Volume 4, Issue 3, March 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

An Efficient File Storage Implementation in Cloud based on Suffix Trees for Encrypted Data Access

Lija Mohan*
Research Scholar, CUSAT.
India

Dr. Sudheep Elayidom M.
Associate Professor, CUSAT.
India

Abstract— Cloud computing provides an efficient solution for data service outsourcing, where data owners can avoid committing large capital expenditures by economically storing their data to public data centers for the convenient management of data storage and utilization. Despite the tremendous benefits, outsourcing data to the commercial public cloud also opens the door for unsolicited data access in the cloud and beyond. Thus, enabling a secure and effective cloud data utilization service is of paramount importance. Given the large number of data users and huge amount of outsourced cloud data, this problem is particularly challenging as it is extremely difficult to also meet the practical requirements of performance, system usability, and high-level user searching experiences. This article investigates these challenges and in particular defines the problem of how to search over encrypted cloud data, which aims at accommodating various typos and representation inconsistencies in different user searching input for acceptable system usability and overall user searching experience, while protecting keyword privacy. In order to further enrich the application spectrum, we also demonstrate how the existing techniques can be made more efficient by applying suffix trees for similarity search, a fundamental and powerful tool that is widely used in information retrieval. We describe the challenges that are not yet met by existing techniques, and discuss the research directions and possible technical approaches for these new search functionalities to become a reality.

Keywords— Cloud Computing, privacy, suffix tree, similarity search

I. INTRODUCTION

Cloud computing is the long dreamed vision of computing as a utility, where cloud customers can remotely store their data into the cloud so as to enjoy the on-demand high-quality applications and services from a shared pool of configurable computing resources [1]. The benefits brought by this new economic computing model include, but are not limited to relief of the burden for storage management, universal data access with independent geographical locations, and avoidance of capital expenditure on hardware, software, and personnel maintenances[1].

As the data produced by individuals and enterprises that need to be stored and utilized (e.g., email, personal health records, photo albums, tax documents, financial transactions) is rapidly increasing, data owners are motivated to outsource their local complex data management systems into the cloud for its great flexibility and economic savings. To protect data privacy and combat unsolicited accesses in the cloud and beyond, sensitive data may have to be encrypted by the data owners before outsourcing [2]; this, however, makes obsolete the traditional data utilization service based on plaintext keyword search. The trivial solution of downloading all the data and decrypting locally is clearly impractical, due to the huge amount of bandwidth cost in cloud-scale systems. Moreover, aside from eliminating local storage management, storing data into the cloud serves no purpose unless they can easily be searched and utilized. Thus, exploring privacy-assured and effective search service over encrypted cloud data is of paramount importance.

Considering the large number of on-demand data users and huge amount of outsourced data files in the cloud, this problem is particularly challenging as it is extremely difficult to also meet the requirements of practical performance and acceptable system usability. In particular, the large number of data users demands fuzzy keyword search to support various typical user searching behaviors and typing habits for acceptable system usability and user searching experiences, while protecting keyword privacy. As common practice, users may search and retrieve the data of their respective interests using any keywords they might come up with. It is quite common that a user's search input might not exactly match those preset keywords due to possible typos, such as Illinois and Ilinois, representation inconsistencies, such as PO BOX and

P.O. BOX, synonyms such as William and Bill, typing habits such as preconfiguration and pre-configuration, and/or lack of exact knowledge about the data. To give a concrete example, statistics from Google (http://www.google.com/jobs/britney.html) shows that only less than 77 percent of the users' searching input exactly matched the name of "Brityney," detected in their spelling correction

system within a three-month period. Therefore, enabling fuzzy keyword search [13] service that aims at accommodating various typos and representation inconsistencies in different user search inputs is of crucial importance for high system usability and overall user search experience. Although fuzzy keyword search is facilitated in modern plaintext search

engines (Google, Bing etc.), this challenging problem remains open in the cloud because of the inherent security and privacy obstacles.

