ACSIJ Advances in Computer Science: an International Journal, Vol. 5, Issue 1, No.19 , January 2016
ISSN : 2322-5157
www.ACSIJ.org

ACSIJ
WWW.ACSIJ.ORG

Provable Data Possession Scheme based on Homomorphic Hash Function in Cloud Storage

Li Yu[1] , Junyao Ye[1, 2] , Kening Liu[1]

[1] School of Information Engineering, Jingdezhen Ceramic Institute, Jingdezhen 333403,China
lailul@163.com

2 Department of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai 200240, China
sdyejunyao@sjtu.edu.cn

Abstract

Cloud storage can satisfy the demand of accessing data at anytime, anyplace. In cloud storage, only when the users can verify that the cloud storage server possesses the data correctly, users shall feel relax to use cloud storage. Provable data possession(PDP) makes it easy for a third party to verify whether the data is integrity in the cloud storage server. We analyze the existing PDP schemes, find that these schemes have some drawbacks, such as computationally expensive, only performing a limited number provable data possession. This paper proposes a provable data possession scheme based on homomorphic hash function according to the problems exist in the existing algorithms. The advantage of homomorphic hash function is that it provides provable data possession and data integrity protection. The scheme is a good way to ensure the integrity of remote data and reduce redundant storage space and bandwidth consumption on the premise that users do not retrieve data. The main cost of the scheme is in the server side, it is suitable for mobile devices in the cloud storage environments. We prove that the scheme is feasible by analyzing the security and performance of the scheme.

Keywords: *Cloud Storage, Provable Data Possession, Homomorphic Hash Function, Data Possession Checking.*

1. Introduction

Cloud storage[1] can satisfy the demand of accessing data at anytime, anyplace. For the users who need inexpensive storage and unpredictable storage capacity, compared with purchasing the entire storage system, purchasing cloud storage capacity needed will obviously bring more convenience and efficiency. Cloud storage not only saves investment for the users but also saves resources and energy of society. However, there are still many problems to be solved such as security, reliability and service level of cloud storage, so it has not been widely used. When the users stores data in the cloud server, they are most concerned about whether the data is integrity. In cloud storage, only when the users can verify that the cloud storage server possesses the data correctly, they shall feel relax to use cloud storage.

Remote data verification allows the client to test the integrity of outsourcing data on an untrusted server. Proof of retrievability(POR) is the integrity verification algorithm proposed by Juels[2], the key method of the POR is to add some random data blocks to the stored data, its insertion position is determined by a pseudo-random sequence, and uses error-correcting codes. These data blocks and the stored data itself does not have any relationship, which are called sentinels, and these sentinels play in the role of tag. The tag is used to test the integrity of data. Provable data possession(PDP) is proposed by Ateniese[3], which has two notable characteristics, one is supporting public verification, another is using homomorphic signature algorithm[4]. These two characteristics make it easy for a third party to verify whether the data is integrity on the server.

Data possession checking(DPC) is proposed by Da Xiao[5], the basic idea is that the verifier randomly assigns several data blocks and its corresponding key, the server computes the hash value and returns to the verifier, then the verifier compares the hash value is consistent with the check block, thereby the verifier can determine whether the data is correctly held. The literature [6] proposed integrity checking for remote data based on RSA hash function. Let N be moduli of RSA, F be big integer representing file, the verifier saves $k = F \bmod \varphi(N)$. During the challenge, the verifier sends $g \in Z_N$, then the server returns $s = g^F mod\ N$, the verifier checks whether the equation $g^k mod\ N = s$ is satisfied.

