DOI: 10.2298/CSIS100327029R

Study of Privacy-preserving Framework for Cloud Storage

Huang RuWei[1,2], Gui XiaoLin[1], Yu Si[1], Zhuang Wei[1]

[1]Department of Electronics and Information Engineering,
Xi'an Jiaotong University, 710049, Xi'an, China
ruweih@126.com
xlgui@mail.xjtu.edu.cn
yusiing@126.com
zhuang2978002@126.com
[2]School of Computer, Electronics and Information,
GuangXi University, 530004, NanNing, China

Abstract. In order to implement a privacy-preserving, efficient and secure data storage and access environment of cloud storage, the following problems must be considered: data index structure, generation and management of keys, data retrieval, treatments of change of users' access right and dynamic operations on data, and interactions among participants. To solve those problems, the interactive protocol among participants is introduced, an extirpation-based key derivation algorithm (EKDA) is designed to manage the keys, a double hashed and weighted Bloom Filter (DWBF) is proposed to retrieve the encrypted keywords, which are combined with lazy revocation, multi-tree structure, asymmetric and symmetric encryptions, to form a privacy-preserving, efficient and secure framework for cloud storage. The experiment and security analysis show that EKDA can reduce the communication and storage overheads efficiently, DWBF supports ciphertext retrieval and can reduce communication, storage and computation overhead as well, and the proposed framework is privacy-preserving while supporting data access efficiently.

Keywords: cloud storage, key derivation, Bloom Filter, privacy security, encrypted keyword retrieval.

1. Introduction

Cloud storage provides on-demand, scalable and QoS guaranteed storage resource, and users can operate their data anytime and anywhere. Facing the powerful and appealing advantages of cloud storage, however, a lot of people and companies are hesitant to put their data in cloud. The main reason is that people and companies are afraid of loss of control on their data. Many vocal consultants, including Gartner, have issued warning on the privacy threats in cloud storage [1]. And there are some incidents of data leakage and loss which verify people’s fears: Google’s Docs was visited by unauthorized users because of a software bug in 2009, which caused data leakage [2]; a cloud storage-provider named MediaMax went out of business in 2008 after losing 45% of client data due to an error of a system administrator [3]; criminals targeted the main cloud service provider Salesforce.com, and succeeded in stealing customer emails and addresses using a phishing attack in 2007 [4]. Therefore, to be sustainable, in-depth development, cloud storage must address the privacy concern, efficient and secure data storage and access.

1.1. Related Work

There have been many works on outsourced storage. Josh [5] developed a privacy-preserving electronic health record system. Based on symmetric and asymmetric encryption, it designed two key derivation schemes and compared the advantages and disadvantages of both. But it didn’t consider the effects of change of user access right and the dynamic operations of data which would influence the effectiveness of key derivation greatly according to the analysis of the following sections. Brian [6] formalized a model called PDAS for preserving privacy and integrity of aggregate query results. PDAS supported privacy protection by dividing the owner’s database into n sections and sending a section to a service provider. Any k ($k \leqq n$) of them can cooperate to recover the entire database, but any smaller group cannot. PDAS didn’t encrypt the data, so the service provider can get some information from partial data though it can’t get the whole database. And it demanded several service providers to cooperate, which is unrealistic. Wang [7] proposed a scheme to access outsourced data securely and efficiently. It built data index by binary tree, generated and managed keys by key derivation, dealt with the dynamics of access right and data by over-encryption and/or lazy revocation. Its shortcomings include that binary tree structure couldn’t reflect the logical relation fully by which owner organizes his data; the scheme of changing user’s access right would make other user whose access right doesn’t change to update certificate, which will bring additional communication overhead; the scheme of updating data needs to store a control block on service provider, which will occupy additional storage resource and it is uneconomical; it didn’t consider how the dynamics of access right and data influences the effectiveness of key derivation; it didn’t think about the collusive attacks in which revoked users cooperate with a service provider.

All of the above researches didn’t support ciphertext-based retrieval, namely service provider can be an agent of data owner to retrieve the owner’s encrypted data according to the user’s encrypted query, which protects the privacy of owner and user well. As service provider undertakes the jobs of retrieving data, owner can be relieved from data management, which reflects the advantage of cloud storage. There have been some works on this subject. Liu [8] proposed symmetric encryption-based ciphertext retrieval method. Song [9] proposed an asymmetric encryption-based ciphertext retrieval scheme. Yasuhiro [10] proposed a scheme based on Bloom Filter to disclose data which match a Boolean query. D.Bonech [11] developed a public key encryption method which enables Alice to provide a key to the gateway that enables the gateway to test whether a keyword is in the email without learning anything else about the email. S.Bellovin [12] proposed a search scheme based on Bloom Filters and group cipher to support ciphertext search. Wong [13] designed an asymmetric scalar-product-preserving encryption to conduct the k-nearest neighbor (KNN) computation on an encrypted database. The above works have some disadvantages: (1) don't support data sharing; (2) have heavy running overheads; (3) sometimes need a semi-trusted third party to transform query or data. Obviously, those researches can't satisfy the demands of cloud storage.

Through the above analysis, we can see that a privacy-preserving, efficient and secure cloud storage framework is needed urgently, which should resolve the following problems: data index structure, generation and management of keys, encrypted keywords retrieval, change of users' access right and dynamic operations on data, and interactions among participants.

1.2. Related Definitions

Definition1 (Key Derivation[14-15]): Data owner organizes his data in tree structure, and chooses a random key as root key, then produces sub key by the following formula: key_{child}=hash(key_{parent}||child_number), where hash() is a public hash function, and the owner only needs to store the root key. When a user asks for the access authorization, the owner will return a minimum key group from which all authorized files' keys can be derived and other unauthorized files' keys can't.

Definition2 (Bloom Filter[16]): Bloom Filter is a probabilistic data structure to test whether an element is a member of a set. A Bloom Filter is represented by an m-bit array. There are also *k* independent hash functions $h_1,h_2,\ldots, h_k$, which produce outputs that are distributed uniformly over the range [1…m].

