EFFECTIVE CLOUD DATA INTEGRITY CHECKING USING SBA ALGORITHM IN CLOUD AUDITING

Jijo T P[1] | R. Sujitha[2]
[1](UG Student, Christ the King Engineering College, jijotp25@gmail.com)
[2](Assistant Professor, Christ the King Engineering College, srisuji14@gmail.com)

Abstract— *To provide as an important application in cloud computing, cloud storage offers user scalable, flexible and high quality data storage and computation services. A growing number of data owners choose to outsource data files to the cloud. Because cloud storage servers are not fully trustworthy, data owners need dependable means to check the possession for their files outsourced to remote cloud servers. To address this crucial problem, some remote data possession checking (RDPC) protocols have been presented. But many existing schemes have vulnerabilities in efficiency or data dynamics. In this paper, we provide a new efficient RDPC protocol based on homomorphic hash function. The new scheme is provably secure against forgery attack, replace attack and replay attack based on a typical security model. To support data dynamics, an operation record table (ORT) is introduced to track operations on file blocks. We further give a new optimized implementation for the ORT which makes the cost of accessing ORT nearly constant. Moreover, we make the comprehensive performance analysis which shows that our scheme has advantages in computation and communication costs*

Keywords— Cloud Storage; Data Possession Checking; Homomorphic Hash Function; Dynamic Operations

1. INTRODUCTION

cloud computing emerges as a novel computing paradigm subsequent to grid computing. By managing a great number of distributed computing resources in Internet, it possesses huge virtualized computing ability and storage space. Thus, cloud computing is widely accepted and used in many real applications. As an important service for cloud computing, cloud service provider supplies reliable, scalable, and low-cost outsourced storage service to the users. It provides the users with a more flexible way called pay-as-you-go model to get computation and storage resources on-demand. Under this model, the users can rent necessary IT infrastructures

Remote data possession checking (RDPC) is an effective technique to ensure the integrity for data files stored on CSS. RDPC supplies a method for data owner to efficiently verify whether cloud service provider faithfully stores the origina l files without retrieving it. Cloud service provider tries to provide a promising service for data storage, which saves the users costs of investment and resource. Nonetheless, cloud storage also brings various security issues for the outsourced data. Although some security problems have been solved In RDPC, the data owner is able to challenge the CSS on the integrity for the target file. the important challenges of data tampering and data lost are still existing in cloud storage. On the one hand, the accident disk error or hardware failure of the cloud storage server (CSS) may cause the unexpected corruption of outsourced files The CSS can generate proofs to prove that it keeps the complete and uncorrupted data. The fundamental requirement is that the data owner can perform the verification of file integrity without accessing the complete original file. Moreover, the protocol must resist the malicious server which attempts to verify the data integrity without accessing the complete and uncorrupted data. Another desired requirement is that dynamic data operations should be supported by the protocol. In general, the data owner may append, insert, delete or modify the file blocks as needed. Besides, the computing complexity and communication overhead of the protocol should be taken into account for real applications

2. RELATED WORKS

2.1 RDPC PROTOCOL

The first RDPC was proposed by Descartes et all based on RSA hash function. The drawback of this scheme is that it needs to access the entire file blocks for each challenge. In 2007, the provable data possession (PDP) model was presented by Attendees et al., which used the probabilistic proof technique for remote data integrity checking without accessing the whole file. In addition, they supplied two concrete schemes (S-PDP, E-PDP) based on RSA. Although these two protocols To overcome this shortcoming, in 2008, they presented a dynamic PDP scheme by using symmetric encryptio n. Nonetheless, this scheme still did not support block insert operation. At the same time, lots of research works devoted to construct fully dynamic PDP protocols. For instance, Saeb et al.provided a RDPC protocol for critical informatio n infrastructures based on the problem to factor large integers, which is easily adapted to support data dynamics. Elway et al.first presented a fully dynamic PDP scheme (DPDP) by using authenticated skip list, which allowed data owner to append, delete, insert and update file blocks at anytime. Wang et al. used Merle hash tree (MHT) to propose another dynamic method for remote data checking, in which each block was hashed to be a leaf node of MHT. By sorting all leaf nodes from left to right, the MHT

