www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 6 Issue 5 May 2017, Page No. 21496-21501
Index Copernicus value (2015): 58.10 DOI: 10.18535/ijecs/v6i5.51

DeyPoS: Deduplicatable Dynamic Proof of Storage for Multi-User Environment

Prof. Ashok Kumar Kalal, Kunal Jondhalekar , Piyush Kumar, Pavan Kumar Pandey, Himanshu Thakur
Jha hodcomp.acem@gmail.com
kunal.jondhalekar@gmail.com
piyushjha81@gmail.com
Pandeypavan03@gmail.com
hthakur5789@gmail.com

Abstract – *DeyPos is a useful cryptographic primitive that permits a user to check the integrity and systematically update the files in a cloud server. There has been many solutions proposed for Dynamic Proof of Storage in singleuser environment but for multi user problems is still unsolvable. A multi-user cloud storage system needs the secure client side cross user deduplication technique, which allows a user to stop the uploading process and gain the ownership of the files immediately, when other owners of the same files have uploaded them to the cloud server.As we know, none of the existing dynamic PoSs can support this technique. In this paper, we elaborate the concept of deduplicatable dynamic proof of storage and propose an efficient construction called DeyPoS, to achieve dynamic Proof of Storage and secure cross-user deduplication, simultaneously.To build a novel tool called Homomorphic Authenticated Tree (HAT) to address challenges such as structure diversity and private tag generation.Hence we prove the security of our construction, and the theoretical analysis and experimental results show that our research is practically valid and applicable.*

KEYWORDS: *Cloud storage, dynamic proof of storage, deduplication.*

1. INTRODUCTION

Deduplicatable Dynamic proof of storage is a part of data outsourcing which is widely used by organisations such as Amazon, Google and Microsoft.Researchers introduced Proof of Storage to check the truthfullness of the files without downloading them from the cloud server.In this scheme a tag which is associated with block verifies the integrity of that block. When a user uploads a file then he/she becomes the uploader of the file but, if uploading same file is attempted by any other user then the system stops the upload of that file and gives the access of the file which has already been uploaded by the other user.This process is done by key value matching. It solves major problems such as private tag generation.This scheme reduces unnecessary computation and provides efficient storage for cloud server.

STORAGE outsourcing is becoming more and more attractive to both industry and academia due to the advantages of low cost, high accessibility, and easy sharing. As one of the storage outsourcing forms, cloud storage gains wide attention in recent years [1] [2]. Many companies, such as Amazon, Google, and Microsoft, provide their own cloud storage services, where users can upload their files to the servers, access them from various devices, and share them with the others. Although cloud storage services are widely adopted in current days, there still remain many security issues and potential threats

Data integrity is one of the most important properties when a user outsources its files to cloud storage. Users should be convinced that the files stored in the server are not tampered. Traditional techniques for protecting data integrity, such as message authentication codes (MACs) and digital signatures, require users to download all of the files from the cloud server for verification, which incurs a heavy communication cost [5]. These techniques are not suitable for cloud storage services where users may check the integrity frequently, such as every hour [6]. Thus, researchers introduced Proof of Storage (PoS) [7] for checking the integrity without downloading files from the cloud server. Furthermore, users may also require several dynamic operations, such as modification, insertion, and deletion, to update their files, while maintaining the capability of PoS. Deypos is proposed for such dynamic operations.

To understand the following contents, we present details about PoS and dynamic PoS. In these schemes each block of a file is attached a (cryptographic) tag which is used for verifying the integrity of that block. When a verifier wants to check the integrity of a file, it randomly selects some block indexes of the file, and sends them to the cloud server. According to these challenged indexes, the cloud server returns the corresponding blocks along with their tags. The verifier checks the block integrity and index correctness. The former can be directly guaranteed by cryptographic tags. In this schemes , the block index is encoded into its tag.. However, dynamic PoS cannot encode the block indexes into tags, since the dynamic operations may change many indexes of non-updated blocks, which incurs unnecessary computation and communication cost. For example, there is a file consisting of 1000 blocks, and a new block is inserted behind the second block of the file. Then, 998 block indexes of the original file are changed, which means the user has to generate and send 999 tags for this update. Authenticated structures are introduced in dynamic PoSs to solve this challenge.

