schemes [10], [8], [7], [11], [12] make the client generate a searchable encrypted index. Here, we briefly introduce the *generic* framework of searchable encryption, which will be followed by our mechanism.

Basically, a searchable encryption scheme includes four stages. Initially, a user *encrypts* a set of files into ciphertexts and a sophisticated *index*. Both file ciphertexts and index are uploaded to a remote server for storage. Later, when the user wants to retrieve the files containing some keyword, he/she generates a *trapdoor* from the keyword and sends the trapdoor (instead of the keyword itself) to the server for search. The server *searches* with the trapdoor and returns a set of ciphertexts, of which the underlying files contain the query keyword. Finally, the user *decrypts* these ciphertexts and obtains the plain files.

This work does not focus on optimizing the searchable encryption design; instead, we focus on how to introduce *reliability* into existing searchable encryption schemes [7].

B. Secret Sharing

A (n, t)-single secret sharing scheme [13] is a randomized protocol for the distribution of secret s among a set of n parties such that the recovery of secret is possible with at least t shares.

In this paper, we utilize single secret sharing in a special case, namely (n, n)-single secret sharing, which can be achieved in linear time complexity. Suppose a secret s is to be

$v = s - {}^{n\ 1}_{j=1}$

$v_1, \ldots, v_{n\ 1}$ are randomly chosen. We assign the share v_i to

ith party for $i = 1, \ldots, n$. In order to reconstruct the secret

s, n parties expose their shares and compute

$s = \sum_{j=}^{n} v_j$.

Note that the simplified (n, n)-single secret sharing scheme is *additive homomorphic*. Specifically, suppose s and s' are two secrets, and v_i and v_i' are respectively the ith (n, n)-single secret share of s and s'. Then $v_i + v_i'$ is the ith (n, n)-single secret share of $s + s'$.

Similar to the (n, n)-single secret sharing, a (n, t)-multiple secret sharing scheme requires that at least t or more parties can pool their secret shadows and reconstruct *multiple* secrets. In the paper, we use a *generic* multiple secret sharing scheme for sharing trapdoor, which can be instantiated with Bai's construction [14].

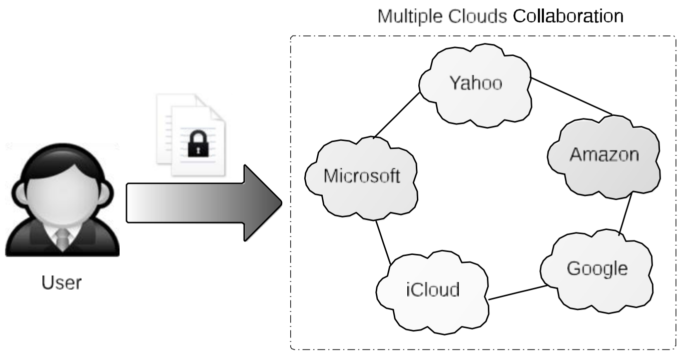


Figure 1. Multi-Cloud Architecture

III. PROBLEM FORMULATION

In this section, we introduce the system architecture, followed by defining the problem and presenting our security goals.

A. System Model and Problem Statement

In this work, we consider the cloud storage services offered in a multi-cloud environment, which involves two types of entities: i) *Users*, who store a large number of encrypted files in multiple clouds and execute keyword-based queries to access and manipulate their stored files; ii) *Cloud Service Providers* (CSPs), who possess storage and computation resources, are willing to cooperatively store and manage the users' files.

We focus on searchability of encrypted data, stored by users in one or many multi-cloud service providers. Informally, searchability (of encrypted data) refers to the ability of end users to retrieve encrypted files without having the CSP to decrypt it. These searches are typically carried out using keywords, which the client uses to locate the desired files.

We formalize this notion of general keyword search problem on plain files in multi-cloud environment.

Definition 1 (General keyword search): Given a user, let **f** be a collection of user files stored in a number of CSPs located in multiple clouds. A keyword search query (w_q) issued by the user retrieves these files (from the CSPs), that contain the query keyword w_q.

In the next subsection, we extend this general keyword search definition to the reliable and private keyword search after the introduction of our security goals.

B. Adversary Model and Design Goals

We consider a "honest-but-curious" adversary, which can compromise a tolerable number (t 1) of CSPs and attempt to learn from the information stored in the sites of the compromised CSPs.

Our design goals include the following objectives:

Reliability. Given n CSPs, the system should still function if at least t ($t < n$) CSPs are available, where t is a predefined threshold value for the system.

Semantic Security. The system should be semantically secure [7] by satisfying the following two requirements. First, given the file index and the collection of encrypted files, no adversary can learn any information about the original files except the file lengths. Second, given a set of trapdoors for a sequence of keyword queries, no adversary can learn any information about the original files except the access pattern (i.e., the identifiers of the files that contain the query keyword) and the search pattern (i.e., whether two searches are looking for the same keyword or not).

Trapdoor Security. We aim to achieve the *conditional* trapdoor security. Specifically, we require that any information about the query keyword *-including the search pattern-* should not be leaked before the multiple CSPs' collaborative search. The requirement holds even if at most (t 1) CSPs are compromised by adversary.

Robustness. We require that: i) when the protocol successfully completes, the correct files are returned and reconstructed by the users; ii) when the protocol aborts, even in the collaborative search stage, nothing is returned and CSPs learn nothing about the file collection or the underlying searched keyword.

Among the design goals above, *reliability* and *trapdoor security* are new features which have not been considered yet in existing works of symmetric searchable encryption [4], [12], [11], [7], [15], [8]. Therefore, in what follows, we give more details about the trapdoor security. We will elaborate on the reliability property in the next section when the protocols are introduced.

Trapdoor security is defined for protecting search pattern. Informally, search pattern is the information about whether any two queries are generated from the same keyword or not. Most known searchable encryption schemes [4], [12], [11], [7], [15], [8] allow to leak searchable pattern, and the limitation has recently been exploited for extracting file contents [16].