implicitly identified the block position which is essential for dynamic operations. However, using MHT caused heavy computation cost. In 2013, Yang and Jia presented an efficient scheme, in which an index table was utilized to support dynamic operations. By the index table, the data owner recorded the logical location and version number for each block for the outsourced file. However, to delete or insert one data block, the verifier had to find the position of the block and shift the remaining entries to insert or delete a row in the index table, which still incurred high computation cost. In, Chen et al. provided a dynamic RDPC scheme by using homomorphic hash function defined in . Unfortunately, their scheme was proved insecure by Yu et al. . To overcome the drawback, Yu et al. presented a new RDPC protocol based on RDPC scheme in and proved the security. They also used MHT to achieve data dynamic operations, which caused the same shortcoming of inefficient as in

2.2 MOTIVATION AND CONTRIBUTION

It is essential for data owners to verify the integrity for the data stored on CSS before using it. For example, a big international trading company stores all the imports and exports record files on CSS. According to these files, the company can get the key information such as the logistics quantity, the trade volume etc. If any record file is discarded or tampered, the company will suffer from a big loss which may cause bad influence on its business and development. To avoid this kind of circumstances, it is mandatory to check the integrity for outsourced data files. Furthermore, since these files may refer to business secret, any information exposure is unacceptable. If the company competitor can execute the file integrity checking, by frequently checking the files they may obtain some useful information such as when the file changes, the growth rate of the file etc., by which they can guess the development of the company. Thus, to avoid this situation, we consider the private verification type in our scheme, that is, the data owner is the unique verifier. In fact, the current research direction of RDPC focuses on the public verification, in which anyone can perform the task of file integrity checking with the system public key. Although RDPC with public verification seems better than that with private verification, but it is unsuitable to the scenario mentioned above blocks is equal to the product for two hash values of the corresponding blocks. We introduce a linear table called ORT to record data operations for supporting data dynamics such as block modification, block insertion and block deletion. To improve the efficiency for accessing ORT, we make use of doubly linked list and array to present an optimized implementation of ORT which reduces the cost to nearly constant level. We prove the presented scheme is secure against forgery attack, replay attack and replace attack based on a typical security model. At last we implement our scheme and make thorough comparison with previous schemes. Experiment results show that the new scheme has better performance and is practical for real applications

2.3 HOMOMORPHIC HASH FUNCTION

Our scheme adopts the homomorphic hash function defined in as the basis, which is described as following: homomorphic hash function defined in as the basis, which is described as following:

First, the algorithm H KeyGen(p,q ,m,s)->K is utilized to obtain the homomorphic key. It takes four security parameters as inputs, in which p and q are two discrete log security parameters, m is the sector count of the message ands is a random seed. It outputs the homomorphic key K(p,q,g) , where p and q are two random big primes of p and q. The detailed process of this algorithm is shown in Fig, in which the function f (x) is the pseudo-random number generator with seed s and outputs the next number in its pseudo- random sequence, scaled to the range (0…x-1).The computation of the hash value X of the message

Equation should be represented by following

$$H(s) + H(v) = \mathrm{mod}(p,q) + \mu p + \beta q + ¥x$$

Function HKeyGen($\lambda_p, \lambda_q, m, s$)
do
$q \leftarrow qGen(\lambda_q)$
$p \leftarrow pGen(q, \lambda_p)$
while $p = 0$ done
for $i = 1$ to m do
do $x \leftarrow f(p-1)+1$
$g_i \leftarrow x^{(p-1)/q} (\mathrm{mod}\ p)$
while $g_i = 1$ done
done
return $(p, q, \vec{g})$

Function $qGen(\lambda_q)$
do
$q \leftarrow f(2^{\lambda_q})$
while q is not prime done
return q

Function $pGen(q, \lambda_p)$
for $i = 1$ to $4\lambda_p$ do
$X \leftarrow f(2^{\lambda_p})$
$c \leftarrow X (\mathrm{mod}\ 2q)$
$p \leftarrow X - c + 1$ //note: $p \equiv 1 (\mathrm{mod}\ 2q)$
if p is prime then return p done
return 0

