Encrypted Searching with Adaptive Symmetric Searchable Encryption Security in Cloud Storage

Mingchu Li[1, a], Wei Jia[2, b], Cheng Guo[*3, c], Lieran Zhang[4, d]

[1] *School of Software Technology, Dalian University of Technology, Dalian, 116620, China*

[2] *School of Software Technology, Dalian University of Technology, Dalian, 116620, China*

[3] *School of Software Technology, Dalian University of Technology, Dalian, 116620, China*

[4] *Department of Maintenance, State Grid Liaoning Liaoyang Electric Power Supply Company, Liaoyang, 111099, China*

[a]*mingchul@dlut.edu.cn,* [b]*jiawei19891011@163.com,* [*c]*guocheng@dlut.edu.cn,* [d]*zhanglieran_sg@163.com*

Abstract.

The insecurity of cloud storage is notorious, it is difficult to both maintain privacy of client's data while still providing the ability to retrieval useful information. In the area of searchable encryption, many previous works presented their constructions based on non-adaptive security definition raised by R.Curtmola. In this paper, we propose an efficient scheme that is secure against more sophisticated server based on adaptive security definition. Furthermore, we combine some components of currently open-source search engine with our scheme to complete a prototype and provide results from experiments on a large-scale dataset to prove the availability and efficiency of our scheme.

Keywords: Cryptographic Cloud Storage; Searchable Encryption; Adaptive Security

Introduction

As cloud storage is increasing in popularity, more and more people choose to outsource data to cloud storage server, which brings many benefits for clients such as accessing data from anywhere and never worrying about the backups. The storage servers do not ensure the confidentiality, so data has to be encrypted before outsourcing. However, the classical cryptographic primitives prevent even the data owners from retrieving documents according to given keywords. Nowadays, many researchers study the searchable encryption to solve the problem.

The concept of searchable encryption was first introduced by Song, Wagner and Perrig in[1]. It can be classified into two categories: symmetric searchable encryption(SSE) and asymmetric searchable encryption(ASE). SSE is a scheme that who searches over the data is also the one who generates it. The latest security definitions for SSE were proposed by R.Curtmola[2]. In that paper, they proposed two security definitions: Non-Adaptive indistinguishability security and Adaptive indistinguishability security. More succinctly, the cloud server of adaptive security can statistic history information including trapdoors and results to perform more sophisticated attacks especially statistical attack. Most existing schemes, [3], [4], [5], are all based on the non-adaptive security definition, so they have no ability to against such powerful servers.

In this paper, we propose an efficient encrypted search scheme that is satisfied with the adaptive security definition in[2]. In order to confuse the cloud server, we use the bilinear map to generate different trapdoors even using same keyword during search phase. Furthermore, we modify an open-source search engine with our solution to complete a prototype system and provide results from experiments.

Preliminaries

Let $\Delta = \{w_1, w_2, ..., w_d\}$ be a dictionary of d words, and 2^{Δ} the set of all possible documents. Let $\mathrm{D} \subseteq 2^{\Delta}$ be a collection of n documents $\mathrm{D} = (D_1, ..., D_n)$. Let $id(D)$ be the identifier of document *D*. let $\mathrm{D}(w)$ be the collection of all document in D that contains the word *w*, then the outcome of

a search for keyword *w* should be the identifiers of documents in $\mathrm{D}(w)$. In addition, the function $\Psi_x(w)$ is used to encrypt the keyword *w*.

A bilinear map is a map $e : G \times G \to G_T$, where *G* is a Gap Diffie-Hellman group and G_T is another multiplicative cyclic group of prime order *p* with the following properties [6]: (1) There exists an efficiently computable algorithm for computing *e*. (2) For all $h_1, h_2 \in G$ and $a, b \in \mathbb{Z}_p$, $e(h_1^a, h_2^b) = e(h_1, h_2)^{ab}$.