We observe that the search pattern may be leaked in two ways. First, if a CSP knows the access pattern, the information about the identifiers of the files that contain the query keyword, it will have a higher success rate in guessing whether two queries are looking for the same keyword by comparing the query results. In particular, after search, the CSP will know the identifiers of files that contain the query keywords if there is no extra prevention in place. If two queries return the same set of files, the CSP are likely to infer that these two queries contain the same keyword and reveal the user's search pattern. Second, the search pattern may also be leaked in that the searchable encryption schemes [4], [12], [11], [7], [15], [8] exploit *deterministic* trapdoor techniques and hence CSPs can learn whether two queries are for the same keyword or not. The CSPs can then store the deterministic trapdoors and the corresponding file identifiers to infer the content of the encrypted query keywords contained in incoming queries.

To address the issue of the deterministic trapdoor, *trapdoor security* aims at guaranteeing that *the only way for leaking search pattern will be restricted to the leakage of access pattern*. In this way, we can hide the search pattern before any collaborative search is carried out.

The formalization is based on a game virtually played between an adversary and a challenger with a security parameter. The adversary has the capability of compromising at most (t 1) CSPs and learning the information (e.g., ciphertexts, trapdoor shares) intended to these compromised CSPs. Whereas, the challenger plays the role of a legitimate user. Note that this game is an abstract formalization of a toy case or files storage and retrieval in case of two keywords, and it is useful to provide our security analysis later. Formally, the game is defined as follows.

Challenge. The adversary claims a collection of files for challenge. The challenger encrypts these files into a set of ciphertexts and an index. The challenger outputs the index. Note that since the relation between files and key- words is just embedded in the index and the encrypted file chunks are useless for adversary's keyword guess, we only provide adversary with the index for simplicity.

Query. The adversary generates two keywords w_0 and w_1, and submits a trapdoor query (, w_0, w_1) to the challenger. Notice that is a (t 1)-element set for simulating the CSPs being compromised. The challenger picks a random bit b, generates a set of trapdoor shares on w_b and just outputs the shares intended for the CSPs in C.

Guess. The adversary outputs b' as the guess of b.

We define the advantage of an adversary in this game as $\Pr[b' = b]$ – 1/2. Then, trapdoor security is defined below.

Definition 2 (Trapdoor security): A keyword search mechanism, specified according to Definition 1, satisfies trapdoor security if all probabilistic polynomial time adversaries have at most a negligible advantage in the above game.

Having defined the main security goals for our scheme, we revisit our Definition 1 as follows.

Definition 3 (Reliable and private keyword search): A reliable and private keyword search scheme in a multi-cloud environment is a keyword search (as specified in Definition 1), which satisfies reliability, semantic security, trapdoor security and robustness.

IV. STRE MECHANISM

In this section, we first provide an overview of our proposed STRE mechanism followed by two naive approaches. The naive approach is introduced to clarify the challenges in design- ing scheme for the aim of searchability and reliability. Then, we present the detailed protocols in the STRE mechanism.

A. Overview

The STRE mechanism consists of three major phases: the *Setup Phase*, *Storage Phase* and *Retrieval Phase*.

Setup Phase. The phase generates a master secret key from a security parameter and assigns the key to a user. Notice that the security parameter which is assumed to be known to all the adversaries, specifies the input size of the problem. Both the resource requirements of the cryptographic algorithm or protocol and the adversary's probability of breaking security are also expressed in terms of the security parameter.

Storage Phase. In the phase, a user takes input a collection of files and the master secret key, and generates a set of file shares and a file index. The file shares and index are uploaded to the corresponding CSP.

Retrieval Phase. This phase includes three steps: First, to search the files containing a certain keyword, the user generates a set of trapdoor shares based on the query keyword and his/her master secret key. The trapdoor share is sent to the respective CSP. Second, the CSPs col- laborate together to search with their individual trapdoor share, and response the search results back to the user.

Third, the user reconstructs and decrypts the results and obtains the clear files, each of which contains the query keyword.

B. Naive Approaches

We first describe two naive approaches and then specify their drawbacks. The first approach is trivial *replication*. In particular, we can replicate the single-user searchable encryption scheme to n CSPs, and each CSP stores the same searchable ciphertexts. Hence, even at most $n-1$ CSPs are unavailable, the remote files are still accessible with keyword search. However, the approach is undesirable, as it costs too much space for saving the redundant data.

For reducing redundancies, while tolerating CSP failures, a traditional solution is erasure coding. Specifically, we can encode the files into a set of shares and distribute the shares to n CSPs, so that even a tolerable number of CSPs are unavailable, the shares from the rest CSPs can be used for reconstructing the plain files. Based on the intuition, the second, *share*-based, approach is described as follows.

For each file, the user encrypts it and further encodes the ciphertext into a set of file shares. In addition, for the keyword related to the file (for simplicity we just assume a single keyword is related to the file), the user applies a one-way function on it. Recall that a one-way function is such that it is easy to compute it on every input but hard to invert given the image. This enables keyword privacy: an adversary cannot know the keyword even if it has recovered the corresponding trapdoor. The user then performs (n, t)-secret sharing on the trapdoor and obtains a set of trapdoor shares. Finally, both file and trapdoor shares are uploaded to respective CSP for storage. To retrieve the files containing a certain keyword, the user generates the trapdoor of the keyword, employs *another* (n, t)- secret sharing on the trapdoor and sends the *new* set of trapdoor shares to CSPs. Note that for an identical keyword, its trapdoor

shared in the storage and retrieval phases should be the same. This ensures that the CSPs are able to run multi-party protocols to perform "decision search" file by file through checking whether the shares received in the retrieval phase and those received in the storage phase are from the same secret. If so, they send the corresponding file shares back to the user, and the user is able to reconstruct and decrypt the results.

We note that the second approach relaxes the reliability guarantee of the first approach (from tolerating $(n-1)$ failures of CSPs to $(n-t)$), and hence enjoys the space bonus: it only needs one tth of the storage cost in the first approach. However, the second approach also has its own drawback as it is very time consuming. In particular, since each CSP does not have enough information on its own to decide whether the trapdoor shares computed and uploaded in storage phase and the fresh shares sent in user's query are from an identical secret, at least one interaction is needed among all the other CSPs to recover and compare secret. In addition, since the search works in the way of per-file decision, the total number of interactions taken for a single keyword query is *linear* to the number of files stored at all the CSPs.