Scalable provable data possession(SPDP) is proposed by Ateniese[7], the difference between SPDP algorithm and PDP algorithm is that the SPDP algorithm is supporting dynamic data. SPDP algorithm adopts label organization as the agreed number of tag, then encrypts the generating tag with a symmetric key and stores it in the server or local. Each challenge uses a tag to verify, but the scheme also restrict the number of verification. Dynamic provable data possession(DPDP) is proposed by Erway[8], the DPDP algorithm uses skip list which is like tree structure to generate tag, compared with SPDP algorithm, the DPDP algorithm is supporting dynamic data integrity verification. Chen proposed another algorithm[9] based DPDP algorithm, which uses RS code and Cauchy matrix

to intensify the robust and dynamic updates of the initial algorithm. In addition, there are integrity verification scheme IPDP in the private cloud and integrity verification CPDP[10] in the hybrid cloud.

In summary, the existing schemes have some drawbacks as follows: (1) most schemes based on public key cryptography are computationally expensive, especially when dealing with large volume of data. (2) Can only perform a limited number provable data possession. (3)Some schemes do not apply to the cloud storage service environment.

This paper proposes a provable data possession scheme based on homomorphic hash function according to the problems exist in the above algorithms. This paper uses the homomorphic hash function in literature [11], proposes a data integrity verification scheme based on homomorphic hash function supporting dynamic data and unlimited challenges. The advantage of homomorphic hash function is that it provides provable data possession and data integrity protection. The scheme is a good way to ensure the integrity of remote data and reduce redundant storage space and bandwidth consumption on the premise that users do not retrieve data. The main cost of the scheme is on the server side, it is suitable for mobile devices in the cloud storage environments. We prove the scheme is feasible by analyzing the security and performance of the scheme.

The remainder of this paper is organized as follows. Section 2 discusses theoretical preliminaries for the presentation. Section 3 describes provable data possession scheme based on homomorphic hash function. Section 4 analyzes the security of provable data possession scheme. Section 5 analyzes the performance of provable data possession scheme. We conclude in Section 6.

2. Preliminaries

We now recapitulate some essential concepts from homomorphic hash function.

2.1 Symbol of Homomorphic Hash Function

Firstly we explain some parameters of homomorphic hash function, all the parameters are generated in the setup stage, as is shown in Table 1.

TABLE 1
SYSTEM PARAMETERS AND PROPERTIES

Name	Description	e.g.
λ_p	discrete log security parameter	1024 bit
λ_q	discrete log security parameter	257 bit
p	random prime,$\|p\|=\lambda_p$	
q	random prime,$q\|p-1, \|q\|=\lambda_q$	
β	block size in bits	16 KB
m	number of "sub-blocks" per block	512 bit
	$m=\lceil \beta/(\lambda_q-1) \rceil$	
g	$1\times m$ row vector of order q elts in Z_p	
G	hash parameters, given by (p, q, g)	
n	number of file blocks	
seed	seed of key stream generator	
MAXR	maximum possible output of R(•)	
MINR	minimum possible output of R(•)	

In this paper, we uses the following two cryptographic primitives:

$H_G(\bullet):\{0,1\}^k \times \{0,1\}^\beta \rightarrow \{0,1\}^{\lambda_p}$

$R(\bullet):\{0,1\}^k \times \{0,1\}^l \rightarrow \{0,1\}^l$

Among which, k is the key length, $H_G(\bullet)$ is homomorphic hash function, $R(\bullet)$ is pseudo-random function, is used as pseudo-random generator.

2.2 Homomorphic Hash Algorithm

In algebra, homomorphism is the constant mapping between two algebraic structures, such groups, rings, fields or vector space. That is to say there exist mapping $\Phi: X \rightarrow Y$, satisfying $\Phi(x+y)=\Phi(x)\times\Phi(y)$, + is the operator of set X, and × is the operator of the set Y.