A set $S=\{e_1, e_2,\ldots,e_n\}$ is expressed by an m-bit array, which can be realized by the following two steps:

(1) Insertion: Initially, all bits in the bit array are set to 0. To add an element e_i to a Filter, independent hash functions of the element are calculated and array bits at position $h_1(e_i)$, $h_2(e_i)$, …, $h_k(e_i)$ are set to 1. If there are *n* elements $e_1, e_2,\ldots, e_n$ in *S*, this insertion is repeated for each of the elements.

(2) Query: To determine if an element *e* belongs to the set *S*, the bits at positions $h_1(e)$, $h_2(e)$, …, $h_k(e)$ are checked. If any selected bit is 0, *e* is definitely not a member of *S*. On the other hand, if all the checked bits are 1, then *e* is considered as a member of the set.

Since Bloom Filter is a probabilistic data structure, it always has a possibility of false positive, in which *e* appears to be in *S* but actually is no. There are three key parameters which can affect the false positive rate: the number of hash functions *k*, the size of the bit array *m* and the number of keywords *n*. We will compute the false positive rate *p* by formula (1):

$$p = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^{k} \approx \left(1 - e^{kn/m}\right)^{k} \qquad (1)$$

Formula (1) is minimized when $k=(m/n)*\ln 2$, in which case it becomes:

$$p = \left(\frac{1}{2}\right)^{k} = (0.6185)^{m/n} \qquad (2)$$

Suppose the false positive rate is less than 0.01% , then k is set to more than 14 and m should be more than $20*n$. The false positive rates are shown to be tunable by careful selection of parameters.

The remainder of the paper is organized as follows. Section 2 builds the cloud storage framework and designs the interaction protocol among participants. In section 3, several key issues in the framework are discussed. In section 4, we analyze the performance of the key techniques in the framework. Finally, section 5 concludes the paper and discusses future extensions.

2. The Privacy-preserving Cloud Storage Framework

Figure 1 reflects the functional modules of data owner, user and storage service provider and the interactions among them. The dashed lines show the functional or structural correspondence of the connected parts. The interaction protocol includes five steps as following:

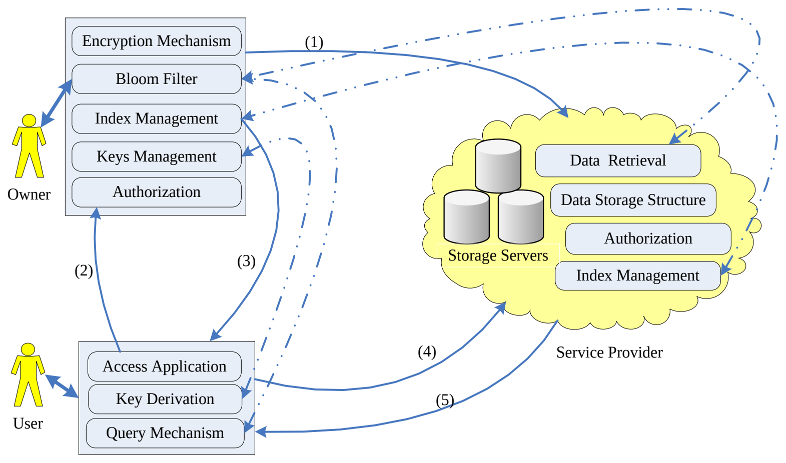

Fig. 1. The privacy-preserving cloud storage framework

(1) Owner(O) chooses a root key key_{root} for file encryption by symmetric encryption algorithm E(), a pair of keys (k_{pub}, k_{pri}) for keywords encryption of

file by asymmetric encryption algorithm AE(). Before $file_i$ is sent to Service Provider(S), owner generates the key k_i of $file_i$ by key derivation algorithm and encrypts $file_i$. Then he encrypts keywords $\{kw_1,kw_2,\ldots,kw_n\}$ by k_{pub} and generates a Bloom Filter BF_i. At last, he sends encrypted files to service provider as following:

$MSG_{OS}=\{O,S, E_{kos}(O, S, E_{k1}(file_1)||BF_1||\ldots||E_{ki}(file_i)||BF_i, t_{modified}, MAC)\}$

And k_{os} is the symmetric key between owner and service provider, $t_{modified}$ reflects the time of last update of the file, MAC(Message Authentication Code) is used to verify the integrity of message.

(2) User(U) requests access authorization from owner. And k_{uo} indicates the symmetric key between user and owner, requestId is the serial number of request:

$MSG_{UO}=\{U, O, E_{kuo}(U, O, requestId, MAC)\}$

(3) Owner verifies user's identity firstly, and searches on access control list to determine the files which can be accessed by user, then sends the minimum key group key_{min} of those files and the certificate(cert) to user. And k_{os} is the symmetric key between owner and service provider, t_{cert} indicates when the certificate is generated, and AR records the update times of the user's access right:

$cert=\{E_{kos}(U, number_{min}, t_{cert}, AR, MAC)\}$

$MSG_{OU}=\{O, U, E_{kuo}(O, U, requestId, number_{min}, key_{min}, cert, MAC)\}$

(4) User sends the certificate to service provider and asks for some files which contain the keyword *kw*. AE() indicates the asymmetric encryption algorithm which is used to encrypt keywords by owner, and k_{pub} is the public key of owner:

$MSG_{US}= \{U, S, O, requestId, AE_{kpub} (kw), cert\}$

(5) Service provider tests the certificate. If it is legitimate, service provider returns those requested files. $E_{ki}(file_i)$ is the file which is encrypted by owner, and service provider never decrypts it.

$MSG_{SU}=\{S, U, requestId, E_{ki}(file_i)||\ldots||E_{kt}(file_t), MAC\}$

User gets the encrypted files, computes the keys of the files from key_{min} by key derivation algorithm, and then decrypts the files. Of course, the files will not be encrypted if they needn't be kept secret.

