STRE: Privacy-Preserving Storage and Retrieval over Multiple Clouds

Jingwei Li[1(✉)], Dan Lin[2], Anna Squicciarini[3], and Chunfu Jia[1]

[1] Nankai University, Tianjin, People's Republic of China
lijw1987@gmail.com, cfjia@nankai.edu.cn
[2] Missouri University of Science and Technology, Rolla, USA
lindan@mst.edu
[3] Pennsylvania State University, State College, USA
asquicciarini@ist.psu.edu

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. 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 in multiple cloud service providers (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.

Keywords: Private keyword search · Searchable encryption · Cloud computing

1 Introduction

Cloud computing is growing exponentially, whereby there are now hundreds of cloud service providers (CSPs) of various sizes. This multi-cloud environment [2,10] 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 that helps make interactions among multiple clouds.

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 guarantees on the trustworthiness of each CSP [7], cloud consumers are typically suggested to adopt searchable encryption techniques [8] to encrypt their outsourced data in

J. Tian et al. (Eds.): SecureComm 2014, Part I, LNICST 152, pp. 36–44, 2015.
DOI: 10.1007/978-3-319-23829-6_3

a way that the encrypted data can be directly searched by the CSPs without decryption. Despite many efforts (e.g., [5,6], etc.) devoted to improving efficiency and security of the searchable encryption, there is little consideration on ensuring the reliability of the searchable encrypted data.

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. It is worth noting that a comprehensive 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 keyword or not [3,4]. In our STRE mechanism, this risk originated from pattern leakage is lowered because the search is conducted in distribution and the search pattern will be revealed only if there are more than t CSPs collude.

The rest of the paper is organized as follows. In Sect. 2, we present the system model as well as introduce notations used in this paper. The proposed STRE mechanism is provided in Sect. 3. Finally, Sect. 4 draws the conclusion of this paper.

2 Model and Notations

2.1 System Model

In this work, we consider the cloud storage services offered in a multi-cloud environment, which involves two types of entities: (1) *Users*, who store a large number of encrypted files in multiple clouds and execute keyword-based queries to access and manipulate their stored files; (2) *Cloud Service Providers* (CSPs), who possess storage and computation resources and are willing to cooperatively store and manage the users' files. Under this architecture, we focus on searchability of encrypted data, stored by users in one or many multi-cloud service

providers. Informally, searchability 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 assume that the CSPs in the multi-cloud environment are honest-but-curious, in that each CSP will honestly execute the proposed protocols but they may be curious and try to learn from the information stored at their sites. Our design goals include the following objectives:

- **Reliability.** Given n CSPs, the system should still function if at most $n - t$ ($t < n$) CSPs have been compromised, where t is a predefined threshold value for the system.
- **Semantic Security.** The system should be semantically secure [3] by satisfying the following two requirements. First, given the file index $\mathbf{I}$ and the collection of encrypted files, no adversary can learn any information about the original files $\mathbf{f}$ except the file lengths. Second, given a set of trapdoors for a sequence of keyword queries, no adversary can learn any information of the original files except the access pattern (i.e., the identifiers of the files that contain the query keyword) or 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. This requirement holds even if at most $(t - 1)$ CSPs collude together.
- **Robustness.** We require that (1) when the protocol successfully completes, the correct files are returned to the users; (2) 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.

2.2 Notations

Let $\Delta = \{w_1, \ldots, w_{|\Delta|}\}$ be a dictionary of $|\Delta|$ distinct keywords in lexicographic order, and 2^{Δ} be the set of all possible files with keywords in Δ. Furthermore, let $\mathbf{f} \subseteq 2^{\Delta}$ be a collection of files $\mathbf{f} = \{f_1, f_2, \ldots, f_{|\mathbf{f}|}\}$, where $\mathsf{id}(f)$ is the identifier of file f whereby the identifier could be a string such as a memory location that uniquely identifies a file, and $\mathbf{f}(w)$ is the lexicographically ordered vector consisting of the identifiers of all files in $\mathbf{f}$ containing the keyword w. Suppose S is a matrix. $S[i][j]$ denotes the element at the ith row and jth column of S, while $S[i]$ denotes the ith column vector of S. If S is a vector, we also utilize $S[i]$ to denote the ith element of S.

3 STRE Mechanism

3.1 Overview

The STRE mechanism consists of two major phases: Storage Phase and Retrieval Phase.