Generally, public key cryptography algorithm has the characteristic of homomorphism, for example, RSA algorithm has homomorphism for multiplication. Let k be public key, n be moduli. The encryption algorithm is $E_k(\cdot)$, for message x and y, we have the following:

$$E_k(x\times y)=(x\times y)^k \, mod\, n=(x^k mod\, n)\times(y^k mod\, n)=E_k(x)\times E_k(y) \quad (1)$$

If some algorithms satisfy the homomorphism for addition, then we have the equation:

$$E_k(x+y)=E_k(x)+E_k(y) \quad (2)$$

Some algorithms satisfy the homomorphism for multiplication, such as RSA. Some algorithms satisfy the homomorphism for addition, such as Paillier. If some algorithms satisfy homomorphism for addition and multiplication, then they are called full homomorphism algorithms. There is no genuine full homomorphic encryption algorithms available at present[12]. Homomorphic hash function means that the hash function has the characteristic of homomorphism. The homomorphic hash function used in this paper is based on literature[11].

Files F represented by $m\times n$ matrices, all the elements in the matrices belong to Z_q, because of $m=\lceil \beta/(\lambda_q-1) \rceil$, so every element less than $2^{\lambda_q}-1$, therefor less than q.

$$F=(b_1,b_2,\cdots,b_n)=\begin{pmatrix} b_{1,1} & \cdots & b_{1,n} \\ \vdots & \ddots & \vdots \\ b_{m,1} & \cdots & b_{m,n} \end{pmatrix} \quad (3)$$

We add two blocks by adding their corresponding column-vectors. That is, to combine the i^{th} and j^{th} blocks of the file, we simply compute:

$$b_i + b_j = (b_{1,i} + b_{1,j}, \cdots, b_{m,i} + b_{m,j}) mod\ q \quad (4)$$

For file F, the computation of hash value is as following, firstly computes the hash value of each data block:

$$H_G(b_j) = \prod_{t=1}^{m} {g_t}^{b_{t,j}}\ mod\ p \quad (5)$$

The hash value of file F is $1 \times n$ row vector, every element in the row vector is the hash value of each block of the file:

$$H_G(F) = (H_G(b_1), H_G(b_2), \cdots, H_G(b_n)) \quad (6)$$

From the calculation process of each block, we can obtain the homomorphism of hash function:

$$\begin{aligned} H_G(b_i + b_j) &= \prod_{t=1}^{m} {g_t}^{b_{t,i}+b_{t,j}}\ mod\ p \\ &= \prod_{t=1}^{m} {g_t}^{b_{t,i}}\ {g_t}^{b_{t,j}}\ mod\ p \\ &= \prod_{t=1}^{m} {g_t}^{b_{t,i}}\ mod\ p \times \prod_{t=1}^{m} {g_t}^{b_{t,j}}\ mod\ p \\ &= H_G(b_i) \times H_G(b_j) \end{aligned} \quad (7)$$

Among which, each block is β bit, hash length is λ_p bit, if we choose β = 16 KB, $\lambda_p = 1024$ bit, then after the hash value of the file is calculated, the expansion of the file is $\frac{\lambda_p}{\beta} = \frac{1024}{16 \times 1024 \times 8} \approx 0.0078$.

2.3 Per-Publisher Homomorphic Hashing

The per-publisher hashing scheme is an optimization of the global hashing scheme just described. In the per-publisher hashing scheme, a given publisher picks group parameters G so that a logarithmic relation among the generators g isknown. The publisher picks q and p as above, but generates g by picking a random $g \in Z_p$ of order q, generating a random vector r whose elements are in Z_q and then computing $\boldsymbol{g} = g^r$.

Given the parameters g and r, the publisher can compute file hashes with many fewer modular exponentiations:

$$H_G(F) = g^{rF} \quad (8)$$

The publisher computes the product rF first, and then performs only one modular exponentiation per file block to obtain the full file hash. The hasher must be careful to never reveal g and r; doing so allows an adversary to compute arbitrary collisions for H_G.

3. Provable Data Possession based on Homomorphic Hash Function

3.1 Scheme Description

The purpose of provable data possession is to allow the user to verify whether the untrusted storage server holds data correctly. Generally, there are two parties: client and storage server. The scheme of provable data possession base on homomorphic hash function is composed of five phases: (1)Setup; (2)TagBlock; (3)Challenge; (4) ProofGen; (5) ProofVerify.