3. Several Key Issues in Framework

3.1. Index Structure based on Multi-tree

Owner organizes his files in accordance with some logical relations. For reflecting the logical relations, the framework constructs the file index by multi-tree. When those files are going to be stored in the servers of service provider, the client software of owner will generate multi-tree index automatically according to their logical relation, as shown in Fig 2.

In such an index, only leaf nodes correspond to files, and non-leaf nodes represent folders or categories of files. Owner encrypts the content and name of a file and changes its' name as file_number\$$E_{kfile\_number}$(file_name), for example 1_2_1\$$E_{k1_2_1}$(Diary), before he sends the file to service provider. The pretreatment prevents service provider from knowing the content and name of the file, which protects the owner's privacy. The service provider will construct an index for every owner according the files' numbers, which can accelerate search on data.

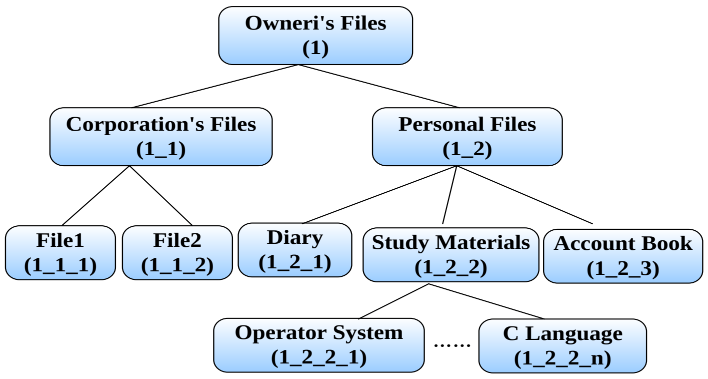

Fig. 2. The index structure of owner's files

3.2. The Extirpation-based Key Derivation Algorithm—EKDA

To have a flexible and fine-grained access control, every file has a unique key. The framework uses symmetric encryption algorithm AES to reduce the burden of encryption and decryption. But how to manage the numerous keys? Key derivation can be used to solve the problem. Owner chooses a random 128-bit key as root key, then produces subkey by the following formula: k_{child}=hash(k_{parent}||child_number), and hash() is a public hash function, for example, SHA-1. Owner only needs to store the root key, which is not only convenient to key management, but also saves the owner's storage space. When a user asks for the access authorization, the owner will return the minimum key group from which all authorized files' keys can be derived and other unauthorized files' keys can't. Key derivation can reduce the communication overhead of participants effectively.

But the effectiveness of key derivation will be harmed in some case: when the access right of a user is changed, owner must use a new key to encrypt the files if owner don't want the user to access the files again. The new key can't be computed by k_{child}=hash(k_{parent}||child_number), and the framework generates the new key by choosing a random 128-bit number. When there are a lot of files using new key or every penultimate level directory has a file using new key, the effect of key derivation is the same as the situation where key derivation is not used, namely owner must return N keys if there are N authorized files.

To solve this problem, we design an extirpation-based key derivation algorithm(EKDA): owner labels the node with “updated” which will use a new key because of the change of user access right in the index tree, and creates a new node in the update tree. The new node has the same number with the original node and has a new key. The course is shown in Fig 3. When the node needs to update again, it can change the key of the node in the update tree. When user requests access authorization, owner will compute the minimum key group by EKDA. The algorithm is as following:

```
public String EKDA(Node[] nodes,int types)
 { if(types==0){  // node[0] is updated in first time
      nodes[0].setUpdated();
      Node updatedNode=new
                   Node(nodes[0].number,keyRandom());
      updateTree.addNode(updatedNode);
      String key=updateNode.getKey();
      encrypt(file,key);
    }else if(types==1){
       // node[0] is updated in non-first time
      String key=Nodes[0].getUpdatedNode().createNewKey();
      encrypt(file, key);
    }else{   // compute the minimum key group
      String key_min="";
      Node parentNode=null;
      for(int i=0;i<nodes.length;i++){
        if(nodes[i].updated==1)

key_min=key_min+nodes[i].getUpdateNode().getKey();
        else{
          parentNode=findParentNode(i,nodes);
          if(parentNode!=null){
            key_min=key_min+parentNode.getKey();
            Node[] newNodes=nextNodes(nodes,parentNode);
            String s=extirpated_keyderivation(newNodes, 3);
            key_min=key_min+s;
          }else key_min=key_min+nodes[i].getKey();
  }}}
  return key_min;
}
```

Here is an example. When owner updates the key of file 1_2_2, he will get the file from service provider and decrypt it by the old key firstly. He asks service provider to delete the file and mark the node 1_2_2 with “updated”. Then he encrypts the file’s content and name with new key of the node in the update tree, and sends the encrypted file to the service provider. When an authorized user can access the files under the folder 1_2, the owner searches the index tree and returns the minimum key group which includes key1_2 and key1_2_2. From the effect of the algorithm, node 1_2_2 seems to be extirpated from the index tree. The algorithm can reduce the number of returned key effectively.

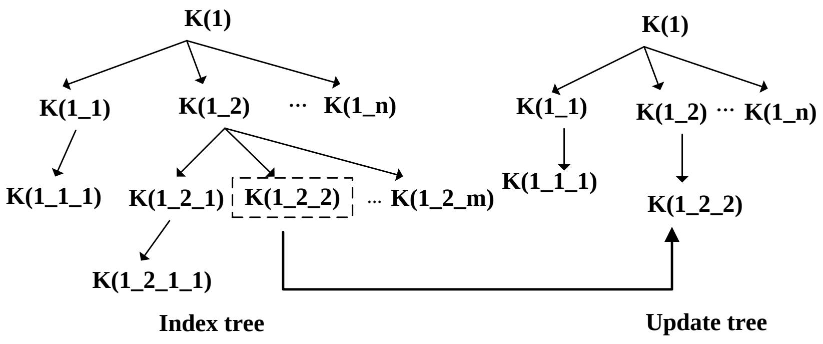

Fig. 3. The correspondence between index tree and update tree

3.3. The Double Hashed and Weighted Bloom Filter——DWBF

Bloom Filter is usually used to store a great deal of elements' information, for example, ten thousand elements. But in our framework, a file has a Bloom Filter which is used to record the keywords of the file and help ciphertext retrieving. And the number of a file's keywords is always less than 50. So the Bloom Filter in our framework is a small Bloom Filter.