In the literature, searchable encryption (e.g., [3, 4], to list a few) is a helpful technique that treats encrypted data as files labeled with keywords, and allows a user to securely search over it through predefined keywords and retrieve files of interest. However, directly deploying these techniques for secure large-scale cloud data search services would not necessarily be adequate, as they are developed as crypto primitives without considering high service-level requirements and in particular the fuzzy search functionality at all (detailed explanation later). Secure data utilization has also been studied in database outsourcing scenarios. Various approaches (not necessarily encryption-based) have been proposed [5] to enable private database queries, which on the other hand are quite different from the secure keyword search problem we investigate here. A good survey on protecting outsourced databases can be found in [5]. Note that a recent breakthrough on fully homomorphic encryption [6], which allows one to compute arbitrary functions over encrypted data without decryption, can theoretically be adopted in achieving a solution to the fuzzy keyword search problem to be introduced in this article. But its performance is far from practical at least in a foreseeable future [7].

To address the aforementioned challenges and boost large scale cloud service adoption, in this article we propose to explore fuzzy keyword search for encrypted cloud data utilization

service. We are proposing an efficient method to implement such fuzzy keyword searches using suffix tree based approach in cloud computing environment.

We describe the challenges that are not yet met by existing searchable encryption techniques, and discuss the research directions and possible technical approaches for this new search functionality to become a reality. In order to further enrich the spectrum of secure cloud data utilization services, we also study how the notion of fuzzy search can be utilized to support similarity search, which is widely used in information retrieval, database cleaning, and even biological sequence analysis [8]. The overall service design will provide built-in security assurances of keyword privacy and data file confidentiality at an acceptable cost and benefit cloud customers who seek to utilize the cloud with strong privacy guarantee. In addition, it is also expected to enable cloud service providers to securely and effectively deliver value from the cloud infrastructure to their enterprise customers, significantly encouraging the adoption of the cloud.

The organization of this article is as follows. We first present the problem setting via describing the high-level architecture of data utilization services in cloud computing. We give an overview of the relevant building blocks. We then describe in more detail the problem of fuzzy keyword search, its natural support of similarity search, and outline the possible technical approaches as well as future research directions on secure and effective cloud data utilization. Finally, we summarize the article with concluding remarks.

II. CLOUD SERVICE ARCHITECTURE

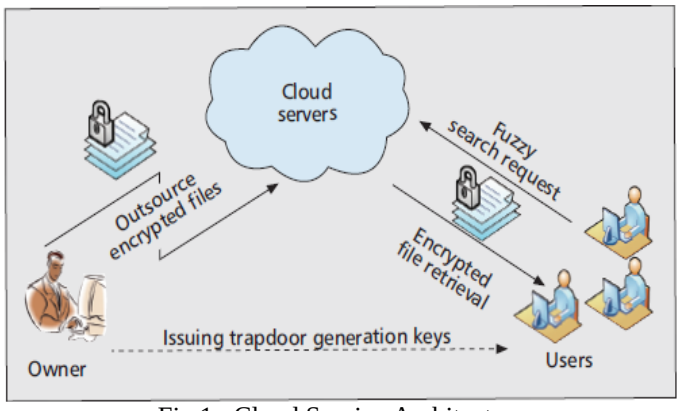

Fig 1 : Cloud Service Architecture

We consider a high-level architecture for cloud data utilization services illustrated in Fig. 1. At its core, the architecture consists of three entities: *data owner*, *user*, and *cloud server*. Under the cloud paradigm, the data owner may represent either an individual or enterprise customer, who relies on the cloud server for remote data storage and maintenance, and thus is relieved from the burden of building and maintaining local storage infrastructure. Assume the data owner has a collection of n data files C = ($F1$, $F2$, …, FN) to be stored in the cloud server, where a predefined set of distinct keywords in C is denoted as W = {$w1$, $w2$, ..., wp}. To keep sensitive data confidential from unauthorized entities, cryptographic methods have to be applied to data collection C by the owner before outsourcing so that the data confidentiality can be assured in the cloud and beyond. To initialize the service, the data owner will distribute search request (trapdoor) generation keys *sk* to authorized users, such as partners in a collaboration team or employees in the enterprise organization.