Firstly, we need to divide the file F into n blocks. In the following phases such as TagBlock phase and ProofVerify phase, all the calculations are based on the file blocks.

1. Setup Phase:

In the *Setup* phase, the input value is $(\lambda_p, \lambda_q, m, s)$, the output value is $G = (p, q, g)$. G is hash parameters, used in homomorphic hash function to produce hash value. *Setup* phase is described as Fig 1.

Setup$(\lambda_p, \lambda_q, m, s) \rightarrow G = (p, q, g)$
Seed PRNG R with s.
do
  q ← qGen(λ_q)
  p ← pGen(q, λ_p)
while p=0 **done**
for(i=1 **to** m) **do**
  do
    x ← R(p − 1) + 1
    $g_i \leftarrow x^{(p-1)/q}\ (mod\ p)$
  while $g_i = 1$ **done**
done
return (p,q,g)

qGen(q, λ_q)
do
  q ← R(2^{λ_q})
while q is not prime **done**
return q

pGen(q, λ_p)
for(i=1 to $4\lambda_p$) **do**
  X ← R(2^{λ_p})
  c ← X(mod 2q)
  p ← X − c + 1
  if p is prime **then return** p
done
return 0

Fig. 1Setup phase

2. TagBlock Phase:

In the TagBlock phase, the client uses pseudo-random generator to generate a series of pseudo-random numbers, then multiply each block of the file F with the corresponding pseudo-random number, and obtain the tag t_i of each block b_i. The client sends the b_i, tag t_i ,p, q to the server, the client saves the hash parameters G and the seed of pseudo-random generator. The detail is described in Fig 2.

TagBlock(G,F)
G=(p,q,g),F=$(b_1, b_2, \cdots, b_n)$
for(j=1 to n) **do**

$$x_j = H_G(b_j) = \prod_{t=1}^{m} g_t{}^{b_{t,j}} \ mod\ p$$

$$r_j = R(seed)$$

done
then Tag=$[x_1 \cdot r_1, x_2 \cdot r_2, \cdots, x_n \cdot r_n]$
return Tag=$[t_1, t_2, \cdots, t_n]$,$t_j = x_j \cdot r_j$
the client sends F,Tag,p,q to the server
saves G and seed

Fig. 2 TagBlock phase

3. Challenge Phase:

In the Challenge phase, the client uses pseudo-random generator to generate k challenge blocks to the server. The detail is described in Fig 3.

Challenge()
the client select k blocks to challenge randomly:
for (j=1 **to** k) **do**

$$r_j' = \lceil rand(time())/(MAXR - MINR) \times n \rceil$$

done
then the client sends<$r_1', \cdots, r_k', k$>to the server

Fig. 3 Challenge phase

4. ProofGen Phase:

In the ProofGen phase, the server calculates b_c and t_c using each block and its corresponding tag, then returns the b_c and t_c to the client. The detail is described in Fig 4.

ProofGen($r_1', \cdots, r_k', k$)

$$b_c = \left(\sum_{i=r_1'}^{r_k'} b_i \right) mod\ q$$

$$t_c = \left(\prod_{i=r_1'}^{r_k'} t_i \right) mod\ p$$

return(b_c, t_c)

Fig. 4 ProofGen phase

5. ProofVerify Phase:

In the ProofVerify phase, the client uses seed to reproduce the corresponding pseudo-random numbers, then verify whether the t_c is exactly the t_c that the client specified. Also verify that the t_c is corresponding to the b_c. The detail is described in Fig 5.

ProofVerify(b_c, t_c)
The client verifies b_c, recalls R(seed) to produce

$$(r_1, r_2, \cdots, r_n)$$

verify:

$$t_c \stackrel{?}{=} H_G(b_c \times r_{r_1'} \times \cdots \times r_{r_k'})$$

If the equation holds, it indicates that the file is intact.