To address the efficiency bottleneck in the second approach, we propose the STRE mechanism. STRE works in a similar way of the (second) straightforward approach, but builds index and shares the "entrance" of inverted file list for accelerating search. STRE achieves constant time of interactions for a single keyword search. We will describe STRE in the next subsection.

C. STRE Protocols

We first present an overview of our proposed STRE, and then elaborate its detailed construction including security techniques for achieving our goals.

STRE builds with *inverted index*. In particular, to enable efficient keyword search, we initially scan the file collection and map each possible keyword to a (linked) list of files, ensuring that any file in the list contains the corresponding keyword. Given a keyword, it is efficient to search the related files by mapping the keyword and traversing the corresponding inverted file list. Moreover, since the inverted list is linked, we can consider the first node as the entrance of the list and use it to traverse the whole list. STRE stores this type of inverted index in each CSP.

To search the file that contains querying keywords, STRE builds upon the idea of sharing similar to the second naive approach. However, instead of sharing the keyword itself, STRE shares the first node (to be consistent, we also call it as trapdoor) mapped by the keyword. The advantage is that if the CSPs reconstruct the trapdoor, they can individually traverse the inverted file list and return search results.

To achieve our security goals, we need to overcome the following three challenges: (i) how to prevent CSPs from knowing the file content? (ii) how to encrypt the inverted index so that the CSP can efficiently search but learn nothing? (iii) how to design a protocol for multi-party secret reconstruction to ensure robustness? In what follows, we describe the de- tailed construction of STRE, as well as how we tackle these challenges.

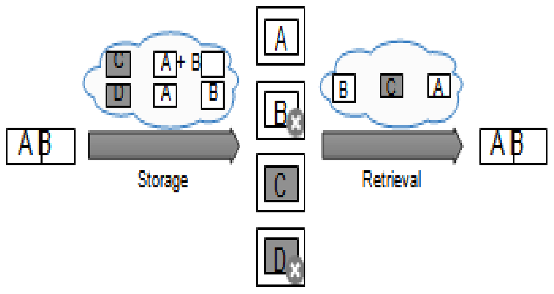


Figure2 A Toy Example for Encrypted File Encoding and Reconstruction

1) *Setup Phase:* The setup phase initializes the underly- ing cryptographic primitives, including two symmetric- key encryption schemes: SKE1 = (Gen, Enc, Dec) and SKE2 = (Gen, Enc, Dec), three pseudorandom functions P(,), Q(,) and R(,). The phase also assigns a master secret key (sk_1, sk_2, sk_3, sk_4) to the user.

   We assume the existence of some form of public-key infras- tructure where CSPs have registered their public keys, and all the entities can look up these public keys. In this way, i) a CSP can authenticate the identities of the other CSPs during collaborative search; ii) the user can authenticate the CSPs so that he/she can be sure that the retrieval protocol is run with the same CSPs that he/she previously ran the storage protocol with.

2) *Storage Protocol:* The storage protocol is for users to encrypt and distribute their files to multiple CSPs. Before uploading the files to the CSPs, the user encrypts the collection of files with sk_3 using SKE1 and encodes the ciphertext of them with erasure coding. Note that since the files have been encrypted, we can just use Rabin's information dispersal algorithm (IDA) [17] in STRE, which does not provide any confidentiality guarantee.

   Each encrypted file is divided into t equal-size *native chunks* to be stored at t CSPs. Moreover, the native chunks can be encoded by linear combinations to form another (n t) *code chunks* to be stored at the other (n t) CSPs. This enables us to reconstruct the encrypted file from any t out of n chunks so as to enhance the reliability of the outsourced files.

Fig. 2 shows a toy example demonstrating how to achieve reliability for n = 4 and t = 2. The encrypted file is split into two native 256-bit chunks (i.e., A and B), which are then encoded into another two code chunks with linear combination. Fig. 2 uses the trivial linear combination as C = A + B and D = A B and abuses "+" and " " as the operations in

Galois field GF(2^8)1. All the chunks are sent to the CSPs for

storage. Suppose two CSPs fail and lose B and D. Since B is the linear combination of A and C, the user can easily revert and compute B from A and C. The encrypted file is finally reconstructed.

To preserve the privacy of the inverted index (see the beginning of this section), we follow the approach of [7], [8]. In particular, we build the encrypted index in the following way. As shown in Fig. 3, for each file containing a keyword w, we assign it with a *node*, and the node is afterwards encrypted

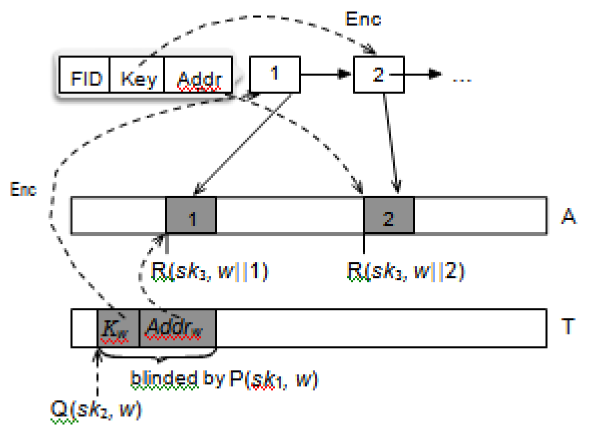


Figure 3 Encrypted Index

with SKE2 and stored in an array A. Each (plain) node contains three types of fields: i) file identifier field records the unique the identifier of the file containing w; ii) key field stores the symmetric key used for the *encryption* of the *next* node (the first node is encrypted with a random key K_w which will be elaborated later); iii) address field stores the address of the encrypted version of *next* node in array A2. In this way, if we can decrypt and obtain the first node of the inverted list in A, all the identifiers of the files containing w can be obtained in a recursive way.