Although Bloom Filter has good performance to reduce communication and storage overhead, it brings a lot of computation overhead. For example, when there are *z* keywords in a file, and there are *k* independent hash functions used in the Bloom Filter, the computation overhead to produce a file's Bloom Filter is *z*k* hash calculations; when the service provider queries a files, he need to do *k* hash calculations. So if there are large numbers of files, the computation overhead of owners and service providers is tremendous. So how to reduce the computation overhead of Bloom Filter is a key problem of ciphertext retrieval scheme in cloud.

In addition to reduce computation overhead of Bloom Filter, how to reduce the cost of false positive of Bloom Filter is another key problem. For example, when the situation of false positive occurs, 10M file and 256k file will have different effects on the communication overhead and the user's satisfaction. So, we should try to reduce the cost of false positive, and at the same time, reduce the computation overhead.

(1) Reduction in computation overhead

The computation overhead of Bloom Filter is mainly used to compute hash functions. So if we can reduce the number of hash functions in a Bloom Filter and can get the same false positive rate with the standard Bloom Filter, the computation overhead can be reduced. There are some researches on the problem. P.C.Dillinger and D.Knuth [17-19] introduced the double hashing technologies, which used two hash functions $h_1(x)$ and $h_2(x)$ to simulate more than two hash functions of the form $g_i(x)= h_1(x)+i*h_2(x)+f(i)$. A.Kirsch [21] evaluated the above methods and got the following results by theoretical analysis, as shown in formula (3):

$$\lim_{n\to\infty} p = (1-e^{-k/c})^k \tag{3}$$

which *p* is the false positive rate of Bloom Filter, *c=m/n*, and *m* is the size of the bit array and *n* is the number of keywords. That is to say, when the number of keywords tends to infinity, the false positive rate of Bloom Filter with two hashing technique is equal to that of standard Bloom Filter. But in our framework, the number of a file's keywords is less than 50, so we want to know whether two hashing techniques are useful when the number of elements is small? We designed the following experiments: we choose the following specific Bloom Filters: the standard Bloom Filter(SBF), the double hashed Bloom Filter(DBF) which $g_i(x)=h_1(x)+i*h_2(x)$, DBF2 which $g_i(x)= h_1(x)+i*h_2(x)+i^2$, DBF3 which $g_i(x)= h_1(x)+i*h_2(x)+i^3$, and DBF4 which $g_i(x)= h_1(x)+i*h_2(x)+i^4$. Then we compute value of $k\in\{\lceil cLn2\rceil, \lfloor cLn2\rfloor\}$ that minimizes $p=(1-\exp(-k/c))^k$. Next, for each of the Bloom Filters under consideration, repeat the following procedure 1000 times: instantiate the Filter with the specified values of *n*, *c* and *k*, populate the Filter with n ciphertext of keywords, and then query $\lceil 10/p\rceil$ elements not in *S*, recording the false positive rate of those Bloom Filters. The result is showed in Fig 4.

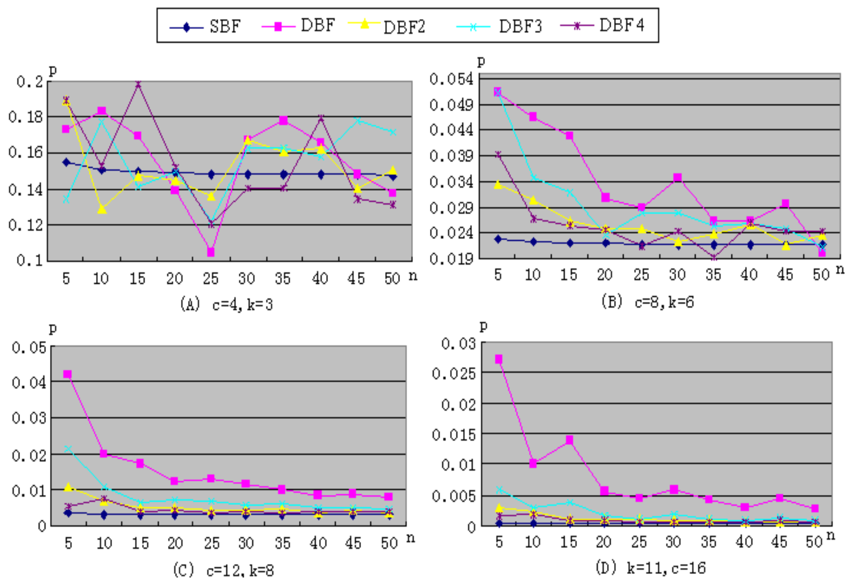

Fig. 4. The false positive rate of several Bloom Filters

From Fig 4, we can find when $k \geq 8$, the false positive rate of DBF4 is closest to that of standard Bloom Filter. So we can replace the standard Bloom Filter with DBF4 ($k \geq 8$). If there are k hash functions used in SBF, the computation overhead of DBF4 is k/2 times as small as that of SBF.

(2) Reduction in cost of false positive

There are some works on the subject. Bruck [21] incorporated the information on the query frequencies and the membership likelihood of the elements into its optimal design; Xie [22] dealt with different elements in a data set depending on their query invalidation cost by clustering elements into different baskets. Those works are suitable for the owner to manage his own files. But in our framework, the situation is different. The Owner outsources his files and the service provider couldn't know how many files will be stored in his servers, and how big the files are, and how often the files are visited before the files are stored. So their methods couldn't be fit for our framework.

So we design a simple scheme based on file's size. Of course, query frequency can be used if the owner knows. But because of data sharing, the owner may have no idea about it. Suppose service provider divides all files into three groups according to their sizes, for example, the files whose size are smaller than 1M belong to the first group, the files whose size is bigger than 1M and smaller than 64M belong to second group, and the file whose size is bigger than 64M belong to the third group. And the relation of the false positive rate of the three groups is: $p_{group1} = 2*p_{group2} = 4*p_{group3}$. Suppose k_i means how many hash functions the Bloom Filter of $group_i$ uses, we can get the result: $k_2=k_1+1$, $k_3=k_2+1=k_1+2$.

