International Journal on Future Revolution in Computer Science & Communication Engineering
Volume: 4 Issue: 12

ISSN: 2454-4248
121 – 125

Secure Multilevel Data Authentication System in Cloud Environment

M. Suuny Kousik Reddy,
Bachelor of Technology,
Electronics and Communication Engineering,
Yogi Vemana University
mskousikreddy1997@gmail.com

Dr. K. Venkata Ramanaiah
Professor
Electronics and Communication Engineering,
Yogi Vemana University,
ramanaiahkota@gmail.com

Abstract **-** Dynamic Proof of Storage is a useful cryptographic primitive that enables a user to check the integrity of outsourced files and to efficiently update the files in a cloud server. Though researchers have planned several dynamic PoS schemes in single user environments, the matter in multi-user environments has not been investigated sufficiently. A sensible multi-user cloud storage system wants the secure client-side cross-user de-duplication technique, that permits a user to skip the uploading method and procure the possession of the files now, once alternative house owners of an equivalent files have uploaded them to the cloud server. To the simplest of our data, none of the present dynamic PoS will support this system. during this paper, we have a tendency to introduce the conception of de-duplicatable dynamic proof of storage associated propose an economical construction referred to as DeyPoS, to realize dynamic PoS and secure cross-user duplication, at the same time. Considering the challenges of structure diversity and personal tag generation, we have a tendency to exploit a unique tool referred to as Homomorphic Authenticated Tree (HAT). We have a tendency to prove the protection of our construction, and therefore the theoretical analysis and experimental results show that our construction is economical in follow.

Keywords*: Deduplication, Proof of ownership, Dynamic proof of storage, Cloud Computing.*

*****

I. Introduction

To the best of our knowledge, none of the existing dynamic PoSs can support this technique. In this paper, we introduce the concept of deduplicatable dynamic proof of storage and propose an efficient construction called DeyPoS, to achieve dynamic PoS and secure cross-user deduplication, simultaneously. Considering the challenges of structure diversity and private tag generation, we exploit a novel tool called HAT. We prove the security of our construction, and the theoretical analysis and experimental results show that our construction is efficient in practice.

To better understand the following contents, we present more details about PoS and dynamic PoS. In these schemes, each block of a file is attached a 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. How to deal with the latter is the major difference between PoS and dynamic PoS. In most of the PoS schemes, the block index is “encoded” into its tag, which means the verifier can check the block integrity and index correctness simultaneously. 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. As a result, the tags are attached to the authenticated structure rather than the block indexes.

Taking the Merkle tree in Fig. 1a as an example Merkle tree is one of the most efficient authenticated structures in dynamic PoS, the tag corresponding to the second file block involves the index of the Merkle tree node v5 that is 5, rather than 2. When a new block is inserted behind the second file block, the authenticated structure turns into the structure. Then, the index in the tag corresponding to the second file block changes, and the user only has to generate 2 tags for this update.

Users ought to be convinced that the files keep within the server don't seem to be tampered. Ancient techniques for safeguarding knowledge integrity, like MACs and digital signatures need users to transfer all of the files from the cloud server for verification that incurs a significant communication value. These techniques don't seem to be appropriate for cloud storage services wherever users could check the integrity oftentimes, like each hour. Thus, researchers introduced Proof of Storage (PoS) for checking

 ISSN: 2454-4248
121 – 125

the integrity while not downloading files from the cloud server. What is more, users may need many dynamic operations, like modification, insertion, and deletion, to update their files, whereas maintaining the potential of PoS.

Dynamic PoS is projected for such dynamic operations. In distinction with PoS, dynamic PoS employ structures, like the Merkle tree. Thus, once dynamic operations are dead, users regenerate tags (which are used for integrity checking, like MACs and signatures) for the updated blocks solely, rather than create for all blocks. To rised perceive the subsequent contents. We tend to gift additional details concerning PoS and dynamic PoS. In these schemes, every block of a file is hooked up a (cryptographic) tag that is employed for substantiating the integrity of that block. Once a champion desires to ascertain the integrity of a file, it every which way selects some block indexes of the file, and sends them to the cloud server. Consistent with these challenged indexes, the cloud server returns the corresponding blocks beside their tags. The champion checks the block integrity and index correctness.