Here, we assume the authorization between the data owner and users is appropriately done. To securely search the file collection for a given keyword *w*, an authorized user uses the trapdoor generation key *sk* to generate a search request *Tw*, and submit it to the cloud, which then performs the search over the data file collection C without decryption and sends back all encrypted files containing the specific keyword *w*, denoted FID*w*. Note that the trapdoor of a keyword *w* can be realized by applying a keyed one-way function f(x) over the keyword *w* with a secret key *sk* as $Tw = f(sk, w)$, which is

similar to [3, 4]. Hence, a trapdoor can be used as a secure search request while protecting keyword privacy. For high level system usability and user search flexibility, fuzzy keyword search has to be enabled in the cloud data utilization service.

Fuzzy keyword search supports various user searching input and/or typing habits by incorporating both minor typos and format inconsistencies. The cloud server is expected to

return the matching files when users' searching inputs exactly match the predefined keywords or the closest possible matching files based on keyword similarity metrics, when exact match fails.

Throughout this article, we consider an "honest-but-curious" cloud server in our service architecture model, which is consistent with most of the existing searchable encryption

schemes. Note that this is different from the security threats in the traditional server-client architecture, where servers are usually considered trustworthy. Here we assume the cloud server only acts in an "honest" fashion to correctly follow the designated protocol specification, but is "curious" to infer and analyze the message flow received during the protocol so as to learn additional information. Thus, from the security point of view, the outsourced data and searched keywords must be protected from the cloud.

III. BUILDING BLOCKS

To implement the above secure and effective cloud data utilization service, we first investigate some relevant building blocks in this section:

Edit Distance and Fuzzy Keyword Search:

Edit distance is one of the quantitative measurements of string similarity, and in this article it serves our purpose to formulate a fuzzy search problem. The edit distance ed(*w*1, *w*2) between two words *w*1 and *w*2 is the minimal number of operations required to transform one of them into the other [10].

The three primitive operations are:

- Substitution: changing one character to another in a word
- Deletion: deleting one character from a word
- Insertion: inserting one character into a word

Using edit distance, the fuzzy search problem can be formulated as follows: given a collection of *N* encrypted data files C = (*F*1, *F*2, …, *FN*), a predefined set of distinct keywords *W* = {*w*1, *w*2, ..., *wp*}, a word *w* in the searching input and a specified edit distance *d*, the execution of fuzzy keyword search should return a set of files possibly containing the word *w*, denoted as FID*w*: if *w* = *wi* OE *W*, return {FID*wi*}; otherwise, if *w* oe OE *W*, return {FID*wi*}, where ed(*w*, *wi*) £ *d*.

IV. TOWARDS SECURE AND EFFECTIVE CLOUD DATA UTILIZATION

In this section, we primarily present possible approaches to address the challenges of fuzzy keyword search over encrypted cloud data.

Fuzzy Keyword Search over Encrypted Cloud Data:

Challenges and Possible Approaches — As stated above, fuzzy keyword search aims at accommodating various typos and representation inconsistencies in different user searching inputs and is thus of crucial importance for high system usability and the overall user search experience. However, supporting fuzzy keyword search over encrypted cloud data is quite challenging due to the inherent security and privacy obstacles: any two words that are approximately equal to each other would no longer be so after their one-way cryptographic transformation (e.g., by pseudorandom functions or hash functions), as required for encrypted keyword search. This also explains why traditional SSE schemes that conduct search only via equality test among the provided trapdoors and the encrypted searchable index do not support fuzzy search functionality.