Besides the array, a look-up table T is included to record the locations and encryption keys of the first node for all the inverted lists. Roughly, for the keyword w, we store the location (in A) and encryption key K_w of the first node of its inverted list in T[Q(sk_2, w)]. We also blind the content (i.e., the location and encryption key of the first node) of T[Q(sk_2, w)] with P(sk_1, w). We note that the above operations are to make the table T look as random as possible from the adversary without user's secret keys.

Finally, the user uploads the encrypted file chunks as well as the index to each CSP. We note that the index is replicated in each CSP for n 1 fault tolerance. On the other hand, even the CSPs accesses the index, they cannot get any useful information as the index has been protected using sophisticated encryption approaches.

3) *Retrieval Protocol:* In order to achieve privacy preserving keyword search over multiple clouds, we propose a novel retrieval protocol that consists of two stages: i) query sharing stage; ii) and reconstruction stage. The query sharing stage generates a (t, n)-secret sharing on the user's keyword query and distribute the shares to n CSPs. The reconstruction stage allows the user to obtain the query results when at least t (t n) CSPs are functioning.

In traditional single CSP storage [7], to retrieve the files including a certain keyword w, the user has to submit the trapdoor as P(sk_1, w) and Q(sk_2, w). Recall the encrypted index construction, the CSP can use Q(sk_2, w) to identify the location of K_w and $Addr_w$ about the inverted list of w, followed by removing the blinding with P(sk_1,w). After obtaining the entrance information K_w and $Addr_w$, the CSP can recursively identify and decrypt each node in the inverted list.

The key idea in STRE is *sharing* the trapdoor. In particular, instead of sending a whole trapdoor, we perform secret sharing on the trapdoor, and distribute each trapdoor share to the respective CSP for search. One benefit of the sharing-based approach is *trapdoor security*. Specifically, the adversary compromising some CSPs can only see a limited number of (random) shares, and not learn the whole trapdoor, or the information that two set of (trapdoor) shares are from the same trapdoor. Hence, the search pattern is hidden before the collaborative search. On the other hand, STRE enjoys the advantage over the second naive approach (see Section IV-B) as it uses an inverted index and shares the entrance information of the inverted list, avoiding the linear-growth reconstruction and comparison.

To make secret sharing on (P(sk_1, w), Q(sk_2, w)), we leverage the idea of multiple secret sharing. Specifically, we build a secret matrix consisting of P(sk_1, w) and Q(sk_2, w), and some random values which are used for checking the correctness of reconstruction. In addition, another "mirror matrix" defined the same as the secret matrix except replacing P(sk_1, w) and Q(sk_2, w) with zero, is published. Later on, when the secret matrix is reconstructed, the mirror matrix can be used for partially cheating detection. STRE finally shares the secret matrix with multiple secret sharing scheme [14] and distributes the share vector as trapdoor share for each CSP.

In the reconstruction stage, our approach uses the secure data aggregation scheme [18]. This is based on the observation that each CSP obtains a trapdoor share (in the form of a vector) from the user, and it is desirable to *collect* a threshold number of trapdoor shares together and further reconstruct the secret P(sk_1, w) and Q(sk_2, w). For security concern, it poses two requirements: i) each CSP should protect its trapdoor share, before obtaining the trapdoor share from other CSPs; ii) each available CSP should *finally* reconstruct the secret.

To address the above two concerns, STRE follows a *two-stage distribution*. In particular, upon receiving the trapdoor share from the user, each CSP creates a zero matrix B_i, and writes the trapdoor share into the corresponding column of the matrix.

$$B_i = \sum n_{B_i} \text{ is}$$

We note that the sum matrix the collection of all the trapdoor shares, which has sufficient information for reconstructing P(sk_1, w) and Q(sk_2, w). In the first distribution of STRE, each CSP makes (n, n)-secret sharing on B_i and is tribute the n shares to the corresponding CSP. In this way, each CSP is able to obtain n "sub-shares" respectively from B_1, B_2, . . . , B_n, and aggregates these sub shares into one share. We note that the aggregated share is also a share of B due to the additive homomorphism of secret sharing. Hence, in the second distribution of STRE, each CSP distributes its locally aggregated share to any other CSP, afterwards sums up all the aggregated shares from the other CSPs and itself, and obtains the collection of trapdoor shares B. We claim that the two-stage distribution can meet our requirements, as it does not need any CSP to expose its trapdoor share beforehand, and all the CSPs collaborate to concurrently reconstruct B.

After obtaining B, each CSP recovers the secret matrix with the multiple secret sharing scheme [14]. It also checks the correctness of reconstruction by comparing the reconstructed secret matrix with the mirror matrix. If the correctness check is passed, the CSP picks out P(sk_1, w) and Q(sk_2, w), uses them to search the files containing the underlying keyword w [7], [8] and returns back the corresponding encrypted file chunks. The user, after collecting all the related file chunks from several CSPs, groups these chunks according to the unique file identifier, recovers the whole encrypted file with erasure coding, and finally decrypts them, obtaining the search results. In summary, STRE can tolerate the failure of (n t) CSPs with an expense of additional (nt)/t times storage space in total. The reason is the erasure coding, which leads to n/t storage blowup. It is also worth noting that although our current discussion is focused on CSPs that store the same amount of file chunks, our mechanism can be easily extended to a more flexible storage strategy. For example, we can encode the encrypted file into more than n chunks and store more than one chunk in the cheaper or more reliable CSP.

V. SECURITY ANALYSIS

We now analyze the security of STRE mechanism in the assumed honest-but-curious environment. Our analysis is in terms of *reliability*, *semantic security*, *trapdoor security* and *robustness*, which were defined in Section III-B.

A. Reliability

Recall from Section IV-C2 that the encrypted file (say c_f) is encoded by (n, t)-erasure coding. Specifically, c_f is divided into *t native chunks*, using which another *n t code chunks* are constructed through linear combination. This operation can be viewed as $(c_1, c_2, . . . , c_n) = E \cdot c_f$

where $c_1, . . . , c_t$ are the native chunks, $c_t, . . . , c_n$ are the code chunks and c_f is used (with a slight abuse of notation) to express the encrypted file in chunk form. E is a (n t) encoding matrix which consists of a (t t) unit matrix in its upper part. The rank of E is t, that is, any t row vectors of E are linear independent.