The previous are often directly bonded by cryptanalytic tags. a way to affect the latter is that the major distinction between PoS and dynamic PoS In most of the PoS schemes, the block index is "encoded" into its tag, which implies the champion will check the block integrity and index correctness at the same time. However, dynamic PoS cannot cypher the block indexes into tags, since the dynamic operations could modification several indexes of non-updated blocks that incurs reserve computation and communication value.

II. Related work

Proof of Storage

The idea behind PoS is to choose few data blocks at random, 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.

This PoS concept was basically introduced by Ateniese *et al* and *Kaliski*. Ateniese [1] introduced introduce a model for provable data possession (PDP) that allows a client that has stored data at an untrusted server to verify that the server possesses the original data without retrieving it. The model generates probabilistic proofs of possession by sampling random sets of blocks from the server, which drastically reduces I/O costs. The client maintains a constant amount of metadata to verify the proof. The challenge/response protocol transmits a small, constant amount of data, which minimizes network communication.

Kaliski [2] introduced a POR (proofs of retrievability) scheme enables an archive or back-up service (prover) to produce a concise proof that a user (verifier) can retrieve a target file F, that is, that the archive retains and reliably transmits file data sufficient for the user to recover F in its entirety. A POR may be viewed as a kind of cryptographic proof of knowledge (POK), but one specially designed to handle a *large* file (or bit string) F. Explored POR protocols here in which the communication costs, number of memory accesses for the prover, and storage requirements of the user (verifier) are small parameters essentially independent of the length of F. To conduct and verify POR, users need to be equipped with devices that have network access, and that can tolerate the (non-negligible) computational overhead incurred by the verification process. This clearly hinders the large-scale adoption of POR by cloud users, since many users increasingly rely on portable devices that have limited computational capacity, or might not always have network access.

Dynamic Proof of Storage

Proofs of retrievability allow a client to store her data on a remote server (e.g., "in the cloud") and periodically execute an efficient audit protocol to check that all of the data is being maintained correctly and can be recovered from the server. For efficiency, the computation and communication of the server and client during an audit protocol should be significantly smaller than reading/transmitting the data in its entirety. Although the server is only asked to access a few locations of its storage during an audit, it must maintain full knowledge of all client data to be able to pass.

Starting with the work of Juels and Kaliski all prior solutions to this problem crucially assume that the client data is static and do not allow it to be efficiently updated. Indeed, they all store a redundant encoding of the data on the server, so that the server must delete a large fraction of its storage to „lose" any actual content. Unfortunately, this means that even a single bit modification to the original data will need to modify a large fraction of the server storage, which makes updates highly inefficient. Overcoming this limitation was left as the main open problem by all prior works.

The work [6], gives the first solution providing proofs of retrievability for dynamic storage, where the client can perform arbitrary reads/writes on any location within her data by running an efficient protocol with the server. At any point in time, the client can execute an efficient audit protocol to ensure that the server maintains the latest version of the client data. The computation and communication complexity of the server and client in our protocols is only polylogarithmic in the size of the client"s data. The starting point of our solution is to split up the data into small blocks and redundantly encode each block of data individually, so that an update inside any data block only affects a few code word symbols. The main difficulty is to prevent the server from identifying and deleting too many code word symbols

 ISSN: 2454-4248
121 – 125

belonging to any single data block. We do so by hiding where the various code word symbols for any individual data block are stored on the server and when they are being accessed by the client, using the algorithmic techniques of oblivious RAM.

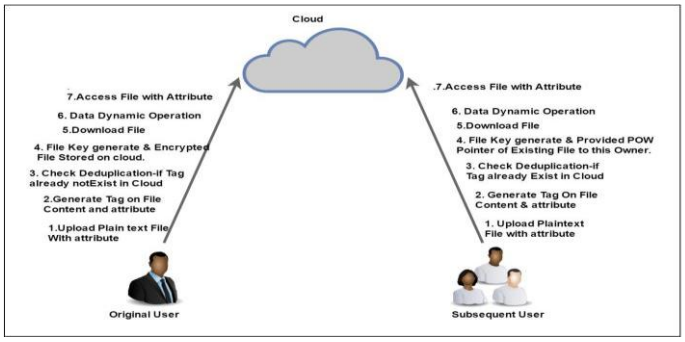
Deduplicatable Dynamic Proof of Storage

Halevi *et al.* [9] introduced the concept of proof of ownership which is a solution of cross-user deduplication on the client-side. It requires that the user can generate the Merkle tree without the help from the cloud server, which is a big challenge in dynamic PoS. Xu *et al.* [10] proposed a client-side deduplication scheme for encrypted data, but 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. 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.

