Environment reliability provision mechanism for cloud storage with dynamic searchable encryption algorithm

Fei Xiang, Li Yi, Cuilan Du, Xiaohui Liu[a] and Xiaohang Zhao

National Computer Network Emergency Response Coordination Center, 100029 Beijing, China

Abstract. Cloud storage are sometimes abandoned in consideration of data security. To solve this problem, this paper implemented the environment reliability provision mechanism (ERPM) for cloud storage with dynamic searchable encryption (DSE) algorithm, which is based on homomorphic encryption and convergent encryption. DSE ensures the confidentiality, privacy and controllability of users' data in cloud. Meanwhile, it achieves ciphertext-operation by cloud directly and efficiently without decrypting. It has sublinear search time, CKA2- security, compact indexes and the ability to add and delete files efficiently.

Keywords: cloud storage; ciphertext search; homomorphic encryption; convergent encryption.

1 Introduction

Cloud storage is a new concept that has been extended and developed on the basic of the cloud computing. It could be considered as cloud computing system centering on the storage and management of data. The merit of cloud storage is its availability (access data at any place), reliability (no worry about the data backup problem), and the mass storage capacity with a low cost. Thus cloud storage is the future of the storage service. But there still exist some disadvantages on cloud storage. Among which a key problem is the lack of security and reliability of the user's data. It is apparent that data owner is not willing to have their private data spied or thieved by in any situation. Despite that lots of individuals are willing to post the open data (web mail, calendar and pictures) on to the cloud, enterprise, government and the users would rather protect their information safe than convenient service provision and low infrastructure cost when it comes to informational safety.

The most direct way to protect user's important data is to encrypt the data. It also has solved the uncontrollable issue that the data stored in the cloud. However, since the data is encrypted and then stored in the cloud, cloud providers cannot provide traditional functions such as data search because they don't know plaintext. If there is no corresponding method to search the ciphertext, the powerful calculating function of cloud computing will decline sharply, and a very poor user experience.

An environment reliability provision mechanism (ERPM, figure 1) is designed to protect the security and privacy of the user data stored in the cloud and fix the critical confidence problem between the data owner and the cloud provider. This new mechanism will protect the storage, and meanwhile, enhance the cloud computing function. The cloud environment won't be degraded to a data storage pool because of the impracticability of the encrypted data. This initiative designed cloud storage mechanism was implemented by dynamic searchable encryption(DSE) based on optimized

[a] Corresponding author : lxh@cert.org.cn

homomorphic encryption(OHE),convergent encryption(CE) and inverted index. With the supporting of this two technology, encrypted data can be queried or updated when it keep stored in the cloud, and the index of encrypted data can resist adaptive chosen-keyword attack and the safe level is CKA2, finally, the index of encrypted data can be updated dynamically base on the modification of user files.

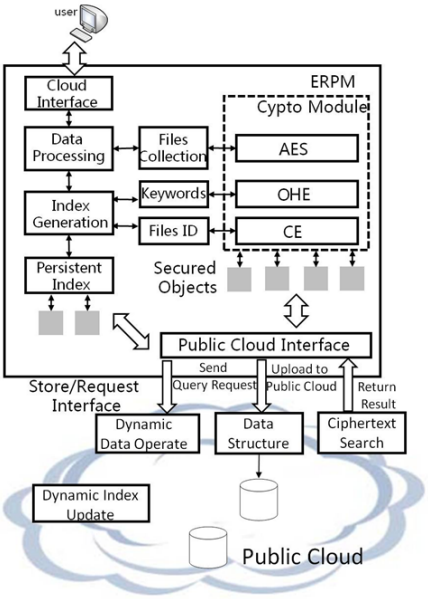

Figure 1. Construction of environment reliability provision mechanism for cloud storage.

2 Related work

Searchable Encryption(SE) first put forwarded by Song et al.[1], this scheme can search the ciphertext directly, and the search time is linearly with the length of files in the collection.