2. Related work

The main idea of PoS is to randomly choose a few data blocks as the challenge. Then, the cloud server returns the challenged data blocks and their tags as the response. Since the data blocks and the tags can be combined via homomorphic functions, the communication costs are reduced. The subsequent works extended the research of PoS, but those works did not take dynamic operations into account. Erway et al. and later works focused on the dynamic data. Among them, the scheme in is the most efficient solution in practice. However, the scheme is stateful, which requires users to maintain some state information of their own files locally. Hence, it is not appropriate for a multiuser environment.The Merkle tree without the help from the cloud server, which is a big challenge in dynamic PoS. The scheme employs a deterministic proof algorithm which indicates that every file has a deterministic short proof. Thus, anyone who obtains this proof can pass the verification without possessing the file locally. Other deduplication schemes for encrypted data were proposed for enhancing the security and efficiency. Note that, all existing techniques for cross-user deduplication on the client-side were designed for static files. Once the files are updated, the cloud server has to regenerate the complete authenticated structures for these files, which causes heavy computation cost on the server-side.

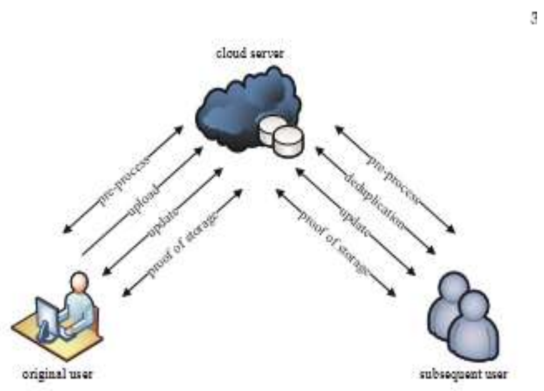


Fig. 2. The system model of deduplicatable dynamic PoS

Deduplication in these scenarios is to deduplicate files among different groups. Unfortunately, these schemes cannot support deduplication due to structure diversity and private tag generation. In this paper, we consider a more general situation that every user has its own files separately. Hence, we focus on a deduplicatable dynamic PoS scheme in multiuser environments. The major techniques used in PoS and dynamic PoS schemes are homomorphic Message Authentication Codes [40] and homomorphic signatures [41] [42]. With the help of homomorphism, the messages and MACs/signatures in these schemes can be compressed into a single message and a single MAC/signature. Therefore, the communication cost can be dramatically reduced. A brief survey of homomorphic MACs and signatures could be referred.

1.1 Contributions

To the best of our knowledge, this is the first work to introduce a primitive called deduplicatable dynamic Proof of Storage which solves the structure diversity and private tag generation challenges. In contrast to the existing authenticated structures, such as skip list and Merkle tree we design a novel authenticated structure called Homomorphic Authenticated Tree (HAT), to reduce the communication cost in both the proof of storage phase and the deduplication phase with similar computation cost. Note that HAT can support integrity verification, dynamic operations, and cross-user deduplication with good consistency. We propose and implement the first efficient construction of deduplicatable dynamic PoS called DeyPoS, which supports unlimited number of verification and update operations.

3 DEDUPLICATABLE DYNAMIC POS

3.1 System Model

Our system model considers two types of entities: the cloud server and users, as shown in Fig. 2. For each file, original user is the user who uploaded the file to the cloud server, while subsequent user is the user who proved the ownership of the file but did not actually upload the file to the cloud server. There are five phases in a deduplicatable dynamic PoS system: pre-process, upload, deduplication, update, and proof of storage. In the pre-process phase, users intend to upload their local files. The cloud server decides whether these files should be uploaded. If the upload process is granted, go into the upload phase; otherwise, go into the deduplication phase. In the upload phase, the files to be uploaded do not exist in the cloud server. The original users encodes the local files and upload them to the cloud server. In the deduplication phase, the files to be uploaded already exist in the cloud server. The subsequent users possess the files locally and the cloud server stores the authenticated structures of the files. Subsequent users need to convince the cloud server that they own the files without uploading them to the cloud server. Note that, these three phases (pre-process, upload, and deduplication) are executed only once in the life cycle of a file from the perspective of users. That is, these three phases appear only when users intend to upload files. If these phases terminate normally, i.e., users finish uploading in the upload phase, or they pass the verification in the deduplication phase, we say that the users have the ownerships of the files.