During reconstruction, after obtaining any t chunks, denoted c_{i1} , . . . , c_{it} , the reconstruction is realized by selecting the corresponding rows of E and constructing the decoding matrix E. Note that E is invertible. Thus

$$c_f = E^{-1}(c_{i_1}, c_{i_2}, . . . , c_{i_t})$$

which ensures reliability of STRE.

B. Semantic Security

We begin by considering file confidentiality. First of all, each file in the collection is encrypted using a symmetric-key encryption scheme. Since the erasure code is used for encoding the encrypted file, and it is retrievable in the case at least t encoded chunks are obtained, which does not affect the file confidentiality. Hence, the security of files can be reduced to that of the used symmetric-key encryption scheme in file encryption.

Secondly, though a copy of the encrypted index is stored at each CSP, it is just used for recording the *relation* of keywords and files, and useless for extracting information from plain files. Based on the above two points, we can claim that the index and the collection of encrypted file chunks never leak any partial information about the original files except the file lengths.

The second important requirement to ensure semantic security is to protect the dynamic process, during which an adversary can dynamically submit keyword queries and receive results to help launch attack. Precisely, semantic security requires that no partial information except access pattern and search pattern is leaked during keyword-based queries. Note that in file retrieval, the process of search is mainly executed with the reconstructed trapdoor (i.e., P(sk_1, w) and Q(sk_2, w) in STRE mechanism) and the encrypted index. Thus, in order to show STRE mechanism achieves this security requirement, following the simulation-based definition from Curtmola and colleagues [7], we need to build a simulator, which is able to learn the allowable leakage, and attempts to simulate the array A, look-up table T and (P(sk_1, w), Q(sk_2, w)) as random objects from adversary's perspective.

Before describing the simulator, we introduce a set of notations. Suppose the file collection **f** has been queried for q times. Without loss of generality, we can denote the queried keywords w_1, w_2, . . . , w_q. We use **f** (w_i) to denote a lexicographically ordered vector consisting of the identifiers of files in **f** containing w_i. Obliviously, the simulator must know the access pattern (see its definition in Section III-B) **f**(w_1), . . . , **f**(w_q). We build the simulations as follows.

- *Build* A* *to Simulate* A. The simulator builds A* as follows. It assign *random* strings (with the same length) for each file identifier in the sets **f** (w_1) through **f** (w_q), and put these random strings in *random* positions of "encryptions" (random strings in fact) as A except that the result is stored in random positions rather than the real encryptions output by SKE2 stored in positions derived from R(sk_3,). Since either the secret key sk_4 or the random keys used to encrypt the node in inverted list cannot be obtained by adversary with all but negligible probability, the adversary cannot distinguish the real random strings from the output by SKE2 or R(sk_4,), and hence A* is indistinguishable from A.
- *Build* T* *to Simulate* T. Similar to the simulation above, the simulator has obtained the random string (say γ_i^*) for the first node in each set **f** (w_1), . . . , **f** (w_q). The simulator just picks two random strings respectively for simulating P(sk_1, w_i) and Q(sk_2, w_i)

ξ_i^* and υ_i^*

for $i = 1, . . . , q$. In particular, the simulator stores the (simulated) result γ_i^* υ_i^* in the position ξ_i^* of T*. In summary, T* consists of q encryptions generated by XOR-ing a (simulated) message with a random string υ_i^* and stored in the position ξ_i^*. For the pseudo-randomness of P(sk_1,) and Q(sk_2,), the adversary cannot distin- guish ξ_i^* and υ_i^* from respective output. Hence T* is *Build* (P_i^*, Q_i^*) *to Simulate* (P(sk_1, w_i), Q(sk_2, w_i)). The reconstructed trapdoor can be easily simulated as (ξ_i^*, υ_i^*) for $i = 1, 2, . . .$, q. The pseudo-randomness of P(sk_1,) and Q(sk_2,) guarantees the indistinguishabil- ity of (ξ_i^*, υ_i^*) from (P(sk_1, w_i), Q(sk_2, w_i)).

C. Trapdoor Security

We now prove that the trapdoor security of STRE mechanism can be reduced to the security of multiple secret sharing scheme [14].

Suppose an adversery has a non-negligible advantage in winning the game elaborated in Section III-B. Our aim is to build a simulator S which is able to *distinguish two different secrets with* ($t - 1$) *shares* in multiple secret sharing scheme [14] by running A. We provide the simulation as follows.

After executing and getting a collection of files **f** , runs the *storage protocol* on **f** to obtain the secret key (sk_1, sk_2, sk_3, sk_4), a collection of (encrypted) chunks and the (encrypted) index. is given the encrypted index.

For 's trapdoor query (, w_0, w_1), where $< t$, responds as follows.

- Construct two same secret matrix S_0, S_1, and respectively write P(sk_1, w_i) and Q(sk_2, w_i) into S_i for i = 0, 1.
- Submit (C, S_0, S_1) to the challenger which returns the trapdoor shares v_i of S_b on each $i \in$ C, where $b \in_R \{0, 1\}$ is kept secret from A and S.
- Return the set of trapdoor shares $\{v_i\}_{i\in C}$ back to A.

Finally, outputs 's submission b' which is the guess of b.

It is clear that the view of A when run as a sub-routine

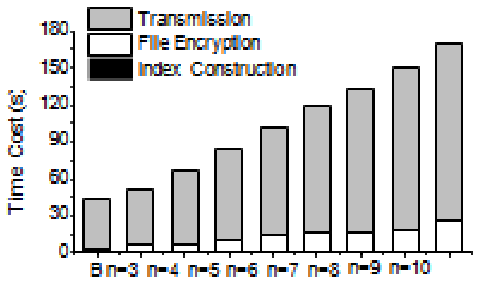
By is identical to the view of the adversary in the game elaborated in Section III-B. Thus, we can claim that if the adversary succeeds in the game of Section III-B, there exists a simulator having the same probability on distinguishing two different secrets from multiple secret sharing scheme with less than t shares.