Problem Statement

Present dynamic PoSs, a tag used for integrity verification is generated by the secret key of the uploader. Thus, other owners who have the ownership of the file but have not uploaded it due to the cross-user deduplication on the client-side, cannot generate a new tag when they update the file. In this situation, the dynamic PoSs would fail.

System Architecture



Here describing our system construction module, to evaluate and implement a deduplicatable dynamic proof of storage and propose an efficient construction called DeyPoS. For this purpose we develop User and Cloud entities. In User entity, a user can upload a new File, Update uploaded File blocks and a user can deduplicate other users File by using deduplicatable dynamic proof of storage.

Our system model considers two types of entities: the cloud server and users. 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.

Implementation Techniques Procedure

Block Generation

In this module, we develop the Block Generation process. 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 Deduplication phase. Though we can create n-blocks in this module, we split the files into 3 Blocks. The Blocks for files are divided equally accordingly and then the blocks are uploaded in the Cloud Server too.

Deduplicatable Dynamic POS

In this module we focus on a Deduplicatable Dynamic PoS scheme in multiuser environments. Deduplicatable Dynamic Proof of Storage is used to deduplicate the other users file with proper authentication but without uploading the same file. Deduplicatable Dynamic Proof of Storage (deduplicatable dynamic PoS), which solves the structure diversity and private tag generation challenges.

The main process of this module is 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 encode 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.

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 deduplication phase. For each update, the cloud server has to reserve the

 ISSN: 2454-4248
121 – 125

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, users only possess a small constant size metadata locally and they want to check whether the files are faithfully stored in the cloud server without downloading them. 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.

Proposed Functions

We propose a concrete scheme of deduplicatable dynamic PoS called DeyPoS. It consists of five algorithms.

- Init
- Encode
- Deduplicate
- Update
- Check.

Functional Procedure

Init()

Cloud Server and user register the Unique ID for initialization. Original registered user can upload the files to the server. Subsequent user register the Unique ID and its registered Password for access the uploaded files.

Encode()

Original users before upload the Files to the Cloud server a encoding process done. In the Encode process the HomographicAuthenticate Tree logic be applied.

Deduplicate()

Detect the duplicate of the ID by verify the database by the Unique Deypos ID and the generated

unique password. If ID and password validated success the subsequent users can access the file

rights otherwise ID consider as Duplication.

Update()

Original users upload the file to the cloud server and then updated. File upload with unique ID for access the files by the subsequent users.

Check()

Check the Validation and verification process for the Files upload and download. Cloud server

Performance and No. of deduplication trials happen when try to access the server files.

Homomorphic Authentication Tree

To implement an efficient deduplicatable dynamic PoS scheme, we design a novel authenticated structure called 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 = (m_1, m_2, m_3, m_4) where m_l represents the ι-th block of the file. Each node in HAT consists of a four-tuple Vi = (I, l_i, v_i, t_i). *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. Denotes the number of leaf nodes that can be reached from the i-th node. is the version number of the i th node. 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, denotes the combination of the blocks corresponding to its leaves. The tag is computed from F(m_i), where F denotes a tag generation function. We require that for any node v_i and its children v_{2i} and v_{2i} +1, F(m_i) = F(m_{2i} ⊙ m_{2i} +1) = F(m_{2i}) ⊗ F(m_{2i} +1) holds, where ⊙ denotes the combination of m_{2i} and m_{2i} +1, and ⊗ indicates the combination of F(m_{2i}) and F(m_{2i} +1), which is why we call it a "homomorphic" tree.

Performance Analysis

We first evaluate the cost in the upload phase. Bellow figure represents the initialization time for constructing Merkle trees and HATs with different sizes of files and blocks. 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.