Storage Phase. This phase consists of two main steps:

- **Step S1.** A master secret key msk is generated from a security parameter 1^λ and given to the user. Note that the security parameter 1^λ which is assumed to be known to all the adversaries, specifying 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.
- **Step S2.** Upon taking a collection of files $\mathbf{f}$ and master secret key msk as input, user generates and uploads encrypted file chunks and file index $(\mathbf{c}_i, \mathbf{I})$ to the ith CSP for $i = 1, 2, \ldots, n$.

Retrieval Phase. This phase includes three steps:

- **Step R1.** The user generates a collection of keyword trapdoors $\{tp_i\}_{i=1}^n$ based on the query keyword w and the master secret key msk, and then send each trapdoor to the respective CSP.
- **Step R2.** n CSPs collaborate together to search w, and the ith CSP returns a collection of encrypted chunks $\mathbf{y}_i$ back to the user for $i = 1, 2, \ldots, n$. Note that if a certain CSP crashes, its response is $\mathbf{y}_i = \emptyset$.
- **Step R3.** The user uses his/her master secret key to obtain a collection of clear files $\mathbf{x}$ from at least t non-empty $\mathbf{y}_i$ in $\{\mathbf{y}_i\}_{i=1}^n$. The correctness of protocol requires that for any file f, $f \in \mathbf{x}$ holds when and only when $\mathsf{id}(f) \in \mathbf{f}(w)$.

3.2 STRE Protocols

Let $\mathsf{SKE1} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and $\mathsf{SKE2} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ denote two symmetric-key encryption schemes. We propose two novel efficient protocols: *Storage Protocol* and *Retrieval Protocol*, respectively used in the storage phase and retrieval phase.

Storage Protocol. The storage protocol is for users to encrypt and distribute their files to multiple CSPs. We present its detail as follows.

Step S1: Given the security parameter 1^λ, the following computations are executed.

(1) Initiate three pseudo-random functions: $\mathsf{P} : \{0,1\}^\lambda \times \{0,1\}^\lambda \rightarrow \{0,1\}^{\lambda + \log_2 r}$, $\mathsf{Q} : \{0,1\}^\lambda \times \{0,1\}^s \rightarrow \{0, 1, \ldots, |\Delta|\}$, $\mathsf{R} : \{0,1\}^\lambda \times \{0,1\}^{s + \log_2(\max_{w \in \Delta} |\mathbf{f}(w)|)} \rightarrow \{0,1\}^{\log_2 r}$, where r is the total number of appearances of keywords in $\mathbf{f}$ and s is the bit-size of each keyword.
(2) After computing $msk_1, msk_2, msk_3 \in_R \{0,1\}^\lambda$ and $msk_4 = \mathsf{SKE1.Gen}(1^\lambda)$, send the master secret key $msk = (msk_1, msk_2, msk_3, msk_4)$ to user.

Step S2: User builds an index $\mathbf{I}$ similar (as shown in Fig. 1) to [3,4]. This index includes a search array A and a look-up table T, which respectively contains r and $|\Delta|$ entries. We then describe how to construct this index for the files consisting of a keyword $w \in \Delta$.

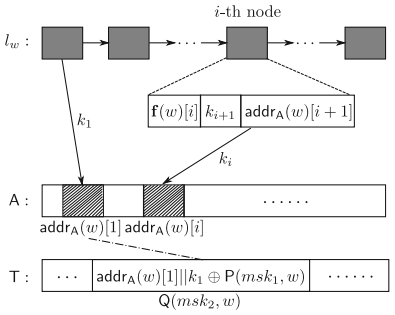

Fig. 1. Compressed Index

(1) Create a list $l_w = (N_1, \ldots, N_{|\mathbf{f}(w)|})$, where each node N_i corresponds to a certain file consisting of keyword w. We specify the structure of N_i as $N_i = \mathbf{f}(w)[i]||k_{i+1}||\ \mathsf{addr_A}(w)[i+1]$, where $\mathbf{f}(w)[i]$ is the identifier of the ith file in $\mathbf{f}(w)$ (i.e., the ith file that consists of w), $k_{i+1} = \mathsf{SKE2.Gen}(1^\lambda)$ is a symmetric key to be used for encrypting the next node N_{i+1} and $\mathsf{addr_A}(w)[i+1] = \mathsf{R}(msk_3, w||\ (i+1))$ which will be the location for storing the encryption of next node N_{i+1} in A. Note that for the last node $N_{\mathbf{f}(w)}$, the stored information $k_{|\mathbf{f}(w)|+1} = 0^\lambda$ and $\mathsf{addr_A}(w)[|\mathbf{f}(w)|+1] = 0^{\log_2 r}$.
(2) For each node N_i where $2 \le i \le |\mathbf{f}(w)|$, compute and store the encryption $C_i = \mathsf{SKE2.Enc}(k_i, N_i)$ at the location $\mathsf{addr_A}(w)[i]$ in A.
(3) For the first node N_1, after randomly picking its encryption key $k_1 = \mathsf{SKE2.Gen}(1^\lambda)$ and storage position $\mathsf{addr_A}(w)[1] = \mathsf{R}(msk_3,\ w||1)$, store $C_1 = \mathsf{SKE2.Enc}(k_1, N_1)$ at the location $\mathsf{addr_A}(w)[1]$ in A.
(4) Mask $\mathsf{addr_A}(w)[1]$ and k_1 by computing and storing $(\mathsf{addr_A}(w)[1]||k_1) \oplus \mathsf{P}(msk_1, w)$ at location $\mathsf{Q}(msk_2, w)$ in T. This enables CSP to use $\mathsf{P}(msk_1, w)$ and $\mathsf{Q}(msk_2, w)$ to know N_1 and further efficiently access the identifiers of files that consist of w.