2.4OPERATION TABLE

operation record table ,it support data dynamics with help of ORT table ,data owner can check the data changes in cloud storage and it inform to the data owner a simple flexible data structure named operation record table (ORT). The table is reserved on the data owner side and used to record all the dynamic behaviors on file blocks. ORT has a simple structure with only three columns, that is Block Position(BP) , Block Index(BI) and Block Version (BV) . The BP represents the physical index for the current block in the file, normally its value is incremented by 1.The BI represents the logical index for the current block, which is not necessary equal to BP but relevant with the time when the block appears in the file. The BV indicates the current version for the block. If the data file is initially created, the BV values for all blocks are 1. When one concrete block is updated, its BV value is incremented by 1. It is noted that using the ORT table will increase the storage overhead of the data owner by O(n), where n is the count of blocks. However, this extra storage cost is very little. For example, a 1GB- filewith 16KB block size only needs 512KB space to store an Arterialized by linked list (< 0.05% of the file size).

3. OUTLINE OF OUR RDPC PROTOCOL

In this paper, we investigate the cloud storage system including two participants: CSS and data owner. The CSS has powerful storage ability and computation resources, it accepts the data owner's requests to store the outsourced data files and supplies access service. The data owner enjoys CSS's service and puts large amount of files to CSS without backup copies in local. As the CSS is not assumed to be trustable and occasionally misbehave, for example, modifying or deleting partial data files, the data owner can check the integrity for the outsourced data efficiently.

A RDPC scheme includes the following seven algorithms: TagGen(K, sk, F)->T . This algorithm is executed by the data owner to produce tags of the file. It inputs the homomorphic key K , private key sk and file F , and outputs the tag set T which is a sequential collection for tag of each block.

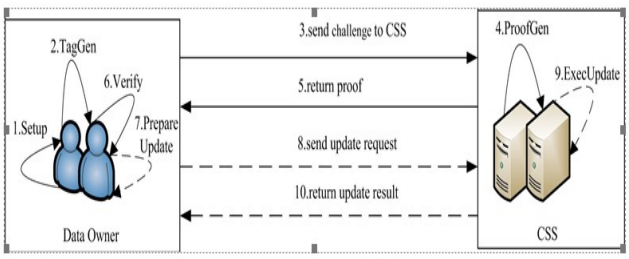
Challenge(c) ->chal . The data owner executes the algorithm to generate the challenge information. It takes the challenged blocks count c as input and outputs the challenge chal . ProofGen(F,T,chal) ->P . The CSS executes this algorithm to generate the integrity proof P . It inputs the file F , tag set T and the challenge chal and outputs the proof P .

Verify(K, sk,chal, P)->{1,0} . The data owner executes the algorithm to check the integrity of the file using the proof P returned from CSS. It takes homomorphism key K , private key sk , challenge chal and proof P as inputs, and outputs 1if P is correct, otherwise it outputs 0

.Prepare Update(Fi ,UT)->URI . The data owner runs this algorithm to prepare dynamic data operations on data blocks. It takes new file block fi the block position i and the update type UT as inputs, and outputs the update request information URI . The parameter UT has three optional elements: insert, modify and delete.

ExecUpdate(URI)->{Success, Fail} . The CSS runs this algorithm to execute the update operation. It inputs URI and outputs execution result. If the update operation is finished successfully, it returns Success , otherwise returns Fail

The complete work procedure of our RDPC protocol is illustrated in Fig., in which solid lines and dash lines represent the processes of data integrity checking and data dynamic operations respectively.



4. SECURITY REQUIREMENT

The CSS is not fully trusted since it might take malicious behaviours on outsourced data and hide data corruption occurrences from data owner so as to keep good reputation. According to , the dishonest CSS may launch three types of attacks on RDPC, namely forge attack, replay attack and replace attack. Forge attack: the CSS forges a valid tag for the challenged block to cheat the data owner. Replay attack: the CSS chooses a valid proof for possession from previous proofs or other informatio n, without accessing the actual challenged block and tag.