Note that, these three phases (pre-process, upload, and deduplication) are executed only once in the life cycle of a file from the perspective of users. That is, these three phases appear only when users intend to upload files. If these phases terminate i.e., users done with the uploading in the uploading of file, or they pass the validation in the deduplication phase, we assume that users have the ownership of the files. In the update phase, users may modify, insert, or delete some blocks of the files. Then, they update the corresponding parts of the encoded files and the authenticated structures in the cloud server, even the original files were not uploaded by themselves. Note that, users can update the files only if they have the ownerships of the files, which means that the users should upload the files in the upload phase or pass the verification in the deduplicationphase. For each update, the cloud server has to reserve the original file and the authenticated structure if there exist other owners, and record the updated part of the file and the authenticated structure. This enables users to update a file concurrently in our model, since each update is only "attached" to the original file and authenticated structure. In the proof of storage phase, user can check the integrity of metadata locally. The files may not be uploaded by these users, but they pass the deduplication phase and prove that they have the ownerships of the files.

3.2 Threat Model

We present the threat model briefly as follows. The cloud server and users do not fully trust each other. A malicious user may cheat the cloud server by claiming that it has a certain file, but it actually does not have it or only possesses parts of the file. A malicious cloud server may try to convince users that it faithfully stores files and updates them, whereas the files are damaged or not up-to-date. The goal of deduplicatable dynamic PoS is to detect these misbehaviors with overwhelming probability. The formal threat model is described in Section 2.4 via various security definitions.

4. HOMOMORPHIC AUTHENTICATED TREE

To implement an efficient deduplicatable dynamic PoS scheme, we design a novel authenticated structure called homomorphic authenticated tree (HAT). A HAT is a binary tree in which each leaf node corresponds to a data block. Though HAT does not have any limitation on the number of data blocks, for the sake of description simplicity, we assume that the number of data blocks n is equal to the number of leaf nodes in a full binary tree. Thus, for a file F = (m1,m2,m3,m4) where mι represents the ι-th block of the file, we can construct a tree as shown in Fig. 1a. Each node in HAT consists of a four-tuple νi = (i,li,vi,ti). i is the unique index of the node. The index of the root node is 1, and the indexes increases from top to bottom and from left to right. li denotes the number of leaf nodes that can be reached from the i-th node. vi is the version number of the i-th node. ti represents the tag of the i-th node. When a HAT is initialized, the version number of each leaf is 1, and the version number of each non-leaf node is the sum of that of its two children. For the i-th node, mi denotes the combination of the blocks corresponding to its leaves. We require that for any node νi and its children ν2i and ν2i+1, F(mi) = F(m2i ⊙ m2i+1) = F(m2i) ⊗ F(m2i+1) holds, where ⊙ denotes the combination of m2i and m2i+1, and ⊗ indicates the combination of F(m2i) and F(m2i+1), which is why we call it a "homomorphic" tree.

5. Algorithm

Cloud computing gives boundless virtualized plan of action to client as administra- tions over the entire web while concealing the stage and executing subtle elements. Distributed storage administration is the administration of evergreen expanding mass of information. To make information administration adaptable in distributed com- puting, deduplication has been a customary method. Information pressure strategy is utilized for dispensing with the copy duplicates of rehashed information in dis- tributed storage to decrease the information duplication. This method is utilized to speedup stockpiling use furthermore be connected to network information exchanges to lessen the quantity of bytes that must be sent.