Moreover, the user encrypts all the files in $\mathbf{f}$ and attaches the MDS code to each of them. Informally, after encryption, each encrypted file is firstly divided into t equal-sized *native chunks*. Further, the native chunks can then be encoded by linear combinations to form another $(n-t)$ *code chunks*. All the n chunks (including native chunks and code chunks) will be sent one-to-one to the n CSPs. This enables us to reconstruct the encrypted file from any t out of n chunks (from t CSPs) so as to enhance the reliability of the outsourced files. The detailed process for each file $f \in \mathbf{f}$ is described as follows.

(5) After computing the ciphertext $c = \mathsf{SKE1.Enc}(msk_4, f)$, divide c into t equal-size native chunks, denoted by $\{c'_i\}_{i=1}^t$.
(6) Construct the n code chunks through linear combination. Specifically, pick an encoding matrix $\overline{E} = [\alpha_{ij}]_{n \times t}$ for some coefficients in the Galois field $\mathrm{GF}(2^8)$ with a rank of t, and compute $c_i = \sum_{j=1}^t \alpha_{ij} c'_j$ for $i = 1, 2, \ldots, n$. Then, each code chunk and the identifier $\mathsf{id}(f)$ form a pair $(\mathsf{id}(f), c_i)$. Note that the encoding matrix $\overline{E}$ should be kept at local for encrypted file reconstruction in future.

Finally, the user uploads the encrypted file chunks as well as the metadata (i.e., the index) to each CSP. Specifically, for $i = 1, 2, \ldots, n$, user sends $(\mathbf{c}_i, \mathbf{I})$ to the ith CSP, where $\mathbf{c}_i = \{(\mathsf{id}(f), c_i) : f \in \mathbf{f}\}$ is the set of pairs of file identifier and its ith code chunk.

Retrieval Protocol. In order to achieve privacy preserving keyword search over multiple clouds, we propose a novel retrieval protocol that consists of two stages: (1) query sharing stage; (2) 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 ($t \leq n$) CSPs are functioning.

Recall that a user's keyword query in [3] is typically described as a pair of *location* (e.g., $\mathsf{Q}(msk_2, w)$) and *blinding value* (e.g., $\mathsf{P}(msk_1, w)$), where *location* records the location of the first node of the searched keyword list in T, and *blinding value* is a shadow for this entry in T to prevent CSPs from accessing it. In order to ensure *trapdoor security*, the query sharing stage in our retrieval protocol conducts secret sharing on both the *location* and the *blinding value*. This type of trapdoor is randomly and completely shared with n CSPs and its privacy is preserved even if at most $(t-1)$ CSPs collude together. In this way, the search pattern is hidden before the collaborative search. More specifically, we leverage Bai's multiple secret sharing scheme [1]. We build and share a secret matrix S consisting of both the secrets and some random values which are used for partially checking the correctness of reconstruction later. We let $S[1][1]$ and $S[1][2]$ record the *blinding value* and *location* respectively, and pick random values to fill the other entries of S. In addition, another "mirror matrix" S' defined the same as S except $S'[1][1] = 0, S'[1][2] = 0$ is published. Later on, when the secret matrix is reconstructed, S' can be used for partially correctness checking.