Fig. 5 ProofVerify phase

There are two approaches for provable data possession, one is verified by data owner, and another is verified by a trusted third party. To delegate the work of provable data possession to a trusted third party has the following advantage, when a dispute is emerging, such as the service provider believes that it stores the data, but the data may be placed on secondary storage or offline storage. But the user demands that the server should provide online access, claiming that the performance does not meet the requirements. So it can be arbitrated by a trust third party.

When we use a third party to audit, we should provide privacy protection technology, that is to say, don't disclose the data to a third party. We can use the following privacy protection approaches :(1) First encrypt data and then calculate the relevant verification information, we use the encrypted data during verification, so won't disclose data; (2) Because we use sampling, the response of sampling is not continuous data, not returning the original data, but returning the verification information of original data. (3) Using a general method of privacy protection, add some random data in the data, this method will add extra cost. We will research on the privacy protection technology when a third party audit in the future.

3.2 Supporting Dynamic Data

Supporting dynamic data mainly consists of two operations: insert and delete.

1. Insert Data Block

Assume that insert data block b_s. The client sends to server for insert request, then the client receives from server: (F, Tag). The client will calculate the tag of the insert block b_s, and the tags of the s-th block after. Then send the updated F′ and Tag′ to the server. Then issue immediately the verification of the data block in order to ensure that the data uploaded is correct. The detail process is described in Fig 6.

Insert(b_s)
for(j=1 **to** n+1) **do**
$r_j = R(seed)$
if(j≥ s)
$x_j = H_G(b_j) = \prod_{t=1}^{m} g_t{}^{b_{t,j}} \ mod\ p$
$t_j = x_j \times r_j$
done
return $Tag' = [t_1, t_2, \cdots, t_{n+1}]$
client sends to server:F',Tag'

Fig. 6 Insert data block

2. Delete Data Block

Assume that delete data block b_k. The client sends to server for delete request, then the client receives from server: (F, Tag). The client will calculate the tags of the k-th block after. Then send the updated F' and Tag' to the server. Then issue immediately the verification of the data block in order to ensure that the data uploaded is correct. The detail process is described in Fig 7.

Delete(b_k)
for(j=1 **to** n-1) **do**
$r_j = R(seed)$
if(j≥ k)
$x_j = H_G(b_j) = \prod_{t=1}^{m} g_t{}^{b_{t,j}} \ mod\ p$
$t_j = x_j \times r_j$
done
return $Tag' = [t_1, t_2, \cdots, t_{n+1}]$
client sends to server:F',Tag'

Fig. 7 Delete data block

4. Security Analysis

4.1 Security of Homomorphic Hash Function

The homomorphic hash function used in this paper is based on discrete logarithm assumption. We analyze whether a probabilistic polynomial time(PPT) adversary can find a pair collision in probabilistic polynomial time. We use the method in literature [13] to define homomorphic hash function. A hash function family is defined by PPT algorithm $Q = (Hgen, H)$. Hgen represents a hash generator, input security parameters $(\lambda_p, \lambda_q, m)$, output a member G of hash function family. For hash function H_G, input the data of length $m\lambda_q$, output the hash value of length λ_p. $\mathcal{A}$ is a PPT adversary trying to a pair collision of the given hash function family.

Definition 1 For any hash function family Q, any PPT adversary $\mathcal{A}$, security parameters $\lambda = (\lambda_p, \lambda_q, m)$, and $\lambda_q < \lambda_p$, $m \le \text{poly}(\lambda_p)$, s.t. $\text{Adv}_{Q,\lambda} = \Pr[G \leftarrow Hgen(\lambda); (x_1, x_2) \leftarrow \mathcal{A}(G): H_G(x_1) = H_G(x_2) \wedge x_1 \neq x_2]$

If PPT adversary $\mathcal{A}$'s time complexity is $\tau(\lambda)$, $\text{Adv}_{Q,\lambda}(\mathcal{A}) < \varepsilon(\lambda)$, $\varepsilon(\lambda)$ is a neglect function, $\tau(\lambda)$ is the polynomial of λ, then Q is a secure hash function.