Our Scheme

In this section, we propose our scheme of SSE. Before the client sends encrypted document collection to cloud storage server, our system perform pre-processing operations to build encrypted indexes with document collection. We modify the inverted index to establish our encrypted indexes, then sending the indexes along with encrypted documents to server. When client want to query a keyword, the system generate a trapdoor using keyword, secret key of client and a random number, then the server can search the documents containing given keyword with the trapdoor and return the identifiers of documents. The whole search procedure involves just one round of communication.

Our scheme of SSE contains four polynomial-time algorithms: Keygen(), BuildIndex(), Trapdoor() and Search().

1) $\mathrm{Keygen}(1^s)$ is a probabilistic key generation algorithm that is run by the client to setup the scheme. It takes a security parameter *s* and returns secret and public keys.
2) $\mathrm{BuildIndex}(SK, \mathrm{D})$ is a probabilistic algorithm run by the client to generate encrypted indexes. It takes a secret key *SK* and document collection D as inputs and returns an index as output. The structure of the index is a encrypted inverted index.
3) $\mathrm{Trapdoor}(SK, w, r)$ is still a probabilistic algorithm, that is different from other systems. It runs by the client to generate the trapdoor T_w for given keyword *w*. It takes a secret key *SK*, keyword *w* and a random number *r* as inputs and returns a trapdoor T_w as output.
4) $\mathrm{Search}(I, T_w)$ is a deterministic algorithm run by server to search for

the documents in D that contain the given keyword *w*. It takes index *I* of the document collection D and the trapdoor T_w as inputs and returns the set of identifiers of documents containing the keyword *w.*

Now let us make further explanation for our SSE scheme. In order to satisfy the adaptive security definition [2], our scheme add the random element to every trapdoor of query, that make the server has no chance to distinguish the view of one history (including the outcome of each search and the pattern of searches) from the view of other with probability non-negligibly better than $\frac{1}{2}$. For example, if a client performs two searches with the same keyword “bank”, our scheme generates different trapdoors using different random numbers. Hence, even the cloud storage server has the history information, still not knowing the client searching for the same underlying word, which protects the pattern of searches. Note that our scheme still keeps simple and efficient. The storage cost is $O(n)$, where *n* presents the number of the distinct keyword in document collection. Furthermore, we trade search time for stronger security, which brings computation costs to our system, but the search time is still relatively short, which we will discuss it in experiment section.

Now we present the details of our algorithm in Fig. 1 as follow.

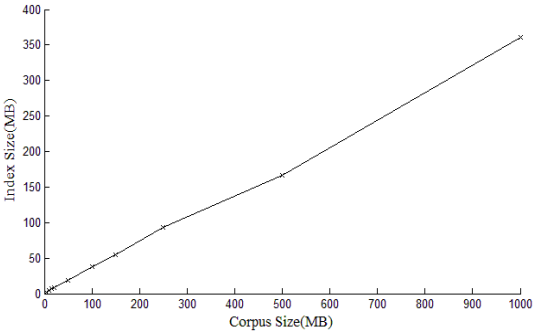
1. Keygen(1^s):
   a) Generate random secret key *SK*: $\alpha, x \leftarrow \{0,1\}^s$
   b) Generate public key *PK*: $PK \leftarrow g^{\alpha}$
2. BuildIndex(SK, D):
   a) Generate inverted index **L** from document collection **D**
   b) For each posting list L_i in **L**:
      Replace the keyword w_i of L_i with $W_i \leftarrow \psi_x(w_i)$
   c) Output index $I \leftarrow (\mathbf{L}, PK)$ and send *I* to server
3. Trapdoor(SK, w, r):
   a) Compute $M \leftarrow u^w$ and $\sigma \leftarrow (\psi_x(w) \cdot M)^{\alpha}$
   b) Generate a random number *r* and the trapdoor $T_w \leftarrow e(\sigma^r, g)$
   c) Output (T_w, M, r) and send them to server
4. Search(I, T_w):
   For each posting list L_i in the index *I*:
   a) Get the Term W_i and compute $E_i \leftarrow e((W_i \cdot M)^r, PK)$
   b) If ($T_w = E_i$):
      Output the corresponding document identifiers contained in L_i

