Deduplicatable Dynamic Proof of Storage for Multi-User Environments

Shrikala Upadhya[1], Dr. G T Raju[2]

4th Semester MTech, Department of CSE, RNSIT, Bengaluru, India
Professor, Department of CSE, RNSIT, Bengaluru, India

ABSTRACT

Dynamic Proof of Storage (PoS) is a useful cryptographic technique that enables a user to check the integrity of outsourced files and to efficiently update the files in a cloud server. Although many dynamic PoS schemes in single user environments were proposed by researchers, the problem in multi-user environments has not been investigated sufficiently. A multi-user cloud storage system requires secure client-side cross-user deduplication technique, which allows a user to skip the uploading process and immediately obtain the ownership of the files, when other owners of the same files have uploaded them to the cloud server. To the best of our knowledge, none of the existing dynamic PoSs can support this technique. In this work, the concept of deduplicatable dynamic proof of storage is introduced and proposed 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, exploited a novel tool called Homomorphic Authenticated Tree (HAT). Aim of this project is to prove the security and efficiency of this construction.

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

---

1. INTRODUCTION

Storage outsourcing is becoming more and more attractive 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. 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.

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. These techniques are not suitable for cloud storage services where users may check the integrity frequently, such as every hour. Thus, researchers introduced Proof of Storage (PoS) 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. The dynamic PoS system is proposed for such dynamic operations. In contrast with PoS, dynamic PoS employs authenticated structures, such as the Merkle tree.

2. RELATED WORK

A. 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.

Later [3][4][5] introduce the notion of outsourced proofs of retrievability (OPOR), in which users can task an external auditor to perform and verify POR with the cloud provider. Proposed POR scheme minimizes user effort, incurs negligible overhead on the auditor, and considerably improves over existing publicly verifiable POR. These above subsequent works extended the research of PoS but those works did not take dynamic operations into account.

B. ***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 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.

Later works [7][8] proposed a dynamic PoR scheme with constant client storage whose bandwidth cost is comparable to a Merkle hash tree, thus being very practical. The construction out performs the constructions of Stefanov et al. and Cash et al., both in theory and in practice. Compared with the existing dynamic PoR scheme, our worst case communication complexity is O(logn) instead of O(n). Among them, the scheme in [7] 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.

C. *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.

Zheng and Xu [11] proposed a solution called proof of storage with deduplication, which is the first attempt to design a PoS scheme with deduplication. Du *et al.* [12] introduced proofs of ownership and retrievability, which are similar to [11] but more efficient in terms of computation cost. Note that neither [11] nor [12] can support dynamic operations. Due to the problem of structure diversity and private tag generation, [11] and [12] cannot be extended to dynamic PoS.

Wang *et al.* [13] [14], and Yuan and Yu [15] considered proof of storage for multi-user updates, but those schemes focus on the problem of sharing files in a group. 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.

3. PROPOSED SYSTEM

Wang *et al.* [13] [14], and Yuan and Yu [15] considered proof of storage for multi-user updates, but those schemes focus on the problem of sharing files in a group. Deduplication in these scenarios is to deduplicate files among different groups. On comparison to all of the existing schemes, proposed scheme considers 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 and homomorphic signatures. 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.

Advantages

1) Proposed scheme introduced a primitive called deduplicatable dynamic Proof of Storage (Dey-PoS), which solves the structure diversity and private tag generation challenges.

2) 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.

3) Dey-PoS supports unlimited number of verification and update operations. The security of this construction is proved in the random oracle model, and the performance is analyzed theoretically and experimentally.

4. HOMOMORPHIC AUTHENTICATION 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 = (m_1 , m_2 , m_3, m_4) where m_ι represents the ι-th block of the file. Each node in HAT consists of a four-tuple V_i = (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. l_i denotes the number of leaf nodes that can be reached from the i-th node. v_i is the version number of the i-th node. t_i 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, m_i denotes the combination of the blocks corresponding to its leaves. The tag t_i 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($m_{2i} \odot m_{2i}$ +1) = F(m_{2i}) $\otimes$ F(m_{2i} +1) holds, where $\odot$ denotes the combination of m_{2i} and m_{2i} +1, and $\otimes$ indicates the combination of F(m_{2i}) and F(m_{2i} +1), which is why we call it a "homomorphic" tree.

5. SYSTEM ARCHITECTURE

System architecture considers two types of entities: the cloud server and users, as shown in Figure 1 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.

A. Pre-Process

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.

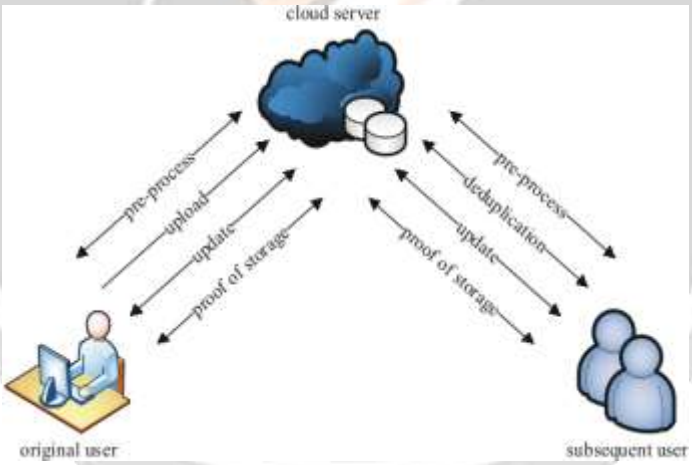

Fig-1: The system model of deduplicatable dynamic PoS

B. Upload

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.

C. Deduplication

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.

The above 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.

D. Update

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 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.

E. Proof of Storage

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. The update phase and the proof of storage phase can be executed multiple times in the life cycle of a file. Once the ownership is verified, the users can arbitrarily enter the update phase and the proof of storage phase without keeping the original files locally.

6. 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.

7. 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.