To address this challenging problem, one possible way is through building up a fuzzy keyword set that incorporates not only the exact keywords but also those that differ slightly due to minor typos, format inconsistencies, and so on. Based on the resulting fuzzy keyword sets, the fuzzy searchable index can be conducted in a similar way as is a traditional SSE scheme. However, given a keyword *wi* and the similarity threshold *d*, directly generating the fuzzy keyword set, denoted *Swi,d*, by simply enumerating all possible words {*wi*¢} satisfying the similarity criteria ed(*wi*, *wi*¢) £ *d* would not work efficiently, because of the impractical size of the resulting fuzzy keyword set. Consider the following 26 listed variants after only one substitution operation on the first character of keyword CASTLE: {AASTLE, BASTLE, DASTLE, …, YASTLE, ZASTLE}. Since there are 6 characters in CASTLE, the number of all variants after one substitution would be 6 ¥ 26. Similarly, the number of all variants after one insertion and

one deletion operation in this case would be 7 ¥ 26 and 6, respectively, making the total number of variants within edit distance one from the keyword CASTLE be as large as 6 X 26+ 7 X 26 + 6 + 1. Here 1 refers to the original keyword. For the general case of *Swi,d* with large *d*, its resulting set can be too huge for practical system usability.

Building a Storage-efficient Fuzzy Keyword Set :

To provide more effective fuzzy keyword set constructions with regard to both storage and search efficiency, we propose to consider only the positions of the three primitive edit operations. That is, we can use a wildcard '*' to denote all three

operations of character insertion, deletion, and substitution at any position, making a much smaller fuzzy keyword set. For example, for the keyword CASTLE with the pre-set edit distance 1, its fuzzy keyword set can be constructed as *S*CASTLE,1 = {CASTLE, *CASTLE, *ASTLE, C*ASTLE,C*STLE, …, CASTL*E, CASTL*, CASTLE*}.The total number of variants after one operation on word CASTLE can now be reduced to only 13 + 1, far less than the exhaustive enumeration approach. Generally, for a given keyword *wi* with length l, the size of *Swi*,1 will be only 2l+1+1, opposing to (2l+1) ¥ 26 + 1 obtained in the straightforward approach. And the larger the preset edit distance *d* is, the more storage overhead can be reduced.

Our preliminary analysis [10] shows that for correctness of the fuzzy search mechanism, search request for keyword *w* is now a trapdoor set {*Tw*¢}*w*¢OE{*Sw*,0, *Sw*,1, …, *Sw*,*d*}, instead of a single trapdoor as in the traditional approach. Here *Sw*,0 denotes the trapdoor of user's original input that is to be checked first by cloud servers for exact match. From the communication cost point of view, the increased size of search request can be a disadvantage, especially when the searching input length|*w*| or distance threshold *d* is large. Thus, novel approaches for constructing fuzzy keyword set are demanded to overcome this issue. One promising approach is to use the *n*-grams techniquefrom plaintext information retrieval [8] to build such a fuzzy keyword set. Here, *n*-grams denotes a contiguous sequence of *n* characters from a keyword and can be used as a specific keyword signature. However, whether this approach affects the search result correctness and at the same time maintains search keyword privacy remains to be thoroughly justified.

V. BUILDING AN EFFECTIVE SEARCHABLE INDEX USING SUFFIX TREE

While suppressing fuzzy keyword set size, the above constructions introduce another challenge: how to generate the searchable index and how to perform efficient fuzzy keyword search. For high-level performance of a cloud-scale data service system, both index size and search time efficiency have to be taken into consideration. Therefore, the intuitive way to build a searchable index is to treat every wildcard variant in the fuzzy keyword set as a real keyword with an inverted index. Upon receiving the search trapdoors, the cloud server then linearly scans the index to find possible matched entries. Such an approach is clearly not appealing for its relatively slow search time performance (linear to the number of all variants in the whole fuzzy
keyword set).

One improvement would be to utilize a Bloom filter to build the space-efficient searchable index for fuzzy search functionality. In particular, we can use a Bloom filter to represent the index for fuzzy keyword set *Swi*,*d* of each keyword *wi* with edit distance *d*. Thus, the trapdoor set {*Twi*¢}*wi*¢OE{*Swi*,0,*Swi*,1,…,*Swi*,*d*} is inserted into keyword *wi*'s Bloom filter as the index stored on the server. Compared to the intuitive scheme, this approach has the advantage of reducing the index size and enhancing the search time efficiency (linear to the number of actual keywords |*W*| in the document collection C).