The homomorphic hash function H_G in this paper is satisfying definition 1. If to construct discrete logarithm problem in finite field through parameters (λ_p, λ_q) is hard, then H_G is a collision-resist hash function. The detail provable process refer to literature [13].

4.2 Security of Pseudo-random Generator

In our scheme, the data blocks and their corresponding tags, p, q are saved in the server. The seed for generating pseudo-random numbers, p, q, g are saved in the client. The tags are verifiable, the construction of tags is using homomorphic hash algorithm and a series of pseudo-random numbers.

In the Challenge phase, the challenger generates k challenge blocks randomly, then send the k blocks to adversary $\mathcal{A}$, $\mathcal{A}$ generates integrity verification P, if P passes the validation, then $\mathcal{A}$ performs a successful deception. If $\mathcal{A}$ deletes the challenge blocks, then sends arbitrary data blocks and its corresponding tags to challenger. At this point, though the return value b_c can be verified is correct corresponding to t_c, $\mathcal{A}$ doesn't know the random numbers r_i used in constructing tags, so the challenger hash the data blocks he has received, using the same seed to generating random numbers, then computes the tags, compared with the tags $\mathcal{A}$ has returned, then you can verify whether the data blocks and tags $\mathcal{A}$ has returned are designated by challenger.

5. Performance Analysis

First, we analyze the performance cost of each phase in provable data possession. We convert all the exponentiations into multiplications. We denote the multiplication cost in Z_p^* as MultCost(p). For calculatingy^x, we need $1.5|x|$ times multiplications using Iterative Square method. First calculate ${y_i}^{2^z}$, build a list for ${y_i}^{2^z} (1 \le z \le \lambda_p)$, need $|x|$times multiplications, then looking for a list needs $|x|/2$ multiplications. During the process of calculating hash value, we all need to look for a list ${y_i}^{2^z}$. So the list is built in the Setup phase. To simplify the performance analysis, we will ignore the computation cost during the following analysis.

In the Setup phase, generating the key G of homomorphic hash function, relating to the random number generation and modulus exponentiation, for

parameters p and q, mainly uses a random number generator and a simple prime testing, for parameter g, needs $m(p-1)/2q$ mod p multiplications, its cost is $m(\lambda_p - 1)\text{MultCost}(p)/2\lambda_q$. However, these parameters are only generated once. For any approach of provable data possession, these parameters are indispensable and their cost is almost the same. In the TagBlock phase, the size of the data block β is 16KB, the output of homomorphic hash function is 1024 bit, so the hash function reduces the storage space of file to its original $\frac{\lambda_p}{\beta} = \frac{1024}{16\times1024\times8} = \frac{1}{128}$. This method of tag organization is very helpful in reducing storage redundancy. We need to compute the hash value of each data block, relating to $nm|p|/2$ mod p multiplications, its cost is $nm\lambda_p MultCost(p)/2$. In the Challenge phase, has the cost of generating two random numbers. In the ProofGen phase, has k times mod q additions, also has k times mod p multiplications, here the cost of multiplication is large, is $c\text{MultCost}(p)/2$. In the ProofVerify phase, has one time homomorphic hash calculation, relating to m times mod p multiplications, its cost is $m\lambda_p MultCost(p)/2$.

In practical use of cloud storage service, performance is always limited by the network bandwidth. Modular multiplication algorithm can be optimized, the optimization method refer to literature [11]. Optimized performance can improve more than 4 times. Zhao et al. [14] proposed the use of the graphics processing unit[15,16] to accelerate the performance of homomorphic hash function. We can also use this method to improve the performance in our scheme.

6. Conclusions

This paper proposes a provable data possession scheme based on homomorphic hash function according to the problems exist in the above algorithms. This method allows users to verify data integrity on the server for unlimited number of times.It also provides provable data possession on the server and data integrity protection. Users only need to save key G, transmission information is little during the verification process, and the verification of provable data possession is just one time homomorphic hash calculation. Through security analysis and performance analysis shows that the method is feasible. The scheme can achieve data recovery. We can use error-correcting code or erasure code to encode data before calculating hash value.