D. Robustness

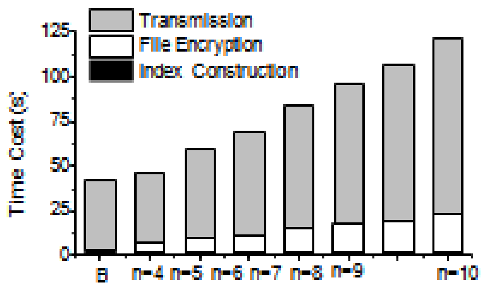
The CSP authenticates the identity of the other CSPs during the collaborative search. If the authentication fails that means the sub-shares distributed by CSPs have been either modified or substituted, and the protocol is aborted. Since the share matrix is imposed with (n, n)-secret sharing and any incorrect sub-share will lead to the failure of reconstruction, none of the CSPs can get the other CSPs' shares properly. If the authentication succeeds, as elaborated in our protocol, each CSP will get correct distributions from all the other CSPs and successfully builds the share matrix for further search.

VI. EXPERIMENTAL STUDY

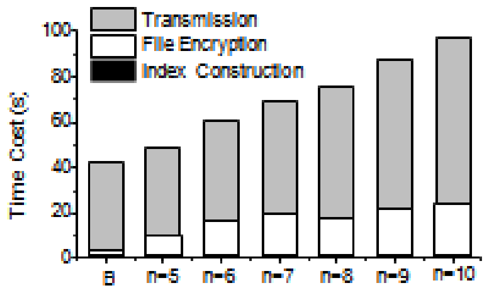
Our experiments were run using as client machine a Linux Mint 14 machine with Intel(R) Core(TM)2 Duo CPU clocked



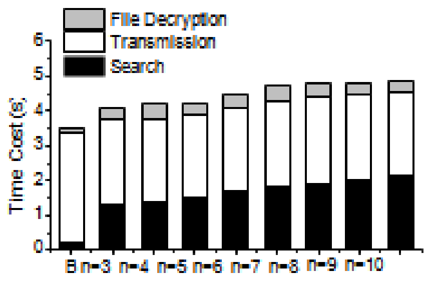
(a) t = 2 in File Storage



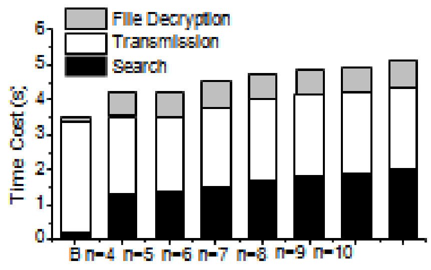
(b) t = 3 in File Storage



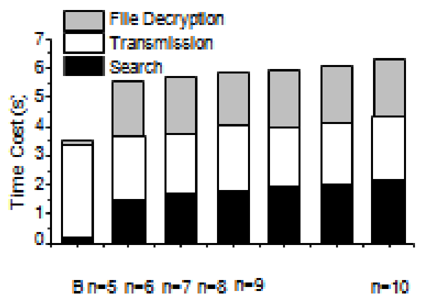
(c) t = 4 in File Storage



(d) t = 2 in File Retrieval



(e) t = 3 in File Retrieval



(f) $t = 4$ in File Retrieval

Fig. 4. File Storage and Retrieval Performance (the leftmost column labeled B in each subfigure is the time cost of baseline approach, while the following columns are for STRE in different cases)

at 2.40 GHz and 2 GB of memory. We simulate the multiple clouds environment by using several 32-bit Linux T1 Micro servers in Amazon EC2 platform. Note that, although this is not an actual multi-cloud storage environment, it is sufficient for the purpose of our experiment, since we virtualize a large number of connected and collaborative cloud computers into a logical multi-cloud environment. This experimental setting is also helpful for simulating storage crash through erasing the stored data at the servers in Amazon EC2 platform.

We use a real world dataset, namely the Enron email dataset [9] as our corpus. We extract a subset of emails (i.e., a total number of 12008 emails in the subdirectory beck-s) from Enron dataset. This dataset is suitable as it can simulate organizations storing emails to a remote server in encrypted form and wishing to search through them from time to time. Before testing, we pre-processed the dataset by generating keywords and permutation tables for R(,), Q(,). Specifically, we cleaned up the corpus through the Porter stemming algorithm [19], removed content-independent words with a spelling corrector, and built inverted index for the most 1500 frequent keywords for generating keywords. The prefix method [20] was used to generate permutation tables serving as R(,) and Q(,).

We adopt a classic symmetric searchable encryption scheme SSE-1 [7] as a baseline approach, which shares an identical in-dex with STRE mechanism but does not consider the reliability of the searchable encrypted data.

A. Performance of File Storage

In this set of experiments, we aim to evaluate the efficiency of our storage protocol by comparing the total time taken to upload the files using our protocol with that using the baseline approach. In Fig. 4(a), Fig. 4(b) and Fig. 4(c), we respectively fix the threshold t = 2, 3 and 4, and vary the number of CSPs from t + 1 to 10.

It is not surprising to see that our STRE mechanism takes more time to complete the file storage than the baseline approach. Note that the index construction time is actually neg- ligible (nearly one second) in all the tests. Our file encryption time is slightly longer than that of the baseline approach and also increases with the number of CSPs. This is because we utilize the Plank's fast Galois Field Arithmetic Library [21] to encode the encrypted files into multiple chunks to be sent to the multiple CSPs, while the baseline approach just needs to encrypt the files one by one. Our transmission time is relatively longer than the baseline approach, which is also due to the existence of the multiple CSPs. The user needs to send all the file chunks to n CSPs rather than just communicating with one CSP in the baseline approach.

Regarding storage space, the index consisting of the array and look-up table takes up to 8736 KB. Besides this, in file encryption, SSE-1 [7] (i.e. our baseline approach), produces 25 MB ciphertext, while each CSP in STRE mechanism just needs $1/t$ times of this space to store the encoded encrypted files whatever n is. This enables us to save nearly (t 1)/t proportion of space for each individual CSP.