(3) The formal description of DWBF

Now, we give the formal description of DWBF(Double Hashed and Weighted Bloom Filter) as following:

$$DWBF=\{\{\{kw_{11},\ldots,kw_{1n}\},\{kw_{21},..,kw_{2m}\},\ldots,\{kw_{q1},\ldots,kw_{qz}\}\},$$
$$\{w_1,w_2,..,w_q\},\{k_1,k_2,\ldots,k_q\},\{sw_1,sw_2,\ldots sw_j\}$$
$$g_i(x)=h_1(x)+t*h_2(x)+t^4,\ t\in Z \text{ and } t\in[0,k_i)\}$$
$$(n,m,z,q,j\in N,\ k_i\in\{k_1,k_2,\ldots,k_q\} \text{ and } k_{j+1}=k_j+1(j\geq 1)).$$

The definition means there are *q* files, and every file has some keywords, w_i is the size of $file_i$, and k_i is the number of hash functions used in the Bloom filter of $file_i$; sw_i is the weight standard to divide files into j+1 groups according to file's size, for example, {1M,64M} means there are three groups, whose range is size≤1M, 1M<size≤64M, and size>64M respectively.

3.4. Change of Access Right

Service provider builds an access right updating list updateAR[Owner_id] for every owner firstly. And the node in list has two properties: node.id is the number of user, and node.times indicates how many times the access right of the user was updated. After $Owner_i$ updates the access right of $User_j$, he sends the update massage to service provider with the number of $User_j$.

Service provider receives the massage and searches the list updateAR[i]. If there is a node with node.id=j, then node.times++; otherwise service provider inserts a new node into updateAR[i] and set node.id=j and node.times=1. When $User_j$ requests files, service provider checks whether there is a node with node.id=j in updateAR[i]. If there is not such a node, service provider returns the files; if there is such a node, service provider will check whether node.times is equal to cert.AR. If node.times is equal to cert.AR, he will return the files; otherwise he will refuse to return the files and remind the $User_j$ that his certification has expired. The above operations prevent revoked user from getting files from service provider.

Of course, a revoked user can steal files when the files are transmitted. There are two methods to solve the problem: one is over-encryption [23] and the other is lazy revocation [24]. Over-encryption asks the service provider to encrypt the files before they are transmitted, which can prevent revoked user getting the files, but not all service providers are willing to provide such a service and encrypting a batch of files increases the economic burden of owner. Lazy revocation doesn't need owner and service provider to do anything before the file is updated because the stolen file is the same as the file which the revoked user had authorization to access. The framework adopts lazy revocation.

3.5. Dynamic Data Operations

Owner has three dynamic operations on data: storage, deletion and update. When owner wants to store a new file, he will find a new number from the index tree according to the logical relation and compute the key by k_{child}=hash(k_{parent}||child_number), and then encrypt the file and send it to service provider. When owner wants to delete a file, he will send a delete message to service provider to delete the file, then mark the node of the file in the index tree with "deleted". When there is a new file which wants to use the number of the deleted file, it will be treated as an updated file.

When the file is updated, the key is valid if there is not a revoked user who could access the file before. Otherwise we need to do the following operations: owner marks the node of the file in index tree with "updated", and inserts a new node with same number and new key into update tree. Then he encrypts the content and name of the file with new key, and sends the encrypted file to service provider. Suppose $t_{modified}$ indicates when the file was modified latest and t_{cert} indicates when the user's certificate was created. When a user requests the file, service provider compares $t_{modified}$ and t_{cert}, cert.AR and node.times of node whose node.id is equal to the user's number in updateAR[owner_id]. If $t_{modified}>t_{cert}$ and cert.AR==node.time, the user is an authorized user whose key is old, so service provider will return the file and remind him to get a new key; if $t_{modified}\leqq t_{cert}$ and cert.AR==node.time, the user is an authorized user who's key is new, so service provider will return the file; if cert.AR<node.time, the user is an revoked user, so service provider will refuse to return the file to him.

4. Performance Evaluation

Our research group is designing and developing a campus-level cloud computing platform named “Qing Cloud”. Based on the above framework, we developed a system of cloud storage by Java. Now we will evaluate the feasibility of the framework by analyzing the effectiveness of EKDA and DWBF, the run-time overhead of the system and privacy security.

4.1. Effectiveness of EKDA

To reflect the real cloud storage environment, the experiment simulates the interactions among multiple users, multiple owners and multiple service providers in Qing Cloud. User requests files, and owner changes users' access right and updates files randomly. Owner stores one hundred files in different organization structures in servers of service provider. Suppose the size of minimum key group of EKDA is *size1*, and the size of minimum key group of common key derivation is *size2*. By computing *size1*/*size2*, the effectiveness of extirpation-based key derivation can be verified, which is showed as Fig 5.

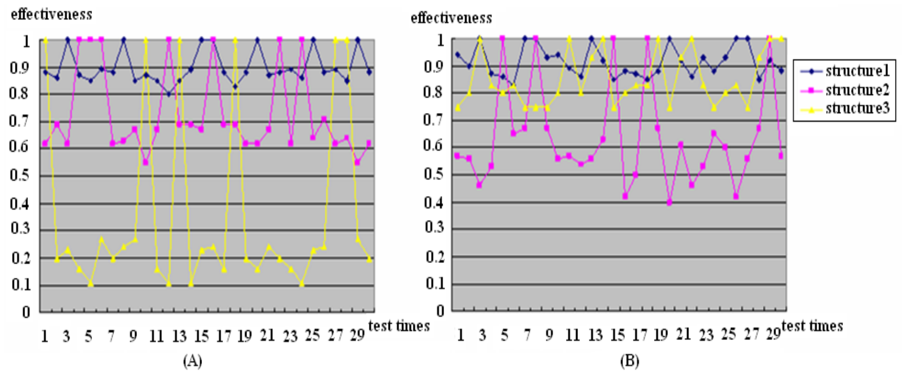

Fig. 5. The effectiveness of EKDA

Figure 5(A) shows the effectiveness when updating the same file in three different file organization structures; figure 5(B) shows the effectiveness when updating another file in the above three structures. From figure 5(A) and 5(B), we can draw the following conclusions: (1) EKDA is very effective because size1/size2≦1; (2) the organization structure of files has a direct influence on the effectiveness of the algorithm; (3) the position of the updated file has a direct influence on the effectiveness of the algorithm; (4) when a file organization structure is fixed, except those points whose value is 1, the points always surround a value. The reason is that the effectiveness of the algorithm is *(1+m)/n* if there are *n* files in a folder which has *m* updated files. So the effectiveness will fluctuate around *(1+m)/n*.