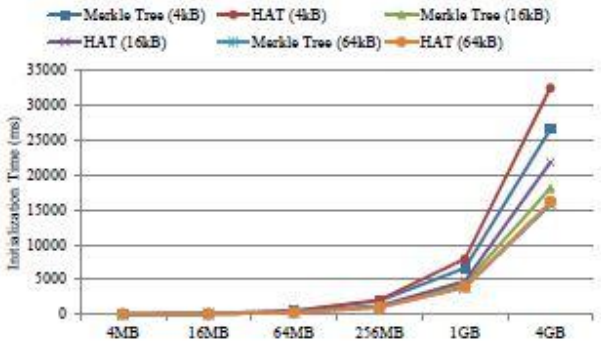


Fig: Initialization time in different file sizes

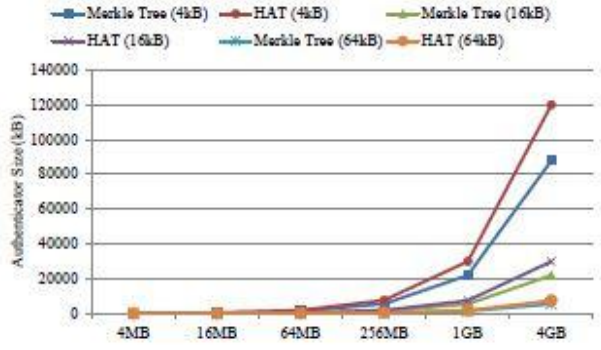


Fig: Authenticator size in different file sizes

 ISSN: 2454-4248
121 – 125

The storage cost of the client is O(1), and the storage cost of the server is shown in above figure. The authenticator size of HAT is larger 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.

III. Conclusion

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 our DeyPos implementation is efficient, especially when the file size and the number of the challenged blocks are large. The first realistic deduplicatable dynamic PoS scheme which makes use of complete necessities in multi-consumer cloud storage systems and proved its security within the random oracle model. The theoretical and experimental results show that the procedure is efficient, peculiarly when the file dimension and the number of the challenged blocks are large.

References

[1] 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.

[2] A. Juels and B. S. Kaliski, Jr., “PORs: Proofs of retrievability for large files,” in Proc. of CCS, pp. 584 – 597,2007.

[3] F. Armknecht, J.-M. Bohli, G. O. Karame, Z. Liu, and C. A. Reuter, “Outsourced proofs of retrievability,” in Proc. of CCS, pp. 831–843, 2014.

[4] H. Shacham and B. Waters, “Compact proofs of retrievability,” in Proc. of ASIACRYPT, pp. 90–107, 2008.

[5] Y. Dodis, S. Vadhan, and D. Wichs, “Proofs of retrievability via hardness amplification,” in Proc. of TCC, pp. 109–127, 2009.

[6] Z. Mo, Y. Zhou, and S. Chen, “A dynamic proof of retrievability (PoR) scheme with o(logn) complexity,” in Proc. of ICC, pp. 912–916, 2012.

[7] E. Shi, E. Stefanov, and C. Papamanthou, “Practical dynamic proofs of retrievability,” in Proc. of CCS, pp. 325–336, 2013.

[8] D. Cash, A. K¨upc¸ ¨u, and D. Wichs, “Dynamic proofs of retrievability via oblivious RAM,” in Proc. Of EUROCRYPT, pp. 279–295, 2013.

[9] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, “Proofs of ownership in remote storage systems,” in Proc. of CCS, pp. 491–500, 2011.

[10] J. Xu, E.-C. Chang, and J. Zhou, “Weak leakage-resilient client side deduplication of encrypted data in cloud storage,” in Proc. Of ASIACCS, pp. 195–206, 2013.

[11] Q. Zheng and S. Xu, “Secure and efficient proof of storage with deduplication,” in Proc. of CODASPY, pp. 1– 12, 2012.

[12] R. Du, L. Deng, J. Chen, K. He, and M. Zheng, “Proofs of ownership and retrievability in cloud storage,” in Proc. of TrustCom, pp. 328–335, 2014.

[13] B. Wang, B. Li, and H. Li, “Public auditing for shared data with efficient user revocation in the cloud,” in Proc. of INFOCOM, pp. 2904–2912, 2013.

[14] B. Wang, B. Li, and H. Li, “Oruta: privacy-preserving public auditing for shared data in the cloud,” IEEE Transactions on Cloud Computing, vol. 2, no. 1, pp. 43–56, 2014.

[15] J. Yuan and S. Yu, “Efficient public integrity checking for cloud data sharing with multi-user modification,” in Proc. of INFOCOM, pp. 2121–2129, 2014.