In the reconstruction stage, our approach is based on the secure data aggregation scheme [9]. In a general sense, for $i = 1, 2, \ldots, n$, after receiving the share v_i on secret matrix S, the ith CSP maintains a $(t \times n)$ matrix $\overline{B}_i$ with its share v_i in the ith column and 0 otherwise. After making a (n, n)-secret sharing on each entry of such a matrix, each CSP keeps one share $\overline{B}_{ii}$ at local and respectively distributes the remaining shares $\{\overline{B}_{ij}\}_{j \neq i}$ to the other $(n-1)$ CSPs. In this way, each CSP is able to obtain n "sub-shares", $(n-1)$ received from other CSPs and one kept by itself, and compute one share, say $\overline{B}'_i$, of the "share matrix" $\overline{B} = [v_1, v_2, \ldots, v_n]$ through summing up all these "sub-shares" (due to the additive homomorphism of (n, n)-secret sharing). Each CSP continues to distribute the summing result to the other CSPs and the "share matrix" $\overline{B}$ can be reconstructed by summing up all the gathered distributions. Then, the rest of reconstruction is identical to Bai's scheme [1] with an additional step for partially checking the correctness of secret reconstructed (using the "mirror matrix" S').

After correctly reconstructing the *location* and *blinding value*, the encrypted file chunks can be found and sent back by each CSP [3,4]. The user groups these chunks according to the unique identifier of file and recovers the whole encrypted files with MDS code. The original files can be derived through decryption.

We now elaborate on the detailed steps of the retrieval protocol. Note that in all the steps, whenever an entity fails or any verification step fails, the entity sends the signal "fail" to the other entities and aborts. Moreover, whenever any entity receives a signal "fail", it aborts as well.

Step R1: Given the master key msk and the query keyword w, the user first builds a secret matrix and its mirror matrix.

(1) Build $(m \times t)$ secret matrix S such that $S[1][1] = \mathsf{P}(msk_1, w)$, $S[1][2] = \mathsf{Q}(msk_2, w)$ and random values are filled at the other entries. We can just set $m = 2t - 2$ to reduce communication overload.
(2) Build a mirror matrix S' as the same as S except $S'[1][1] = S'[1][2] = 0$. Then, S' is published out for correctness check in future.

Secondly, the user performs the following computations to make a multiple secret sharing on the secret matrix S.

(3) After randomly picking a $(m \times t)$ matrix A of rank t, compute the projection matrix $M = A(A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}} \mod p$ and publish the reminder matrix $R = S - M \mod p$ where p is a public big prime number.
(4) Randomly choose $(t \times 1)$ vectors x_i for $i = 1, 2, \ldots, n$ such that any t of $\{x_i\}_{i=1}^n$ are linearly independent, and compute each share $v_i = Ax_i \mod p$.

Finally, user submits the share v_i to the ith CSP for $i = 1, 2, \ldots, n$, to retrieve all the files containing the keyword w.

Step R2: For the ith CSP, $i = 1, 2, \ldots, n$, upon receiving the share v_i, it first submits and collects shares through multi-party computation according to the following steps:

(1) Build a matrix $\overline{B}_i$ such that $\overline{B}_i[i] = v_i$ and the other entries of $\overline{B}_i$ are filled with 0.
(2) After making an (n, n)-secret sharing on $\overline{B}_i$, i.e., randomly pick $\overline{B}_{i1}, \ldots, \overline{B}_{in}$ such that $\sum_{j=1}^n \overline{B}_{ij} = \overline{B}_i$, send $\overline{B}_{ij}$ to the jth CSP for $j = 1, \ldots, i-1, i+1, \ldots, n$. Note that the matrix $\overline{B}_{ii}$ is kept at local by the ith CSP.
(3) Upon receiving $\overline{B}_{ji}$ from the other CSP, where $j = 1, \ldots, i-1, i+1, \ldots, n$, send back (to the jth CSP) a response ack. Note that if the response is not received by the jth CSP, the jth CSP needs to set $\overline{B}_{jj} = \overline{B}_{jj} + \overline{B}_{ji}$.
(4) Suppose $\overline{B}_{1i}, \ldots, \overline{B}_{(i-1)i}, \overline{B}_{(i+1)i}, \ldots, \overline{B}_{ni}$ have been successfully gathered and responded. Compute and broadcast $\overline{B}_i = \sum_{j=1}^n \overline{B}_{ji}$ ($\overline{B}_{ii}$ is the local share computed by ith CSP) to all the other active CSPs (i.e., the CSPs which have successfully sent back valid response before).
(5) After gathering $\overline{B}_1, \ldots, \overline{B}_{i-1}, \overline{B}_{i+1}, \ldots, \overline{B}_n$ from the other CSPs and $\overline{B}_i$ from local, the ith CSP computes and obtains the share matrix $\overline{B} = \sum_{j=1}^n \overline{B}_j$.