4.2. Effectiveness of DWBF

We will analyze the performance of Bloom filter from computation, communication, storage overheads and the false positive rate.

(1) Reduction in Communication overhead and false positive rate

We will compare the communication overhead and false positive rate of DWBF with those of standard Bloom filter(SBF). Suppose there are two weight standards to divide the files: (i){1M, 64M}; (ii) {256K, 1M, 64M}, and k_1=8 to all owner. There are 100 files whose size ranges from 1k to 1G according to some percentages. For example, if the number of files whose size is smaller than 1M is *n1*, bigger than 1M and smaller than 64M is *n2*, or bigger than 64M is *n3*, 4:4:4 means n1:n2:n3. Every file has ten keywords. And *p1* is the false positive rate of DWBF, *p2* is the false positive rate of SBF, *w1* is the communication overhead of DWBF produced by false positive, and *w2* is the communication overhead of SBF. We repeat the experiment 100 times, and query [10/p_{min}] elements not in the keywords of 100 files every time. The result is showed in Fig 6. We can draw the following conclusions: (1) all of ratios are less than 1, so DWBF is good at reducing false positive rate and communication overhead; (2) the more big files there is, the smaller the false positive rate is, which is shown in figure 6(A) and 6(C); (3) the more levels a weight standard has, the smaller the cost of false positive is, which is shown in figure 6(B) and 6(D).

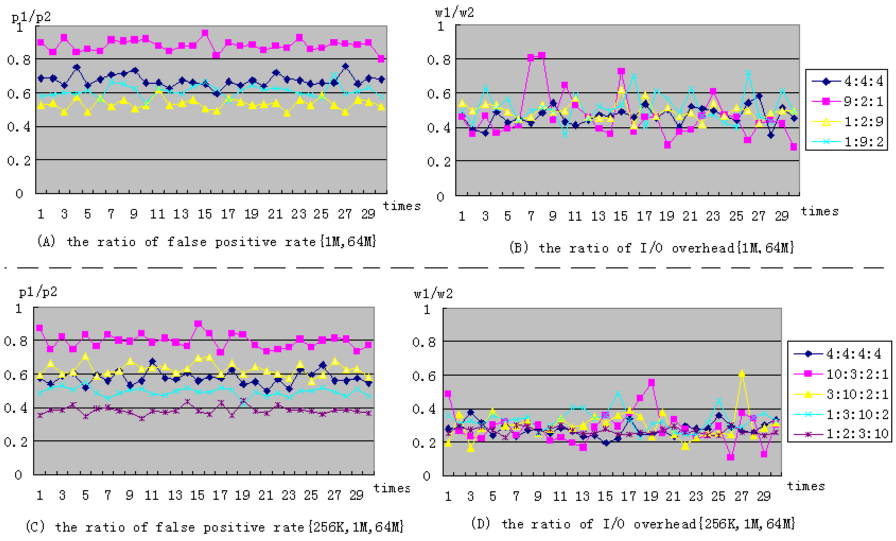

Fig. 6. The performance of DWBF compared with SBF

(2) Overhead of DWBF

In the framework, elliptic curve encryption algorithm(ECC) is used as the asymmetric encryption method which adopts 160-bit key. Owner encrypts keywords of files by ECC, and then transforms the ciphertexts of a file's keywords into a Bloom Filter. Suppose there is a file which has 10, 20, 30, 40, 50 keywords respectively, and the size and the keywords of the file is generated randomly. The result is showed in Table 1.

Table 1. Overhead of DWBF

		Storage and Communication Overhead			Computation Overhead	
num	k	ECC(bit)	DWBF(bit)	DWBF/ECC	Owner(ms)	Service Provider(ms)
10	8	2678	115	0.00043%	0.97	0.1
	9	2682	129	0.00048%	1.01	0.09
	10	2681	144	0.00054%	1.11	0.11
20	8	5367	230	0.00043%	1.97	0.1
	9	5355	259	0.00048%	1.96	0.1
	10	5370	288	0.00054%	1.98	0.1
30	8	8049	346	0.00043%	2.85	0.1
	9	8046	389	0.00048%	2.9	0.1
	10	8044	432	0.00054%	2.93	0.1
40	8	10737	461	0.00043%	3.83	0.1
	9	10733	519	0.00048%	3.89	0.1
	10	10728	577	0.00054%	3.93	0.1
50	8	13411	577	0.00043%	4.71	0.09
	9	13420	649	0.00048%	4.83	0.1
	10	13442	721	0.00054%	5.03	0.09

From Table 1, we can draw the following conclusions: (1) DWBF can reduce the communication and storage overheads greatly; (2) the computation overhead of DWBF is so small and increases so slow that it wouldn't be a burden to owner; (3) DWBF has good query performance because the computation overhead of query keeps around 0.1ms with the increment of keywords.

4.3. Run-time Overhead of the System

Run-time overhead of the whole system is measured from three aspects: communication, computation and storage overhead, as showed in Table 2.