B. Performance of File Retrieval

Without loss of generality, we randomly pick a query keyword and then evaluate the efficiency of retrieving all the files containing the query keyword. Note that in this evaluation, we assume the worst case, that is, all the n CSPs participate in the collaborative search but only t of them return the query results.

One query with the keyword America returns 961 files in our corpus. Fig. 4(d), Fig. 4(e) and Fig. 4(f) show the time taken to execute this query when varying the number of CSPs and threshold t. As expected, to provide reliability guarantee, our STRE mechanism needs more time to retrieve the files compared to the baseline approach. Specifically, the retrieval protocol in STRE mechanism consists of two processes and hence it requires two rounds of interaction with the CSPs as well as encrypted file reconstruction. Though the search and file decryption are slower than the baseline approach, our retrieval protocol is more efficient in terms of transmission. The main reason is that the multi-cloud setting enables us to concurrently receive data with multiple threads at the user side. To sum up, our STRE mechanism achieves both security and reliability without introducing significant overhead compared to the baseline approach, since our execution time is still within seconds.

VII. RELATED WORK

Since our work is related to both privacy-preserving key-word search and reliability issues in the cloud, we give a brief review of these two threads of works.

A. Privacy-Preserving Keyword Search

The problem of searching on encrypted data can be solved in its most generic case using the work of Goldreich and Ostrovsky [22] [23] on oblivious RAMs. Unfortunately, this approach requires multiple interactions and has a high compu-tation overhead. Even though the recent work [24] has greatly improved the efficiency of oblivious RAMs, it is based on a stronger assumption over the architecture. That is, either hybrid cloud setting or a public cloud equipped with trusted computing base is required such that the ORAM nodes and oblivious load balancer can be deployed in a fully trusted environment. Therefore, for higher efficiency, the security requirements must be weakened appropriately by allowing some limited information (i.e., the access pattern and search pattern) about the messages and the queries revealed to the adversary.

Based on this intuition, searchable encryption was first introduced by Song et al. [4], in which a user stores his/her encrypted data in a semi-trusted server and later searches with a certain keyword. In their proposal [4], each word is independently encrypted under a specified two-layered encryption. Given a token (i.e., trapdoor) for a keyword, the server can strip the outer layer and assert whether the inner layer is of the correct form. Later, Goh [12] introduced bloom filter [25] to construct secure indexes for keyword search, which allows server to check if a file contains a certain keyword without scanning the entire file. Chang et al. [11] offered similar solutions to the problem of privacy-preserving keyword search through building pseudo-random bits to mask a dictionary-based keyword index for each file. A formal treatment to symmetric searchable encryption was presented by Curtmola et al. [7]. They provided improved security notions for symmetric searchable encryption and presented “index” approach, in which an array and a look-up table are built for the entire file collection. Each entry of the array is used to store an encryption of file identifier associated with a certain keyword, while the look-up table enables one to efficiently locate and decrypt the appropriate entry from array.

Toward extending the functionalities of privacy preserving keyword search, Li et al. [26] considered the problem of fuzzy keyword search over encrypted data, and proposed a solution with the wildcard-based fuzzy set construction method. Cao et al. [27] solved the challenging problem of privacy-preserving multi-keyword ranked search, and proposed a scheme based on secure inner product computation. Sun et al. [28] then made improvements on the multi-keyword ranked search scheme [27], and presented an efficient scheme under a stronger threat model.

Aiming at providing searchable encryption with efficient update, Liesdonk et al. [15] presented two schemes: the first one transforms each unique keyword to a searchable representation such that user can keep track of metadata items via appropriate trapdoor. The second one deploys a hash chain by applying repeatedly a hash function to an initial seed. Since only the user knows the seed, he/she can traverse the chain forward and back- ward, while the server is just able to traverse the chain forward only. Kamara et al. [8] provided the first symmetric searchable encryption construction satisfying sublinear search time, security against adaptive chosen keyword attacks, compact index and the ability to add and delete files efficiently. Their solution is based on Curtmola et al’s work [7], but adds another data structure namely deletion table and deletion array to record the metadata for the added/deleted files, which can be utilized to adaptively change the appropriate entries in search array and search table. The dynamic symmetric searchable encryption has been extended to paralleled setting [10]. Recently, Cash et al. [29] implemented a prototype supporting multiple keyword search and dynamic updates for large databases. Naveed et al. [30] proposed a dynamic searchable encryption scheme via blind storage for further hiding filename and length. Stefanov et al. [31] and Hahn et al. [32] improved dynamic searchable encryption on security and efficiency respectively.

Some security concerns about existing searchable encryption schemes have been raised. Islam et al. [33] introduced a novel attack that exploits data access pattern leakage to disclose significant amount of sensitive information using a modicum of priori knowledge. Liu et al. [16] showed that the search pattern leakage can result in non-trivial risks. They proposed two concrete attacks exploiting user’s search pattern and some auxiliary background knowledge to disclose the underlying keyword in user’s query.

For the aim of complete security, although oblivious RAM-based protocol [22], [23] can be used for completely hiding access pattern and search pattern, as we stated before they are computationally intensive and do not scale well for real world datasets. The recent proposals in [33], [16] for the same purpose are based on adding dummy keywords/files to obfuscate query/result, causing communication overhead and effective in just limited cases. As with existing work, our mechanism cannot keep the privacy of access pattern, but it is able to protect search pattern before collaborative search.

In summary, we note that existing works enable efficient, flexible and dynamic privacy-preserving keyword search, but fail to consider reliability of outsourced data. Specifically,

existing works on searchable encryption rely on a single server/CSP, which could still be vulnerable to a single-point of failure [34] even though cloud storage provides an on-demand remote backup solution.

B. Reliable Cloud Storage

Ensuring data availability in distributed storage system is critical, since node failures are prevalent. The classical approach is via *erasure codes,* which encodes original data and stripes encoded data across multiple nodes. Erasure codes can tolerate multiple failures and allow the original data to remain accessible by decoding the encoded data stored in other surviving nodes. Recent studies (e.g., [35], [36] to list a few) proposed regenerating codes for distributed storage. Regenerating codes built on the concept of network coding aim at intelligently mixing data blocks that are stored in existing storage nodes, and then generating data at a new storage node. It is shown that regenerating codes reduce the data repair/recovery traffic over traditional erasure codes subject to the same fault tolerance level.