Acknowledgments

We are grateful to the anonymous referees for their invaluable suggestions. This work is supported by the National Natural Science Foundation of China (Nos. 61472472). This work is also supported by JiangXi Education Department (Nos. GJJ14650 and GJJ14642).

References

[1] Feng Dengguo, Zhang Min, Zhang Yan, et al. Study on cloud computing security. Journal of Software, 2011, 22(1): 71-83

[2] Juels A，Kaliski B S，Por S. Proof of rerievability for large files. Proc of the 14th ACM Conf on Computer and Communications Security. New York: ACM， 2007: 584-597.

[3] Ateniese G，Burns R，Curtmola R， et al． Provable data possession at untrusted stores. Proc of the 14th ACM Conf on Computer and Communications Security. New York: ACM， 2007: 598-609.

[4] Johnson R, Molnar D, Song D, et al. Homomorphic signature schemes. Proc of CT-RSA． New York: Springer, 2002:244-262.

[5] Da Xiao，Jiwu Shu，Kang Chen,et al． A practical data possession checking scheme for networked archival storage. Journal of Computer Research and Development, 2009, 46(10):1660-1668.

[6] Deswarte Y, Quisquater J J, and Saidane A. Remote integrity checking. Proc of IICIS'03, Switzerland, Nov.13-14, 2003: 1-11.

[7] Ateniese G, Dipr, Mangini L V, et al. Scalable and efficient provable data possession. Proc of the 4th International Confon Security and Privacy in Communication Netowrks (SecureComm 2008). New York: ACM, 2008:1-10.

[8] Erway C，Kupcu A，Papamanthou C，et al. Dynamic provable data possession. Proc of the 16th ACM Conf on Computer and Communications Security (CCS 2009). New York: ACM,2009: 213-222.

[9] Chen B， Curtmola R. Robust dynamic provable data possession. Distributed Computing Systems Workshops (ICDCSW), 32nd International Conference on． Macau: IEEE, 2012: 515-525.

[10] Zhu Y, Wang H, Hu Z, et al. Cooperative provable data possession. Beijing: Peking University and Arizona University，2010.

[11] Krohn M, Freedman M J， Mazieres D. On-the-fly verification of rateless erasure codes for efficient content distribution. Proc of IEEE Symposium on Security and Privacy． Lee Badger: IEEE, 2004: 226-240.

[12] Bruce Schneier. Schneier on Security. http://www.schneier.com/blog/archives/2009/07/homomorphic_enc.html.

[13] Bellare M, Goldreich O, and Goldwasser S. Incremental cryptography: the case of hashing and signing. Advances in Cryptology-CRYPTO'94, Santa Barbara, CA, Aug. 1994: 216-233.

[14] Zhao Kaiyong, Chu Xiaowen, Wang Mea. Speeding up homomorpic Hashing using GPUs. The 2009 (44th) IEEE Conference on Communication (ICC 2009), Dresden, Germany, June 14-18, 2009: 1-5.

[15]Bowers K D, Juels A, and Oprea A. HAIL: a high-vailability and integrity layer for cloud storage. Proceedings of ACMCCS'09, Chicago, Illinois, USA, Nov. 9-13, 2009: 187-198.

[16] Shacham H and Waters B. Compact proofs of retrievability. Proceedings of ASIACRYPT '08, Melbourne, Australia, Dec.7-11, 2008: 90-107.

Li Yu received her M.S. degree from JXAU University, Nanchang, China, in 2004. His research interests include Cryptography, Information Security, Space Information Networks and Internet of Things.

Junyao Ye is a Ph.D. student in Department of Computer Science and Engineering, Shanghai JiaoTong University, China. His research interests include information security and code-based cryptography.

Kening Liu received his M.S. degree from Minzu University of China, Beijing, China, in 2001. His research interests include subliminal channel, LFSR, code-based systems and other new cryptographic technology.