The encryption algorithm of SE scheme[2,3,4,5] which based on index takes as input an index δ and n files $\mathbf{f}=(f_1,\ldots,f_n)$ and outputs an encrypted index γ and n ciphertexts $\mathbf{c}=(c_1,\ldots,c_n)$. [2,3,4,5] can encrypt the files **f** using any symmetric or asymmetric encryption algorithm.

The general workflows of SE are: when to search for a keyword w, the user generates a search token τ_w and the server can find the identifiers $\mathbf{ID}_w$ of the files that contain w wiht τ_w, γ and **c**. User can recover correct ciphertext c_w from these $\mathbf{ID}_w$. The cloud provider learns limited information about the user's query, but it knows that whatever keyword is being searched for is contained in the files encrypted as c_w. Of course, there are ways to hide this information, for example using oblivious RAMs[6], but such an approach is inefficient.

The security notion for SE has developed over time. The first notion, security against chosen-keyword attacks (CKA1) [2,3]: (1) γ and **c** don't reveal any information about files set **f** except the number of files n and their length; and (2) γ and τ_w reveal at most the outcome of the search $\mathbf{ID}_w$. But only the search queries are independent of (γ, **c**) and of previous search results, CKA1-security provides security. So Curtmola et al. proposed the stronger notion of adaptive security against chosen-keyword attacks (CKA2) [3].

All of the CKA2-secure SE schemes[3,4,5] have limitations from a practical point of view. [3] requires $O(\#\mathbf{f})$ time to search(#**f** denotes the number of files in the collection). The encrypted index of [4] is very large. And strictly speaking, [3,4] aren't dynamic; because they can't add or delete files without either re-indexing the entire data collection. Liesdonk et al. [5] and Kamara et al. [8] have proposed SE constructions that are CKA2-secure and (explicitly) dynamic. In scheme of [5], search time is logarithmic in the number of keywords which is may be efficient enough for practical purposes. The main shortcoming of [5] is that the size of the encrypted index is relatively large (roughly the

same as [4]).Although the scheme of [7] solved the problem of index size, but its token τ_w is not probability encryption ,so there is still a certain security risk. The data structure of [7] is array, which must be defined a fixed length. If it is difficult to determine the length of the array in advance, then the array must be large enough to be suitable for storing data, which obviously means a waste of memory.

3 Key issues in framework

3.1 Preliminaries and notation

$\mathbf{f}=(f_1,\ldots,f_n)$ denotes the file collection, any file f_i is consists of the keyword collection $W=(w_1,\ldots,w_m)$. We assume that f_i has an unique identifier id(f_i). $\mathbf{f}_w$ denotes the file collection that contain keyword w, $\mathbf{f}_w \subseteq \mathbf{f}$. $\mathbf{c}=(c_1,\ldots,c_n)$ denotes the ciphertext collection of **f,** $\mathbf{c}_w$ denotes the ciphertext collection of $\mathbf{f}_w$. #**f** donates the number of files in **f**, #**f**w is the number of files that contain the keyword w, $\#\mathbf{f}_{kw}$ donates the number of keywords in file collection.

The set of all binary strings of length n is denoted as$\{0,1\}^n$, and the set of all binary strings of arbitrary length as$\{0,1\}^*$. Given a single linked list **L,** we refer to its i^{th} element as **L**[i] and to its total number of elements as #**L**. **p** denotes the pointer variable.

3.2 Optimized homomorphic encryption(OHE)

The idea of homomorphic encryption(HE) is derived from privacy homomorphism, which is first proposed by Adleman et al. [8] HE allows users to calculate the ciphertext directly, and the calculation is the same operation of the plaintext and encryption. HE has important significance for security and confidentiality of data which has been entrusted a third party to manage and operate.

Li et al. [9] has proposed an encryption algorithm both with additive and multiplicative homomorphism. And the encryption algorithm is relatively simply and usable. This paper optimizes the encryption algorithm(Optimised Homomorphic Encryption, OHE) of [9], and applies it to dynamic searchable encryption(DSE).OHE introduces a pseudo-random sequence, so the plaintexts having the same content can be different after encryption. OHE has reached the level of security requirements, improved the efficiency of data processing and security, but costed less computational overhead.