Suppose the amount of files which is authorized to access by User$_j$ is n_j, the amount of files which satisfies User$_j$'s query is s_j; Owner$_i$ has m_i users, the size of encrypted file$_k$ is f_k and the length of its DWBF is bf_k, the high of index tree is h and the nodes in index tree occupies q bits, Owner$_i$ has p files, and the average amount of keywords of every file is g; we adopt 128-bit key, *hash()* indicates the computation overhead of hash function, *E()* and *D()* is the computation overhead of encrypting and decrypting file by symmetric encryption, *AE()* is the computation overhead of encrypting keyword by asymmetric encryption; *len* is the key amount in minimum key group generated by key derivation. The analysis is shown in Table 2. And (O) indicates that owner undertakes the overhead, the rest may be deduced by analogy.

Our framework reduces run-time overhead immensely by the following measures: (1) storage overhead of owner, communication overhead of number group and key group is reduced greatly by EKDA; (2) According to the framework, file retrieval is done by service provider instead of owner, which relieves the computation overhead of owner; (3) DWBF can reduce the storage, communication and computation overhead as well; (4) To use multi-tree structure, the length of file's serial number is shorter than or equal to the height of the tree, which reduces the communication overhead of number group.

Table 2. Run-time Overhead of the System

Type		Overhead
Communication Overhead	minimum key group	len*128
	minimum number group	len*(h/2)
	file and DWBF	$\sum_{k}^{p}{}_{1}(\mathbf{f}_k+\mathbf{bf}_k)$
Computation Overhead	ciphertext retrieval(S)	(1/2)*n_j*2*hash( )
	key derivation (O)	n_j*(1/2)*(1+h)*hash( )
	key derivation (U)	s_j*(1/2)*(1+h)*hash( )
	file and keywords encryption(O)	p*(E( )+g*(AE( )+2*hash( )))
Storage Overhead	key(O)	128
	IndexTree(O)	p*h/2*q
	IndexTree(S)	p*h/2*q*m_i
	file and bloom Filter	$\sum_{k}^{p}{}_{1}(\mathbf{f}_k+\mathbf{bf}_k)$

4.4. Privacy Security

From Fig 1, we can find there are several potential threats to users' privacy: (1) during the course of files transmitting, outside attacker can steal the files by eaves-dropping; (2) inside attacker is easy to steal the files because the files are stored in service provider's servers; (3) a malicious user and a service provider cooperate to steal the owner's files, which is called as collusive attack; (4) when the user queries, service provider may take a peep at the content of query which is the privacy of user.

Aimed at the first attack, attacker can't decrypt the file if he hasn't key. To the second attack, owner encrypts the content and name of the files by symmetric encryption, encrypts keywords of files by asymmetric encryption, and transforms encrypted keywords into BF by hash functions. So the encrypted files and BF are stored in servers of service provider. The symmetric keys are transmitted from owner to user, the private key of asymmetric keys is only known by owner, and the ciphertexts of keywords are not stored in service provider. So the insider attacker couldn’t decrypt the files and keywords.

Against the third attack, owner’s certificate limits the scope of files which can be accessed by a user. And certificate is encrypted by the symmetric key K_{os}, which just can be decrypted by owner and service provider, so it can prevent the user from retrieving other files which is out of the scope. Because of the false positive rate of Bloom Filter, service provider may return the files which don’t meet the query. But the file is in the scope of authorization, so it won’t leak owner’ privacy. When service provider is in collusion with malicious users and retrieves files which is out of the authorized scopes, service provider can find the files meeting the query, but he can’t decrypt those files because he haven’t the keys.

If service provider wants to know the content of user’s query, it can only do that by exhaust algorithm because he hasn’t the private key of owner. Support there are eighty-five letters of which a filename can be made in alphabet, when there is a five-letter keyword, it spends 30ms to encrypt a string and retrieve Bloom Filter one time by a computer with 1.86GH dual-core CPU and 2GB memory. So, 2.11 years will be spent to find out the five-letter keyword, which is considered as difficult calculation. So the privacy of users can be protected.

From the above analysis, the proposed framework does well in privacy security.

5. Conclusion

In this paper, we propose a privacy-preserving cloud storage framework, which includes an interactive protocol among participants, multi-tree index, extirpation-based key derivation algorithm(EKDA) for key management and double hashed and weighted Bloom Filter-based search on encrypted keyword(DWBF), which are combined with lazy revocation to deal with the changes of users’ access right and dynamic operations of data. The framework supports the interactions among multiple users, multiple owners and multiple service providers, but only supports owner-write-user-read. The experiment and security analysis show that EKDA can reduce the communication and storage overheads efficiently, DWBF supports encrypted keywords retrieval and can reduce communication, storage and computation overhead as well, and the proposed framework is privacy-preserving while supporting data access efficiently.

To support privacy protection further, the future works include the following aspects. First, we plan to improve the EKDA to adapt the change of access right better. Second, we are going to study the encryption techniques which support ciphertext computing. Finally, we will integrate the cloud storage system with the virtual machine system which is developed by our research group, and realize a real cloud environment which supports computation and storage.

Acknowledgments. This work was supported in part by the National Science Foundation of China under Grant No.60873071, the National Science Foundation of China under Grant No.91018011, the National High-Tech Research and Development Plan of China under Grant No.2008AA01Z410, and IBM' Shared University Research Plan.