Algorithm 1 Path search algorithm

```
1:  procedure PATH($T, \mathcal{I}$)
2:      for $\iota \in \mathcal{I}$ do
3:          if $\iota > l_1$ then
4:              return 0
5:          $i_\iota \leftarrow 1, ord_\iota \leftarrow \iota$
6:      $\rho \leftarrow \{1\}, st \leftarrow$ TRUE
7:      while $st$ do
8:          $st \leftarrow$ FALSE
9:          for $\iota \in \mathcal{I}$ do
10:             if $l_{i_\iota} = 1$ then
11:                 continue
12:             else if $ord_\iota \leq l_{2i_\iota}$ then
13:                 $i_\iota \leftarrow 2i_\iota$
14:             else
15:                 $ord_\iota \leftarrow ord_\iota - l_{2i_\iota}, i_\iota \leftarrow 2i_\iota + 1$
16:             $\rho \leftarrow \rho \cup \{i_\iota\}$
17:             if $l_{i_\iota} > 1$ then
18:                 $st \leftarrow$ TRUE
19:     return $\rho$
```

Algorithm 2 Sibling search algorithm

```
1:  procedure SIBLING($\rho$)
2:      $\psi \leftarrow \emptyset, \rho \leftarrow \rho \setminus \{1\}, \varrho \leftarrow \emptyset, i \leftarrow 1$
3:      while $\rho \neq \emptyset \vee \varrho \neq \emptyset$ do
4:          if $2i \in \rho$ then
5:              $i \leftarrow 2i, \rho \leftarrow \rho \setminus \{i\}$
6:              if $i+1 \in \rho$ then
7:                  $\varrho \leftarrow \varrho \cup \{(i+1, \mathrm{FALSE})\}, \rho \leftarrow \rho \setminus \{i+1\}$
8:              else
9:                  $\varrho \leftarrow \varrho \cup \{(i+1, \mathrm{TRUE})\}$
10:         else if $2i+1 \in \rho$ then
11:             $i \leftarrow 2i+1, \rho \leftarrow \rho \setminus \{i\}, \psi \leftarrow \psi \cup \{i-1\}$
12:         else if $\varrho \neq \emptyset$ then
13:             pop the last inserted $(\alpha, \beta)$ in $\varrho$
14:             $i \leftarrow \alpha$
15:             if $\beta = \mathrm{TRUE}$ then
16:                 $\psi \leftarrow \psi \cup \{i\}$
17:     return $\psi$
```

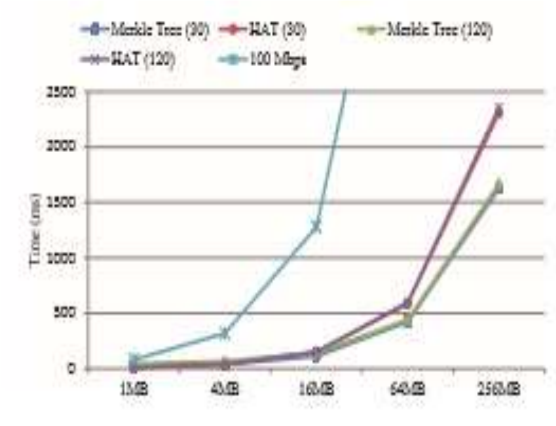
6. Result & Analysis

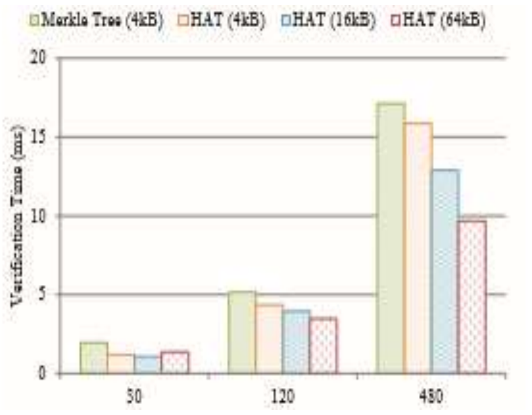
We first evaluate the cost in the upload phase. The initialization time is similar in all schemes. For example, the initialization time for constructing Merkle tree and HAT is 6.7s and 7.9s, respectively, for a 1GB file of 4kB block size. The storage cost of the client is O(1), and the storage cost of the server is shown in Fig. 4. The authenticator size of HAT is lager than that of the Merkle tree. However, when Merkle tree is employed in PoS scheme, it requires more space for storing tags of file blocks. As a result, the storage cost of our scheme is similar to other Merkle tree based PoS schemes. When the block size is 4kB, the authenticator size is less than 3% of the file size in our scheme.