Replace attack: the CSS utilizes the other valid pair for block and tag as the proof of the challenged one, which may has been tampered or discarded. A secure RDPC protocol should be able to resist all the attacks above, which guarantees that anyone who can construct valid proof passing the verification should actually possess the entire file. we use a data possession checking game to captures the data possession property which covers all the three attacks. The game which involves challenger served as data owner and an adversary served as untrusted CSS is shown as follows: Setup. executes KeyGen algorithm to get the homomorphic key K and private key sk . Both of them are kept secretly by Query. can make two types of queries with Tag query. adaptively chooses amount of data blocks and sends them to for querying the tags. executes theTagGen algorithm to obtain a valid tag of each block and returns all the tags to Proof verificatio n query. generates data possession proofs for the blocks whose tags have been queried and submits the proofs to executes the Verify algorithm to check the validation for the proofs and returns the results to These queries can be repeated polynomial times. Challenge. submits challenge chal to and requires to reply data possession proof P of the challenged blocks. Forge. computes a proof P and returns it to wins the game if P is a correct proof. Definition 1. A RDPC scheme is secure if any probabilistic polynomial-time (PPT) adversary can win the data possession game on a set of blocks with non-negligible advantage, there exists a knowledge extractor which can extract the challenged blocks with non-negligible probability. is comparable to that of verifying one signature and is increased only gradually when the batch size n is increased.

5. DYNAMIC RDPC SCHEME

In real applications, outsourced data file is dynamic. This inspires us to construct dynamic RDPC schemes with various data block operations. In this paper, we use ORT table as the auxiliary tool to support data dynamic operations. The similar idea has been adopted by To upgrade our static scheme described in the sub-section III.A to be a fully dynamic scheme, we have to do another two works. Firstly, we need to make slight modification on the algorithm TagGen of the static scheme.

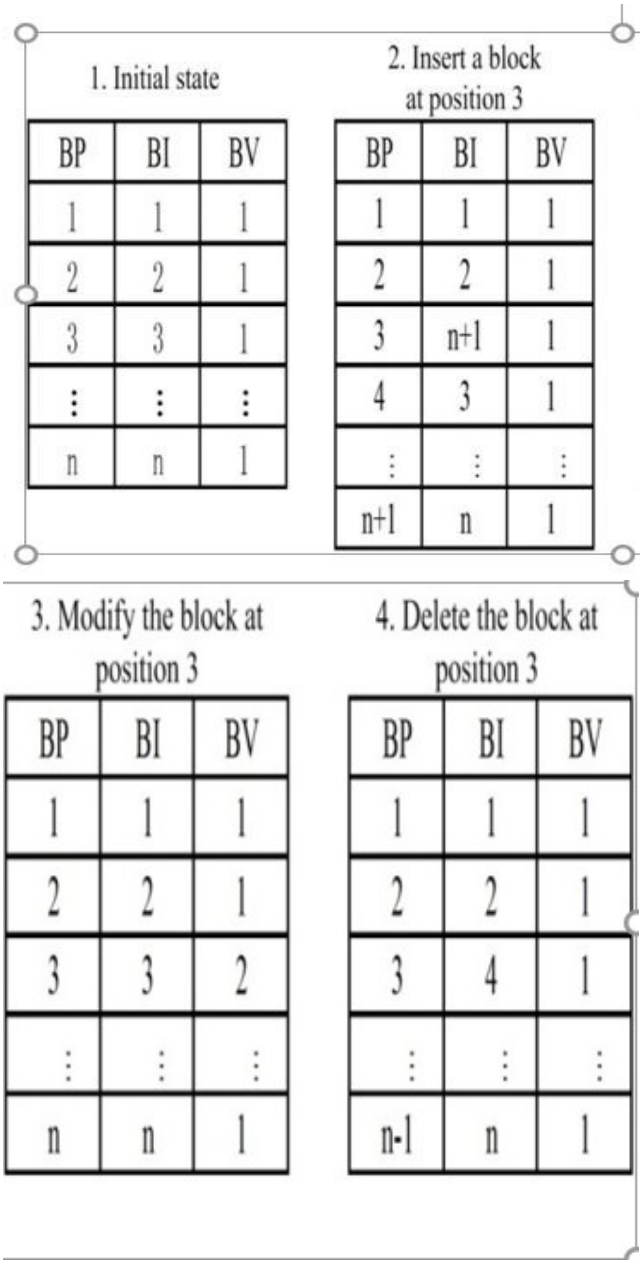
Secondly, we should implement the algorithms ExecUpdate and PrepareUpdate , which are the core functions for data dynamics. Then, we will present these two works respectively. Modificatio n on TagGen . In the new TagGen algorithm, besides doing all the works in static scheme, the data owner has to initialize the ORT table and store the information for all the blocks to the ORT. As shown in Fig, in initialization phase all the BP values are the real indexes of blocks with ascending order,