Figure 1. Our SSE scheme algorithm

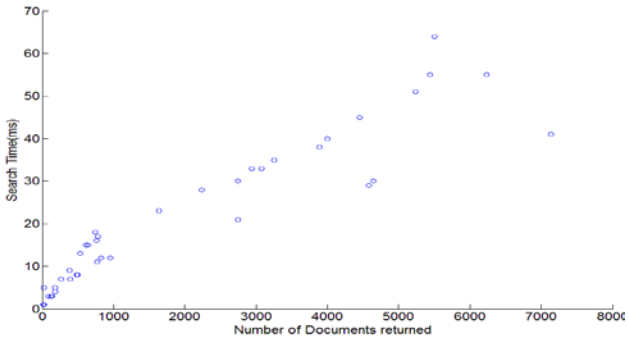
Experimental results

We modify the open-source search engine Apache Lucene[7] to build our

prototype system, which makes our system easily use the tokenizer and index builder functions. By replacing the plaintext terms of inverted indexes with ciphertext, we implant our own analyzer into the procedure of index building and searching. The dataset used for experiments is distracted from Wikimedia database and the size is over 1G. We use two common computers to organize the experiment structure, one as client and the other as server. Furthermore, when we evaluate the speed of searching, we eliminate the network delay associate with the communication between client and server, because the speed of network equipment is unstable.



(a)The relationship between corpus size and index size



(b) The relationship between returned documents and search time

Figure 2. computation and storage cost

The client needs to build the encrypted index before transmitting encrypted documents and index to the cloud storage server. Fig. 2 shows the computation and storage cost during our experiments. We use a text file whose size is over 1GB and we divide the text file into about 10,000 documents. As we can see from Fig. 2(a), the encrypted index is smaller than the original documents, which

brings low storage burden to the server. Fig. 2(b) shows that the search time is related to the number of returned documents, so it takes longer to search an extremely common keyword. However, even the returned documents achieve 60% of the whole document set, the search time still under one second, which shows the efficiency of our scheme. Note that the stop words such as conjunctions and pronouns are not included in keyword set of our experiments.

Conclusion

In this paper, we have further studied the problem of symmetric searchable encryption, which safeguards the data privacy while still offering search ability to clients. Our scheme is designed under the latest adaptive security definition, which makes our scheme can against more sophisticated attacks. Our protocol only needs one round of communication, as well as the searching speed and storage cost are all linear to the distinct words in document collection. Finally, we implement a prototype system by modifying open-source search engine and perform experiments over real-world dataset to prove the availability and efficiency of our scheme.

Acknowledgment

This research was partly supported by National Natural Science Foundation of China under grant NO. 61100194, 61272173, 61403059, 6140060, and the general program of Liaoning Provincial Department of Education Science Research under grants L2014017.

References

[1] Song, Dawn Xiaoding, David Wagner, and Adrian Perrig. "Practical techniques for searches on encrypted data." Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on. IEEE, 2000.

[2] Curtmola, Reza, et al. "Searchable symmetric encryption: improved definitions and efficient constructions." Proceedings of the 13th ACM conference on Computer and communications security. ACM, 2006.

[3] Tang, Yinqi, et al. "Phrase search over encrypted data with symmetric encryption scheme." Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on. IEEE, 2012.

[4] Chai, Qi, and Guang Gong. "Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers." Communications (ICC), 2012 IEEE International Conference on. IEEE, 2012.

[5] Kissel, Zachary A., and Jie Wang. "Verifiable Phrase Search over Encrypted Data Secure against a Semi-Honest-but-Curious Adversary." Distributed Computing Systems Workshops (ICDCSW), 2013 IEEE 33rd International Conference on. IEEE, 2013.

[6] Boneh, Dan, Ben Lynn, and Hovav Shacham. "Short signatures from the Weil pairing." Advances in Cryptology—ASIACRYPT 2001. Springer Berlin Heidelberg, 2001. 514-532.

[7]Hatcher, Erik, Otis Gospodnetic, and Michael McCandless. "Lucene in action." (2004).