Next, we evaluate the cost in the deduplication phase. Fig. 5 presents the communication cost when the file size is 1GB. The communication cost considers the data sent from users and the data sent from the cloud server.The communication cost in our scheme is more efficient than the cost of Merkle tree based schemes, since users has to send all challenged file blocks to the cloud server for generating leaf nodes of Merkle tree in those schemes. When the block size is 4kB and the number of the challenged blocks is 480, the communication cost of Merkle tree based solution [15] is almost 2MB, while the cost of DeyPoS is 104kB. Fig. 6 shows the communication cost of different file sizes, where the block size is fixed on 4kB. When the number of challenged file blocks are fixed, the communication cost stays at a steady level in Merkle tree based schemes since the major cost is to transmit the corresponding file blocks. However, the communication cost grows logarithmically with respect to the file size because the number of nodes in the sibling set grows logarithmically.

Finally, we show the experimental results in the proof of storage phase. Since the challenge size which is the size of data sent from users is constant and negligible (less than 100B) in both DeyPoS and Merkle tree based solutions, Fig. 10 only depicts the proof size which is the amount of data sent from the cloud server. DeyPoS requires a lower cost than Merkle tree based scheme because the tags in HAT are homomorphic. When we challenge 480 blocks , the proof size is less than 80kB, which is negligible small in practice. Fig. 11 presents the proof size of different file sizes, where the block size is fixed on 4kB. Obviously, DeyPoS requires less bandwidths in all situations. When the block size is 4kB [5] [14], the block size is less than 10kB.

As a consequence, our scheme, DeyPoS, which is based on a HAT, reduces the communication cost in both the deduplication phase and the proof of storage phase. The computation cost is as efficient as the one in Merkle tree based dynamic PoSs.





7. Conclusion & Future Work

We proposed the comprehensive requirements in multi-user cloud storage systems and introduced the model of deduplicatable dynamic PoS. We designed a novel tool called HAT which is an efficient authenticated structure. Based on HAT, we proposed the first practical deduplicatable dynamic PoS scheme called DeyPoS and proved its security in the random oracle model. The theoretical and experimental results show that ourDeyPoS implementationis efficient, especially when the file size and the number of the challenged blocks are large.

8. References

[1] S. Kamara and K. Lauter, “Cryptographic cloud storage,” in Proc. of FC, pp. 136–149, 2010.

[2] Z. Xia, X. Wang, X. Sun, and Q. Wang, “A Secure and Dynamic Multi-Keyword Ranked Search Scheme over Encrypted Cloud Data,” IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 2, pp. 340–352, 2016.

[3] Z. Xiao and Y. Xiao, “Security and privacy in cloud computing,” IEEE Communications Surveys Tutorials, vol. 15, no. 2, pp. 843–859, 2013.

[4] C. A. Ardagna, R. Asal, E. Damiani, and Q. H. Vu, “From Security to Assurance in the Cloud: A Survey,” ACM Comput. Surv., vol. 48, no. 1, pp. 2:1–2:50, 2015.

[5] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, “Provable data possession at untrusted stores,” in Proc. of CCS, pp. 598–609, 2007. [6] G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, “Scalable and Efficient Provable Data Possession,” in Proc. of SecureComm, pp. 1–10, 2008.

[7] G. Ateniese, S. Kamara, and J. Katz, “Proofs of storage from homomorphic identification protocols,” in Proc. of ASIACRYPT, pp. 319–333, 2009.

[8] C. Erway, A. Ku¨pcu¨, C. Papamanthou, and R. Tamassia, “Dynamic provable data possession,” in Proc. of CCS, pp. 213–222, 2009.

[9] R. Tamassia, “Authenticated Data Structures,” in Proc. of ESA, pp. 2–5, 2003. [10] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, “Enabling public verifiability and data dynamics for storage security in cloud computing,” in Proc. of ESORICS, pp. 355–370, 2009.