References

1. Gartner: Assessing the Security Risks of Cloud Computing(ID Number:G0015778 2).(2008)
2. Jessica E. V.: Google Discloses Privacy Glitch(2009). [Online]. Available: http://blogs.wsj.com/digits/2009/03/08/1214/ (current June 2011)
3. Michael K.: MediaMax/The Linkup: When the cloud fails (2008). [Online]. Available: http://blogs.zdnet.com/projectfailures/?p=999(current June 2011)
4. Greenberg, A.: Cloud Computing's Stormy Side(2008).[Online]. Available: http://www.forbes.com/2008/02/17/web-application-cloud-tech-intel-cx_ag_0219 cloud.html(current June 2011)
5. Josh Benaloh, Melissa Chase, Eric Horvitz, Kristin Lauter: Patient controlled encryption: ensuring privacy of electronic medical records. In Proceedings of the 2009 ACM workshop on Cloud computing security. Chicago, IL, USA, 103-114.(2009)
6. Brian Thompson, Stuart Haber, William G. Horne, Tomas Sander, Danfeng Yao: Privacy-Preserving Computation and Verification of Aggregate Queries on Outsourced Databases. In Proceedings of the 9th International Symposium Enhancing Privacy Enhancing Technologies, Seattle, WA, 185-201. (2009)
7. Weichao Wang, Zhiwei Li, Rodney Owens, and Bharat Bhargava: Secure and Efficient Access to Outsourced Data. In Proceedings of the 2009 ACM workshop on Cloud computing security. Chicago, Illinois, USA, 55-66. (2009)
8. Qin Liu, Guojun Wang, Jie Wu: An Efficient Privacy Preserving Keyword Search Scheme in Cloud Computing. In Proceedings of the 2009 International Conference on Computational Science and Engineering. Vancouver, BC, Canada, 715-720. (2009)
9. Dawn Xiaodong Song, PDavid Wagner, PAdrian Perrig. Practical Techniques for Searches on Encrypted Data. In Proceedings of the 2000 IEEE Symposium on Security and Privacy. Berkeley, California, USA, 44-55. (2000)
10. Yasuhiro Ohtaki: Partial Disclosure of Searchable Encrypted Data with Support for Boolean Queries. In Proceedings of the 2008 Third International Conference on Availability, Reliability and Security. BARCELONA, SPAIN, 1083-1090. (2008)
11. D.Bonech, G.D.Crescenzo, R.Ostrovsky, and G.Persiano: Public-key encryption with keyword search. In Proceedings of Eurocrypt 2004. Interlaken, Switzerland, 506-522. (2004)
12. S. Bellovin and W.Cheswick: Privacy-enhanced searches using encrypted Bloom Filters[Online]. Cryptology ePrint Archive, Report 2004/022. Available: http://eprint.iacr.org/2004/022.pdf. (2004)
13. Wai Kit Wong, David Wai-lok Cheung, Ben Kao, Nikos Mamoulis: Secure kNN computation on encrypted databases. In Proceedings of the 35th SIGMOD

international conference on Management of data. Providence, Rhode Island, USA, 139-152. (2009)
14. M.J.Atallah, M.Blanton, N.Fazio, and K.B.Frikken: Dynamic and Efficient Key Management for Access Hierarchies In Proceedings of the 12th ACM conference on Computer and communications security. Alexandria, VA, USA, 190–202. (2005)
15. RuWei Huang, Si Yu, Wei Zhuang, XiaoLin Gui: Design of Privacy-Preserving Cloud Storage Framework. In Proceedings of the 9th International Conference on Grid and Cloud Computing. Nanjing,China, 128-132. (2010)
16. B.Bloom: Space/time Trade-offs in Hash Coding with Allowable Errors. Communications of the ACM, 13(7), 422-426. (1970)
17. P.C. Dillinger and P. Manolios: Bloom Filter in Probabilistic Verification. In Proceedings of the 5th International Conference on Formal Methods in Computer-Aided Design. Austin, Texas, USA, 367-381. (2004)
18. P.C. Dillinger and P. Manolios: Fast and Accurate Bitstate Verification fro SPIN. In Proceedings of the 11th International SPIN Workshop on Model Checking of Software. Barcelona, Spain, 57-75. (2004)
19. D.Knuth: The Art of Computer Programming, Volume 3: Sorting and Searching. Addison-Wesley. (1973)
20. A.Kirsch and M.Mitzenmacher: Building a Better Bloom Filter. In Proceedings of the 14th Annual European Symposium on Algorithms. Zürich, Switzerland, 1-33. (2006)
21. Jehoshua Bruck, Jie Gao, and Anxiao Jiang. Weighted Bloom Filter. The 2006 IEEE International Symposium on Information Theory. Pasadena, CA, 2304-2308. (2006)
22. Kun Xie, Yinghua Min, Dafang Zhang, Gaogang Xie, and Jigang Wen: Basket Bloom Filters for Menbership Queries. TENCON 2005. Melbourne, Australia, 1-6. (2005)
23. S.D.C.di Vimercati, S.Foresti, S.Jajodia, S.Paraboschi, and P.Samarati: Over-encryption: Management of Access Control Evolution on OutSourced Data. In Proceedings of the 33th international conference on Very large databases. Vienna, Austria, 123-134. (2007)
24. M.Kallahalla, E.Riedel, R.Swaminathan, Q.Wang, and K.Fu. Plutus: Scalable Secure File Sharing on Untrusted Storage. In Proceedings of the FAST '03 Conference on File and Storage Technologies. San Francisco, California, USA, 29-42. (2003)

Huang RuWei received BSc and MSc in Computer Science from GuangXi University in China in 2001 and 2004, respectively. From 2007, she studied as a PhD student in Xi'an Jiaotong University in China. She has been active in Network Computing, Service Computing and Network Security. She has published 3 articles in international and national journals, 3 articles in the proceedings of international conferences.

Gui XiaoLin graduated with a BSc in Computer at Xi'an Jiaotong University of China, and received MSc and PhD in Computer Science from Xi'an Jiaotong University in China in 1993 and 2001, respectively. Since joining Xi'an Jiaotong University in 1988, he has been an active researcher in Network

Computing, Network Security, and Wireless Networks. His recent research covers secure computation of open network systems (include Grid, P2P, Cloud); dynamic trust management theory; development on community network. He has published more than 80 articles in international and national journals. He was the recipient of New Century Excellent Talents in University of China in 2005.

Yu Si received BSc in Computer Science from ShaanXi University of Science and Technology in 2008. From 2008, he began a continuous academic project that involves postgraduate and doctoral study in Xi'an Jiaotong University. He has been active in Cloud Computing and Virtualization security. He has published 1 article in national journal.

Zhuang Wei received BSc in Computer Science from Xi’an Jiaotong University in 2010. From 2010, he studied as a Graduate student in Xi’an Jiaotong University in China. He has been active in Cloud Computing and Virtualization security.

Received: March 27, 2010; Accepted: January 19, 2011.