the BI values are the same as BP , and the values of BV are initialized to 1. Simultaneously, to enhance the security of the dynamic RDPC protocol, i is updated to where BIi and BVi denote the BI value and the BV value of the block Fi .

PrepareUpdate(Fi ',i,UT)->URI . This algorithm is responsible for preparing the pre-works for dynamic operations. The data owner is able to execute three types of operations on his outsourced file, namely 'insert', 'delete' and 'modify'. The pre-works for the three dynamic operations will be finished by the same algorithm PrepareUpdate with different parameters. The three different situations are described as follows. PrepareUpdate(Fi ',i,insert)->URI . For the 'insert' operation, the value of the parameter UT is set to be 'insert'. The parameter Fi ' denotes the new block to be inserted, and i is the position where the new block will be inserted. The data owner first inserts a new row after the (i 1)-th row in ORT, and sets the values of the new entry to (i, Max{BI}1,1) , where Max{BI} means the max value of all BI in ORT. At the same time, all the entries in the original ORT from the i – th to the end move backward in order, with the all BP value The Fi ' denotes the new version for data block to be updated, i is the position of the block. The data owner replaces the version number VI by BVi -> in the i -th row of ORT and calls the algorithm TagGen to re- calculate the tag Ti ' for the new block Fi ' . Then the data owner submits the request URI->(Fi,,ri,modify) to the CSS. PrepareUpdate(null,i,delete)URI . For deleting blocks, the value of the parameter UT is set to be 'delete'. The data owner just needs to delete the i -th row in ORT and move forward the entries from the (i 1)-th row to the tail in order. The corresponding BP values of the moved entries decrease 1 while the BI and BV remain unchanged. After that, the d0ataowner sends the request URI ->(null,null,i,delete) to the CSS.

ExecUpdate(URI)->{Success,Fail}. When the CSS receives the update request URI from the data owner, it updates the file blocks and tags according to the value of URI . For the case of 'insert', the CSS inserts the new block Fi ' to the file and the new tag Ti ' to the tag list, which are both at the i position; For the case of 'modify', the CSS replaces the old version of the block and tag at the position i with the new input pair (Fi ',Ti ') ; and for the case of 'delete', CSS only needs to delete the block and tag pair at the position i . When finishing the work rightly, the CSS restatement When finishing the work rightly, the CSS returns Success to the data owner, otherwise returns Fail . For the case of 'insert', the CSS inserts the new block Fi ' to the file and the new tag Ti ' to the tag list, which are both at the i position;

For the case of 'modify', the result



BP	BI	BV
1	1	1
2	2	1
3	3	1
⋮	⋮	⋮
n	n	1

BP	BI	BV
1	1	1
2	2	1
3	n+1	1
4	3	1
⋮	⋮	⋮
n+1	n	1

BP	BI	BV
1	1	1
2	2	1
3	3	2
⋮	⋮	⋮
n	n	1

BP	BI	BV
1	1	1
2	2	1
3	4	1
⋮	⋮	⋮
n-1	n	1

5.1OPTIMIZED IMPLEMENTATION OF ORT

It reveals the changing process of the ORT for different types of dynamic operations. Obviously, ORT is a linear data structure. So array and linked list are two traditional means for implementing ORT . However, using array has advantage over the element location but brings great costs on 'insert' and 'modify' operations, which have to copy and move all the elements behind the insert or modify index. Using linked list to realize ORT will remove the cost element coping and moving and just needs to move node pointers which spends nearly negligible cost. But it increases