To further improve the search efficiency, one way is to build a Suffix Tree-traverse searching mechanism, where a multi-way tree can be constructed for storing all fuzzy keyword elements in {*Swi*,*d*} over a finite symbol set. In particular, we can divide each trapdoor in the set {*Twi*¢}*wi*¢OE{*Swi*,0, *Swi*,1,…,*Swi*,*d*} as a collection of concatenated symbols, which are all represented as q-bit binary vector. Assuming l is the output length of one-way function *f*(x), there would be l/q symbols for each search trapdoor. The key idea here is that in the multiway tree, all trapdoors sharing a common prefix have a common symbol node. The root is associated with an empty set, and the symbols in a trapdoor can be recovered in a search from the root to the leaf that ends a trapdoor. All similar words in this suffix tree can be found via depth-first search. Such an example is given in Fig. 2.

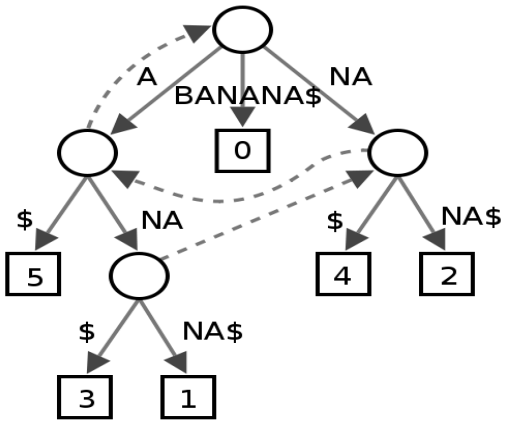

Figure 2: Suffix tree Suffix tree for the text BANANA.

Article [11] has shown that for each search trapdoor, such a searchable index can reduce the search cost on the cloud server to a constant related to trie height l/q, which has nothing to do with the number of files or the size of related key words. An experimental result for cloud side searching time cost is shown in Fig. 3. The experiment is conducted using C programming language on the Amazon EC2 cloud with sampled datasets from a Request for Comments database (RFC) (http://www.ietf.org/rfc.html). Readers are referred to [11] for further mechanism designs, detailed experiment results with different settings, as well as security reasoning.

VI. SUPPORT SIMILARITY SEARCH

Similarity search [8] is a widely used powerful tool in information retrieval, which aims at the discovery of all similar results from a data collection with respect to a given search request and distance measure. By following the formulation of the fuzzy search problem above, the similarity search can be defined as follows: given *N* encrypted data files C = (*F*1, *F*2,
… , *FN*), a predefined set of distinct keywords *W* = {*w*1, *w*2, ..., *wp*}, a word *w* in the searching input and a specified edit distance *d*, the execution of similarity search should return a set of files {FID*wi*}, where ed(*w*, *wi*) £ *d*. Given the definitions of fuzzy search and similarity search, we can see that the notion of fuzzy search naturally supports similarity search, although the purposes of the two are quite different. Recall that in the fuzzy search mechanism, the trapdoor of a user's original input (i.e., *Sw*,0) is always checked first by the cloud server for exact match. If exact match exists, all the trapdoors for the wildcard variants of search input *w* will be discarded, and only the files for exact matches will be returned.

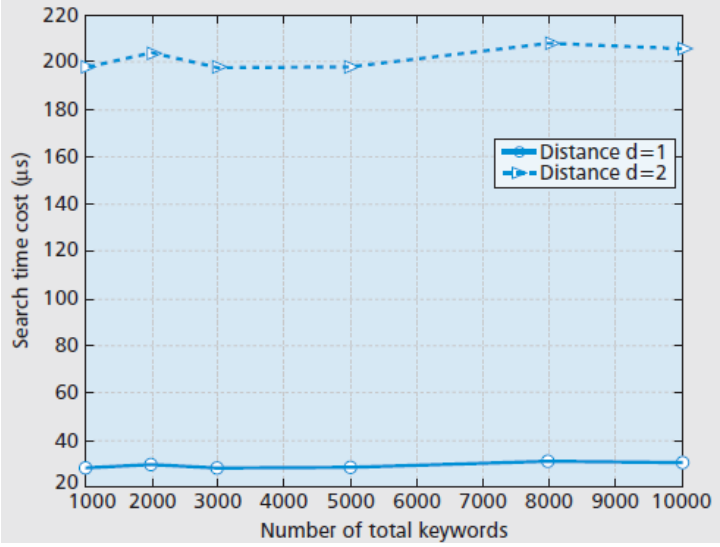

Fig 3: *An exemplary experiment result on the constant cloudside search time cost with edit distance* d *set to be 1 and 2, respectively.*