Unlike existing work on reliable storage, which builds with a proxy based architecture and makes the proxy server as an interface for storage repair between client applications and clouds, reliability in this paper is considered as *result recovery*. That is, after extracting at least some partial results from CSPs, the *user* locally recovers the clear files containing the searched keyword.

VIII. CONCLUSION

In this paper, we propose the STRE mechanism, to promote reliability of outsourced searchable encrypted data. In STRE, user's searchable encrypted data is strategically distributed to and stored at multiple CSPs, so as to achieve high crash tolerance. Besides reliability, the STRE mechanism also affords efficient and flexible storage properties and partially hidden search pattern. Extensive experiments demonstrate the efficiency of our scheme. As indicated in the experiment results, the largest overhead of our proposed mechanism compared with the classical approaches is communication time. This is because increasing the number of CSPs implies adding data redundancy. Thus, to reduce data redundancy and save storage and bandwidth, we will explore the regenerating code methods, which have been recently utilized to minimize the bandwidth for file retrieval while tolerating failures.

References

[1] J. Weinman. (2013) Will multiple clouds evolve into the intercloud?

[2] Y. Zhu, H. Hu, G.-J. Ahn, and M. Yu, "Cooperative provable data possession for integrity verification in multicloud storage," *IEEE Transactions on Parallel Distributed Systems*, vol. 23, no. 12, pp. 2231–2244, 2012.

[3] D. Owens, "Securing elasticity in the cloud," *Communications of the ACM*, vol. 53, pp. 46–51, 2010.

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

[5] S. Kamara and K. Lauter, "Cryptographic cloud storage," in *Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, vol. 6054, pp. 136–149.

[6] H. Blodget. (2011) Amazon's cloud crash disaster permanently destroyed many customers' data.

[7] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *ACM Conference on Computer and Communications Security*, 2006, pp. 79–88.

[8] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption." in *ACM Conference on Computer and Communications Security*, 2012, pp. 965–976.

[9] B. Klimt and Y. Yang, "The enron corpus: A new dataset for email classification research," in *Machine Learning: ECML 2004*, ser. Lecture Notes in Computer Science, 2004, vol. 3201, pp. 217–226.

[10] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, 2013, vol. 7859, pp. 258–274.

[11] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data." in *Applied Cryptography and Network Security*, ser. Lecture Notes in Computer Science, vol. 3531, 2005, pp. 442–455.

[12] E.-J. Goh, "Secure indexes," IACR Cryptology ePrint Archive 216, Tech. Rep., 2003.

[13] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[14] L. Bai and X. Zou, "A proactive secret sharing scheme in matrix projection method," *International Journal of Security and Networks*, vol. 4, no. 4, pp. 201–209, 2009.

[15] P. van Liesdonk, S. Sedghi, J. Doumen, P. H. Hartel, and W. Jonker, "Computationally efficient searchable symmetric encryption," in *Secure Data Management*, ser. Lecture Notes in Computer Science, 2010, vol. 6358, pp. 87–100.

[16] C. Liu, L. Zhu, M. Wang, and Y. an Tan, "Search pattern leakage in searchable encryption: Attacks and new constructions," *Information Sciences*, vol. 265, no. 1, pp. 176–188, 2014.

[17] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *Journal of the ACM*, vol. 36, no. 2, pp. 335–348, Apr. 1989.

[18] X. Zhao, L. Li, G. Xue, and G. Silva, "Efficient anonymous message submission," in *IEEE INFOCOM*, 2012, pp. 2228–2236.

[19] M. F. Porter, "An algorithm for suffix stripping," *Program: Electronic Library and Information Systems*, vol. 14, pp. 130–137, 1980.

[20] J. Black and P. Rogaway, "Ciphers with arbitrary finite domains," in *Topics in Cryptology - CT-RSA 2002*, ser. Lecture Notes in Computer Science, 2002, vol. 2271, pp. 114–130.

[21] J. S. Plank, "Fast galois field arithmetic library in c/c++," University of Tennessee, Tech. Rep., 2007.

[22] R. Ostrovsky, "Efficient computation on oblivious rams," in *ACM symposium on Theory of computing*, 1990, pp. 514–523.

[23] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," *Journal of the ACM*, vol. 43, pp. 431–473, 1996.

[24] E. Stefanov and E. Shi, "Oblivistore: High performance oblivious cloud storage," in *IEEE Symposium on Security and Privacy*, 2013, pp. 253–267.

[25] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communication of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[26] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *IEEE INFOCOM*, 2010, pp. 441–445.

[27] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," in *IEEE INFOCOM*, 2011, pp. 829–837.

[28] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li, "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," in *ACM Symposium on Information, Computer and Communications Security*. ACM, 2013, pp. 71–82.

[29] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Dynamic searchable encryption in very-large databases: Data structures and implementation," in *Network and Distributed System Security Symposium*, 2014.

[30] M. Naveed, M. Prabhakaran, and C. Gunter, "Dynamic searchable encryption via blind storage," in *IEEE Symposium on Security and Privacy*, 2014, pp. 639–654.

[31] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage," in *Network and Distributed System Security Symposium*, 2014.

[32] F. Hahn and F. Kerschbaum, "Searchable encryption with secure and efficient updates," in *ACM Conference on Computer and Communications Security*, 2014, pp. 310–320.

[33] M. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *Network and Distributed System Security Symposium*, 2012.

[34] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communication of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[35] B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote data checking for network coding-based distributed storage systems," in *ACM Workshop on Cloud Computing Security*, 2010, pp. 31–42.

[36] Y. Hu, Y. Xu, X. Wang, C. Zhan, and P. Li, "Cooperative recovery of distributed storage systems from multiple losses with network coding," *IEEE Journal on Selected Areas in Communications*, vol. 28, pp. 268–276, 2010.