massive overhead on nodes location for retrieving, inserting and modifying elements especially when the data size is big. Thus, to get better accessing efficiency, we present a novel hybrid data structure for realizing ORT, which is composed of array and doubly linked list. We

integrate the array's merit to narrow the range of . But it increases massive overhead on nodes location for retrieving, inserting and modifying elements especially when the data size is big. Thus, to get better accessing efficie ncy, we present a novel hybrid data structure for realizing ORT, which is composed of array and doubly linked list. We integra te the array's merit to narrow the range of values.entries of ORT to be nodes of the list with the same order. Since the doubly linked list is an ordered linear structure which implicitly contains the BP values of the nodes, each node only needs to store the BI and BV values. And then we split the entire doubly linked list into a number of sub-lists with constant length and create a pointer array to store the head node of each sub-list. Suppose the count of all nodes is N
, the length for the sub-list is L , the size for the pointer array is W
, then it has⌈ N L ⌉.Locate position. When locating the node at position V , we first compute the index W ' of the sub-list which
the node belongs to by the equation W ' ⌊ (V 1) L⌋, and then move
the node pointer backward by (V W ' L 1) times from the head node.
Compared with linked list, we can use the hybrid data structure to reduce the average location cost on pointer movement from N 2 to L 2 .Support operations. Upon fixing the operation position, we can insert or delete node easily which only need to move pointers for four times. Reconstruct. After inserting or deleting nodes, the length of the doubly linked list will change. In order to keep the constant length of the sub-lists, it requires to reset the head node of the sub-lists behind the operation position. Thus, along with the overhead on locating, the average costs on inserting and deleting node is (W L) 2, which is very small compared with linked list.Fig.4 demonstrates the detailed process for dynamic operations on the hybrid data structure, in which PA denotes the pointer array, each node contains the values of BI and Jointed by '& ' and the length of the sub-list is set to 100.

6. CONCLUSION

In this paper, we study the issue for integrity checking of data files outsourced to remote server and propose an efficient secure RDPC protocol with data dynamic. Our scheme employs a homomorphic hash function to verify the integrity for the files stored on remote server, and reduces the storage costs and computation costs of the data owner. We design a new lightweight hybrid data structure to support dynamic operations on blocks which incurs minimum computation
costs by decreasing the number of node shifting. Using our new data structure, the data owner can perform insert, modify or delete operation on file blocks with high efficiency. The presented scheme is proved secure in existing security model. We evaluate the performance in term of community cost, computation cost and storage cost. The experiments results indicate that our scheme is practical in
cloud storage.

REFERENCES

[1] Venugopal, J. Broberg, and I. Brandic,“Cloudcomputing and emerging IT platforms: Vision, hype, and reality fordelivering computing as the 5th utility,” Future Gener. Comp. Sy., vol.2 5, no. 6, pp. 599 – 616, 2009.

[2] H. Qian, J. Li, Y. Zhang and J. Han, “Privacy preserving personal healthrecord using multi-authority attribute-based encryption withrevocation,” Int. J. Inf. Secur., vol. 14, no. 6, pp. 487-497, 2015.

[3] J. Li, W. Yao, Y. Zhang, H. Qian and J. Han, “Flexible and fine-grained attribute-based data storage in cloud computing,” IEEE Trans. Service Comput., DOI: 10.1109/TSC.2016.2520932.

[4] J. Li, X. Lin, Y. Zhang and J. Han, “KSF-OABE: outsourced attribute-based encryption with keyword search function for cloud storage,” IEEE Trans. Service Comput., DOI: 10.1109/TSC.2016. 2542813.

[5] J. Li, Y. Shi and Y. Zhang, “Searchable ciphertext-policy attribute-based encryption with revocation in cloud storage,” Int. J. Commun. Syst., DOI: 10.1002/dac.2942.

[6] J.G. Han, W. Susilo, Y. Mu and J. Yan, “Privacy-Preserving Decentralized DKiesytr-iPboultiecdy SAystttreimbust,e v-Bola. s2e3d, nEon.1c1ry, pptpio. n2,1”5 0IE-2E1E62 ,T 2r0an1s2