Detail as follows:

Encryption:

(1) Randomly generate 2 primes(not less than 512bit);

(2) $N=P\times Q$;

(3)Given plaintext m_i; r_i is generated by PRF F, r_i may be negative; Ciphertext $c_i=(m_i+P\times r_i) \bmod N$

Decryption: $m_i=c_i \bmod P$.

Let's prove OHE is additive and multiplicative homomorphism.

1) Additive homomorphism

We encrypt plaintexts m_1 and m_2 with encryption algorithm above, and get ciphertext $c_1=(m_1+P\times r_1) \bmod N$ and $c_2=(m_2+P\times r_2) \bmod N$

Add the plaintxets: $m_3= m_1+ m_2$;

Add the ciphertxets:$c_3 = c_1+ c_2=(m_1+ m_2+P\times(r_1+ r_2)) \bmod N$.

Decryption c_3 : $c_3 \bmod P=(c_1+ c_2) \bmod P$

$=(m_1+ m_2+P\times(r_1+ r_2)) \bmod P$

$= m_1+ m_2 =m_3$.

2) Multiplicative homomorphism

multiply the plaintxets: $m_4= m_1\times m_2$;

multiply the ciphertxets: $c_4 = c_1\times c_2=((m_1+P\times r_1)\times(m_2+P\times r_2)) \bmod N$

$=(m_1\times m_2+ m_1\times P\times r_2+ m_2\times P\times r_1+ P^2\times r_1\times r_2) \bmod N$

$=(m_1\times m_2+ P\times(m_1\times r_2+ m_2\times r_1+ P\times r_1\times r_2)) \bmod N$.

Decryption c_4 : $c_4 \bmod P= c_1\times c_2 \bmod P=((m_1+P\times r_1)\times(m_2+P\times r_2)) \bmod P$

$=(m_1\times m_2+ P\times(m_1\times r_2+ m_2\times r_1+ P\times r_1\times r_2)) \bmod P= m_1\times m_2 =m_4$.
So, OHE is both additive and multiplicative homomorphism.

3.3 Convergent encryption(CE)

Convergent encryption[10] produces identical ciphertext files from identical plaintext files, irrespective of their encryption keys. It uses own hash value to encrypt files, so ciphertext c_f is determined completely by plaintext P_f, and the cloud storage service providers can judge whether the file is the same or not without the user's key.

3.4 Build ciphertext index

Ciphertext index structure of this paper is universal inverted index [11], and the update way is merge strategy. We assume there are 3 files with 3 keywords respectively, and build the inverted index structure of plaintext as table 1.

Table 1. Inverted index of plaintext.

Dictionary	**Inverted list**		
keyword 1	id(f_1)	id(f_2)	id(f_3)
keyword 2	id(f_2)		
keyword 3	id(f_1)	id(f_3)	

We encrypt the keywords with OHE, and encrypt inverted list with CE. And we get the inverted list γ of ciphertext as table 2.

Table 2. Inverted index of ciphertext(γ).

Dictionary	**Inverted list**		
OHE(keyword 1)	CE(id(f_1))	CE(id(f_2))	CE(id(f_3))
OHE(keyword 2)	CE(id(f_2))		
OHE(keyword 3)	CE(id(f_1))	CE(id(f_3))	

3.5 Dynamic searchable encryption algorithm(DSE)

In CE, the hash function H is selected uniformly at random from the set of all mappings of strings of length m to strings of length n: $H \in_r \{h \mid h : \{0,1\}^m \to \{0,1\}^n\}$;the encryption function E is selected uniformly at random from the set of all permutations that map keys of length n and plaintext strings of length m to ciphertext strings of length m : $E \in_r \{e \mid e : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^m \wedge \forall k, pt_1, pt_2 [pt_1 \neq pt_2 \to e_k(pt_1) \neq e_k(pt_2)]\}$, k is the key, pt_1 and pt_2 are plaintexts.