Based on this understanding, to support similarity search via the aforementioned fuzzy search mechanism, we can instruct the cloud server to always search all the received trapdoors of variants for the search input and return every matches within the searchable index (including the exact matches). As a result, the closest possible matching files based on keyword similarity metrics will always be sent back as the result for private similarity searches, even if users' original searching input exactly matches the predefined keywords. That is, data users will receive {FID*wi*}, whenever ed(*w*, *wi*) £ *d*. Some initial results of this branch of research can be found in [11]. While the similarity search functionality can be faithfully fulfilled by the aforementioned fuzzy search mechanism, it shares the same search performance and also inherits some disadvantages, such as the relatively large communication overhead from the wildcard variants of the search input. Therefore, whether there is a specific yet better scheme for private similarity search defined above can be an interesting topic for future research. For example, for improved search efficiency, B+ tree-based structures from plaintext similarity search [8] could be promising techniques to form the basis of a secure yet effective similarity search service over encrypted cloud data.

VII. CONCLUSION

In this article, we focus on the utilization of encrypted cloud data with practical system usability and high-level user searching experience. We have investigated these challenges, defined the problem of fuzzy keyword search over encrypted cloud data, and further studied its natural support of similarity search in information retrieval. We have outlined several methods like bloom filters, suffix trees etc that can be used for implementing this fuzzy keyword search.

REFERENCES:

[1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, US Nat'l Inst. of Science and Technology, 2011.
[2] Cloud Security Alliance, "Security Guidance for Critical Areas of Focus in Cloud Computing," http://www.cloudsecurityalliance.org, Dec. 2009.
[3] R. Curtmola *et al.*, "Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions," *Proc. ACM CCS*, 2006, pp. 79–88.
[4] D. Boneh *et al.*, "Public Key Encryption with Keyword Search," *Proc. EUROCRYPT*, 2004, pp. 506–22.
[5] P. Samarati and S. De Capitani di Vimercati, "Data Protection in Outsourcing Scenarios: Issues and Directions," *Proc. ASIACCS*, 2010, pp. 1–14.
[6] C. Gentry, "Fully Homomorphic Encryption Using Ideal Lattices," *Proc.STOC, 2009*, pp. 169–78.

[7] —, "Computing Arbitrary Functions of Encrypted Data," *Commun.ACM*, vol. 53, no. 3, 2010, pp. 97–105.

[8] Z. Zhang *et al.*, "Bedtree: an All-Purpose Index Structure for String Similarity Search based on Edit Distance," *Proc. SIGMOD*, 2010, pp. 915–26.

[9] S. Kamara and K. Lauter, "Cryptographic Cloud Storage," *Proc. Finan-cial Cryptography Wksps., Jan. 2010, pp. 136–49.*

[10] J. Li et al., "Fuzzy Keyword Search Over Encrypted Data in Cloud Computing," Proc. IEEE INFOCOM, Mini-Conf., 2010, pp. 441–45.

[11] C. Wang et al., "Achieving usaBle and Privacy-Assured Similarity Search Over Outsourced Cloud Data," Proc. INFOCOM, 2012, pp. 451–59.

[12] J. Sun et al., "Hcpp: Cryptography based Secure EHR System for Patient Privacy and Emergency Healthcare," Proc. ICDCS, 2011, pp. 373–82.

[13] Kui Ren, Cong Wang, Qian Wang, "Toward Secure and Effective Data Utilization in Public Cloud" , IEEE Network,Novemeber-December 2012.