Then, the ith CSP attempts to reconstruct the secret matrix S as follows:

(6) Randomly collect any t columns from $\overline{B}$ and construct the matrix B.

(7) Calculate projection matrix $M = (B(B^{\mathrm{T}}B)^{-1}B^{\mathrm{T}}) \mod p$ and the secret matrix can be reconstructed as $S'' = M + R \mod p$.
(8) Verify the correctness of reconstruction by checking all the entries except $S''[1][1]$ and $S''[1][2]$ of S'' and S'. If not passed, return back to Step R2.(6).

Step R3: Upon computing $P = S''[1][1]$ and $Q = S''[1][2]$, the ith CSP proceeds as follows to collect and return the code chunks:

(1) Compute $\mathsf{T}[Q] \oplus P = tmp$ and parse tmp as loc and k. Then, loc is the location of the first node of l_w in A and k is the symmetric key used for the encryption of this node.
(2) Compute $info = \mathsf{SKE2.Dec}(k, \mathsf{A}[loc])$.
(3) After parsing $info$ as id, loc and k, fetch the code chunks (id, c_i) with $(\mathsf{id}, c_i) \in \mathbf{c}_i$. Then, test $loc||k$: if $loc||k \neq 0^{\lambda + \log_2 r}$, return back to Step R3.(2).

After gathering all the code chunks $\{(\mathsf{id}, c_i)\}_{\mathsf{id}\in\Gamma}$ for $i = 1, 2, \ldots, n$, where Γ is an underlying set of file identifiers satisfying the search criterion as intrinsically indicated above, the ith CSP sends back results $(i, \{(\mathsf{id}, c_i)\}_{\mathsf{id}\in\Gamma})$.

Step R4: Upon receiving the results, the user continues to proceed as follows.

(1) Suppose $\overline{\Omega}$ is the set of CSP identifiers i, the chunks of which have been successfully received. If $|\overline{\Omega}| < t$, user reports "fail" and the protocol is aborted. Otherwise, he/she randomly selects t-element set $\Omega \subseteq \overline{\Omega}$, and constructs a matrix E from the corresponding t row vectors of $\overline{E}$. Recall that $\overline{E}$ is the encoding matrix of encrypted files maintained by user. The rank of $\overline{E}$ is t, which guarantees that E is invertible. Straightforwardly, for each file, its encrypted form can be reconstructed by multiplying the inverse matrix of E with the corresponding code chunks.
(2) Finally, user uses msk_4 to decrypt the reconstructed encrypted files and obtains the search results in plain.

Finally, it is 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.

4 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.

Acknowledgments. Jingwei Li's work was funded by China Scholarship Council. Dan Lin's work was funded by the National Science Foundation (NSF-CNS-1250327 and NSF-DGE-1433659). Chunfu Jia's work was supported by grant National Key Basic Research Program of China (Grant No. 2013CB834204), National Natural Science Foundation of China (Grant No. 61272423), Natural Science Foundation of Tianjin, China(Grant No. 14JCYBJC15300).

References

1. Bai, L., Zou, X.: A proactive secret sharing scheme in matrix projection method. Int. J. Secur. Netw. **4**(4), 201–209 (2009)
2. Chen, H.C.H., Lee, P.P.C.: Enabling data integrity protection in regenerating-coding-based cloud storage. In: Proceedings of the 31st IEEE International Symposium on Reliable Distributed Systems, pp. 51–60 (2012)
3. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: ACM Conference on Computer and Communications Security, pp. 79–88 (2006)
4. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: ACM Conference on Computer and Communications Security, pp. 965–976 (2012)
5. Li, J., Wang, Q., Wang, C., Cao, N., Ren, K., Lou, W.: Fuzzy keyword search over encrypted data in cloud computing. In: IEEE Conference on Information Communications, pp. 441–445 (2010)
6. Li, J., Li, J., Chen, X., Liu, Z., Jia, C.: Privacy-preserving data utilization in hybrid clouds. Future Gener. Comput. Syst. **30**(1), 98–106 (2014)
7. Owens, D.: Securing elasticity in the cloud. Commun. ACM **53**, 46–51 (2010)
8. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: IEEE Symposium on Security and Privacy, pp. 44–55 (2000)
9. Zhao, X., Li, L., Xue, G., Silva, G.: Efficient anonymous message submission. In: 31th IEEE International Conference on Computer Communications, pp. 2228–2236 (2012)
10. Zhu, Y., Hu, H., Ahn, G.J., Yu, M.: Cooperative provable data possession for integrity verification in multicloud storage. IEEE Trans. Parallel Distrib. Syst. **23**(12), 2231–2244 (2012)