We generate the DSE=(Keygen;Bulidindex;Encryption;Trapdoor;AddToken;DelToken;Add; Del; Searchindex;Decryption) with OHE and CE and pseudo-random function(PRF) R:$\{0,1\}^x\times\{0,1\}^* =\{0,1\}^x$; G:$\{0,1\}^x\times\{0,1\}^*=\{0,1\}^*$; T: $\{0,1\}^x\times\{0,1\}^*=\{0,1\}^x$. Table 3 is the fully DSE scheme.

Table 3. The fully DSE scheme.

Keygen(s,n,s'):
Input: s,n,s' (security parameter)
Step 1: use Rabin-Miller[12] algorithm to generate 2 prime K_{priv1} and K_{priv2} as private key pair which not less than s-bits;
Step 2: compute public key $K_{pub}=K_{priv1}\times K_{priv2}$, $K=(K_{priv1}, K_{priv2}, K_{pub})$ is the key of keyword;
Step 3: the key of file id(f_i) is $h_{(i)}= H(\mathrm{id}(f_i))$;
Output: K'(the encryption key of file)

Bulidindex(#$\mathbf{f}_{kw}$, K, h):
Input: **#$\mathbf{f}_{kw}$, K, h**
for $1\le i\le \#\mathbf{f}_{kw}$,
Step 1: $r_i \leftarrow$PRF $R_x(i)$, (and it is x-bits);
Step 2: let $c_{kwi} \leftarrow (kw_i+ K_{priv1}\times r_i) \bmod K_{pub}$
Step 3: for $1\le i\le \#\mathbf{f}_w$, let $c_{\mathrm{id}(fi)} \leftarrow E_{h(i)}(\mathrm{id}(f_i))$
Output: Inverted index of ciphertext γ.

Encryption(f, K'):
Input: **f**, K'
Step: for $1\le i\le \#\mathbf{f}$, $c_i \leftarrow$AES $_{K'}(f_i)$)
Output: **c**

Trapdoor(K, w):
Input: ***K, w***
Step 1:$g \leftarrow$PRF $G_x(w)$, $t \leftarrow$PRF $T_x(w)$;
Step 2:$r_w \leftarrow$PRF $R_x(w)$ (r_w may be negative);
Step 3:$I_w \leftarrow g\times t\times K_{priv2}$;
Output: $\tau_w \leftarrow (w+ K_{priv1}\times r_w) \bmod K_{pub}$

Searchkw(τ_w, I_w, γ, K_{pub}):
Input: τ_w, I_w, γ, K_{pub}, and plaintext kw_s after encryption is $c_{kws}=(kw_s+ K_{priv1}\times r_s) \bmod K_{pub}$
for $1\le i\le \#\mathbf{f}_{kw}$,Compute $Result = ((\tau_w - c_{kws})\times I_w) \bmod K_{pub}$
If $Result$ =0, query keywords $w\in\mathbf{f}$, output $\mathbf{f}_{id}$ ($\mathbf{f}_{id}$ is file identifier collection which contains keyword w);
Else $w\notin\mathbf{f}$, output "can't find the required files".

Decryption(K', c, $\mathbf{f}_{id}$):
Input: ***K'*, c, $\mathbf{f}_{id}$**
for $1\le i\le \#\mathbf{f}_w$, $f_i \leftarrow$DECRYPT_AES $_{K'}(f_i)$)
Output: f

AddToken(Bulidindex, Searchkw, K', f_a) and **Add(γ, c, τ_a, c_f)**(the operation of add file)
Input: ***K'*, f_a,γ, c, τ_a, c_f**
Step 1: $c_{fa} \leftarrow$AES $_{K'}(f_a)$);(f_a is the file which is want to be added)
Step 2: **Bulidindex** for f_a
Step 3: **Searchkw** from f_a whether there is identical keyword in γ
for $1\le j\le \#\mathbf{f}_{kw}$, $Result \leftarrow ((c_{kwj} - c_{ai})\times I_w) \bmod K_{pub}$ (a_i denotes i^{th} element of f_a)
Step 4: if $Result$=0, $\tau_a \leftarrow E_{h(a)}(\mathrm{id}(f_a))$,
p1=(struct node*)malloc(sizeof(struct node)) (assign a new node)
p[#$\mathbf{f}_j$]->next=p1 (#$\mathbf{f}_j$ is the number of **f** that contain the keyword j)
p1->next=NULL
* p1=τ_a
Step 5:else $\tau_a= c_{ai}$,
p1=p2=(struct node*)malloc(sizeof(struct node)) (assign 2 new nodes)
p[#$\mathbf{f}_{kw}$]->next=p1
p1->next=NULL
* p1=τ_a
head->next =p2
p2->next=NULL
* p2= $E_{h(a)}(\mathrm{id}(f_a))$
Output: c_{fa}, **c'** (the new ciphertext collection)

DelToken(n, f_d) and **Del(γ, c, τ_d)** (the operation of delete file)
Input: ***n*, f_d, γ, c, τ_d**
Step 1: $\tau_d= E_{h(d)}(\mathrm{id}(f_d))$ (τ_d is the token of f_d which is want to be deleted)
Step 2: search for all the same values in the ciphertext index γ, and delete them, i.e.
for $1\le j\le \#\mathbf{f}_w$,if $c_{\mathrm{id}(d)}= c_{\mathrm{id}(fi)}$,$\mathbf{p}_{j-1}$->next= $\mathbf{p}_{j+1}$
Output: **c'**, γ'(the ciphertext index which has been updated)

4 Performance evaluation

4.1 Correctness of DSE

Theorem 1: The query operation of DSE can search the correct keywords.
Proof: When query keywords, compute:
$Result = ((\tau_w - c_s)\times I_w) \bmod K_{pub}$
$= (((w+ K_{priv1}\times r_w)- (m_s+ K_{priv1}\times r_s))\times I_w) \bmod K_{pub}$
$=((w- m_s) \times I_w +(r_w - r_s)\times K_{priv1}\times I_w) \bmod K_{pub}$
$=((w- m_s) \times g\times t\times K_{priv2} +(r_w - r_s)\times K_{priv1}\times g\times t\times K_{priv2}) \bmod K_{pub}$

$\because K_{pub} = K_{priv1} \times K_{priv2}$

$\therefore ((r_w - r_s) \times K_{priv1} \times g \times t \times K_{priv2}) \bmod K_{pub} = 0$;

if $w = m_s$, $\because w - m_s = 0, \therefore ((w - m_s) \times g \times t \times K_{priv2}) \bmod K_{pub} = 0$;

else if $w \neq m_s$, $((w - m_s) \times g \times t \times K_{priv2}) \bmod K_{pub} > 0$.

Theorem 2: The delete operation of DSE can search and delete all correct file identifiers.

Proof: File ID is encrypted by CE, there are 2 identical plaintexts pt_1 and pt_2,

$\because H(pt_1) = H(pt_2)$, i.e. $key(pt_1) = key(pt_2)$,

$\therefore E_{H(pt1)}(pt_1) = E_{H(pt2)}(pt_2)$.

$\therefore pt_1 = pt_2 \Rightarrow E_{H(pt1)}(pt_1) = E_{H(pt2)}(pt_2)$

So the delete operation of DSE can search and delete all correct file identifiers.

4.2 Security of DSE

Theorem 3: OHE is security against chosen-ciphertext attacks.

Proof: Build a ciphertext dictionary[13]

$E_k([p_{11},p_{12},\ldots,p_{1n}]) = [1,0, \ldots, 0] = c_1$

$E_k([p_{21},p_{22},\ldots,p_{2n}]) = [0,1, \ldots,0] = c_2$

……

$E_k([p_{m1},p_{m2},\ldots,p_{mn}]) = [0,0, \ldots,1] = c_m$

For any given ciphertext $E_k([x_1,x_2,\ldots,x_n]) = [a_1,\ldots,a_m]$, attacker can compute$[x_1,x_2,\ldots,x_n] = \sum a_i [p_{i1},p_{i2},\ldots,p_{in}]$. The premise of this kind of attack is to use the same key to encrypt data, and it is a deterministic encryption algorithm with additive homomorphism.

But, this kind of attack is is invalid for OHE, because it uses three non-identifiable random encryption parameters r,g,t:

$E_{k,r,g,t}([p_{11},p_{12},\ldots,p_{1n}]) = [1,0, \ldots,0] = c_1$

$E_{k,r,g,t}([p_{21},p_{22},\ldots,p_{2n}]) = [0,1, \ldots,0] = c_2$

……

$E_{k,r,g,t}([p_{m1},p_{m2},\ldots,p_{mn}]) = [0,0, \ldots,1] = c_m$

This means that encrypts any plaintext m several times, and the ciphertexts of m are difference. It also means that the identical ciphertext can correspond to different plaintext m. The probability of decrypting ciphertext c and get the correct plaintext is $1/2^n$ (n donates the binary digits of plaintext space).Then the probability of decrypting all ciphertext $c_i (i=1,\ldots,m)$ and get the correct plaintext is $1/2^{nm}$. This probability is very small, so OHE can against chosen-ciphertext attacks.

Theorem 4: Cloud providers can't know the detailed content which users search.

Proof: The index that users store in the cloud is $\mathbf{C}_{index}$=(OHE(keyword$_i$),CE(file IDs)). When users want to search keywords, they send search token Trapdoor=OHE(query keyword), and cloud providers just need subtract $\mathbf{C}_{index}$ and Trapdoor. Since cloud providers only operate ciphertext, then they can't know the detailed content that return to the users. Only users can decrypt and get the required information buy key.

4.3 Execution efficiency of DSE

The data structure of DSE is single linked list which node is also single linked list, as figure 2. Master_List stores the address of the header node which points to the sub list of the keywords. The number of Master_List nodes is equal to the number of keywords in index. Head node of Sub_Lists is keyword, and node is file identifiers.

Linked list can make full use of computer memory space that achieve flexible memory dynamic management, and it overcomes the shortcomings that the array list must know the size of the data in advance. But the linked list has lost the advantage of random reading array, and has increased the space overhead because of extra pointer domain of the node.

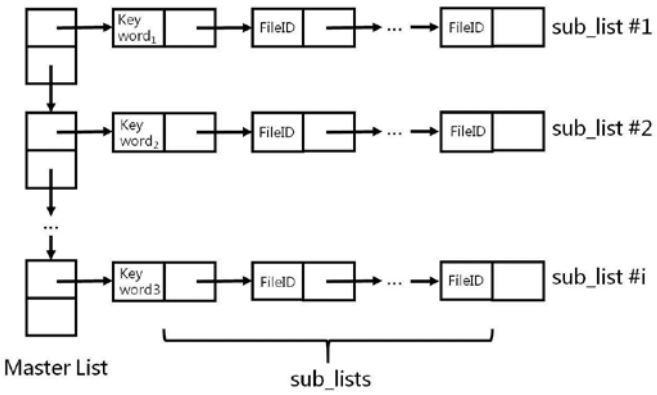

Figure 2. The data structure of DSE.

For the linked list of this paper, the time complexity of add data is O(1), query or access specific number node is O(n),and delete a node is O(n).

Table 4 shows us about comparison of several SE schemes. Need to be explained is that the search time is for each search keyword w, and update time is for each file f. $|\mathbf{f}|$ is its bit length, $\#W$ is the size of the keyword space and m_f is the maximum number of files in which a keyword appears.

Table 4 summarizes the differences between our scheme and others that have appeared in the paper. And we can see that DSE can achieve the dynamic update of the ciphertext index and it is CKA2-security. DSE has made the search time as short as possible, and the index is moderate size.

Table4. Comparison of several SE schemes[7].

Scheme	Dynamic or not	Security	Search time	Index size
Z-IDX[2]	Yes	CKA1	$O(\#\mathbf{f})$	$O(\#\mathbf{f})$
SSE-1[3]	No	CKA1	$O(\#\mathbf{f}_w)$	$O(\sum_w \#\mathbf{f}_w + \#W)$
SSE-2[3]	No	CKA2	$O(\#\mathbf{f})$	$O(\#\mathbf{f} \cdot \#W)$
CK10[4]	No	CKA2	$O(\#\mathbf{f}_w)$	$O(\#W \cdot m_f)$
vLSDHJ10[5]	Yes	CKA2	$O(\log \#W)$	$O(\#W \cdot m_f)$
KPR12[7]	Yes	CKA2	$O(\#\mathbf{f}_w)$	$O(\sum_w \#\mathbf{f}_w + \#W)$
This paper	Yes	CKA2	$O(\#\mathbf{f}_{kw})$	$O(\sum_w \#\mathbf{f}_w + \#W)$

5 Conclusion

In this paper we introduce a dynamic searchable encryption (DSE) algorithm which based on homomorphic encryption and convergent encryption algorithm to achieve environment reliability provision mechanism(ERPM). Firstly, DSE algorithm can fully protect the confidentiality and privacy of the data stored in the cloud and let the cloud hosting data become controllable. Secondly, in such an encrypted cloud storage environment, user can still utilize the functions offered by cloud platform to manage, search and mine the ciphertext at any time. The ciphertext index is security against chosen-ciphertext attacks, and it is CKA2-security. The most important feature is that the ciphertext index can be updated dynamically according to the add and delete operation of plaintext.

Comparing with other related algorithms, DSE algorithm can achieve higher security level (see table 4) and index dynamic update with adding index size and search time. The secure cloud storage system can fully satisfy the user security requirement on stored data and maintain the powerful processing and computing capabilities of clouding computing related to the operations on ciphertext.

Acknowledgments

This work was supported by the Science and technology major projects No. 2011x03002-005-01.

References

1. D.X.Song, D.Wagner, A.Perrig:Practical Techniques or Searches on Encrypted Data,*Proceedings of the IEEE Symposium on Security and Privacy*, 36-49(2000)
2. E.J.Goh:Secure Indexes,*Proceedings of the 2004 Workshop on Information Security Applications*, **7(15)**, 73-86(2004)
3. R.Curtmola, J.Garay, S.Kamara:Searchable symmetric encryption: Improved definitions and efficient constructions,*Proceedings of ACM Conference on CCS*, 79-88(2006)
4. M.Chase, S.Kamara:Structured encryption and controlled disclosure,*Proceedings of ASIACRYPT 2010*, 577-594 (2010)
5. V.P.Lisedonk, S.Sedghi, J.Doumen: Computationally efficient searchable symmetric encryption, *Proceedings of Workshop on SDM*, 87-100(2010)
6. O.Goldreich, R.Ostrovsky:Software protection and simulation on oblivious RAMs,Journal of the ACM, **43(3)**, 431-473(1996)
7. S.Kamara,C.Papamanthou, T.Rodeder:Dynamic Searchable Symmetric Encryption,*Proceedings of the 2012 ACM conference on CCS*, 965-976(2012)
8. R.L.Rivest, L.Adleman, M.L.Dertouzos:On data banks and privacy homomorphisms,In Foundations of Secure Computation, 169-180(1978)
9. M.Y.Li, J.Li, C.Huang:A Credible Cloud Storage Platform based on Homomorphic Encryption, Netinfo Security, **09**, 35-40 (2012)
10. J.R.Douceur, A.Adya, W.J.Bolosky:Reclaiming Space from Duplicate Files in a Serverless Distributed File System,TechReport,Microsoft Research, Number:MSR-TR-2002-30 (July 2002)
11. J.Zobel, A.Moffat:Inverted files for text search engines,ACM Computing Surverys, **38(2)**, 16-31(2006)
12. M.O.Rabin:Probabilistic Algorithm for Testing Primality,Journal of Number Theory, **12**,128-138 (1980)
13. N.Ahituv, Y.Lapid,S.Neumann : Proeessing encrypted data,Communications of the ACM, **30(9)**, 777-780(1987)