A PROFICIENT SECURE MECHANISM FOR RANDOM OWNERSHIP MANAGEMENT IN CLOUD BASED ENVIRONMENT

Mr. T.SANDEEP [1], Mrs. V. TEJASWINI[2*]

1 Final Year MCA Student, QIS College of Engineering and Technology, Ongole

2* Assistant Professor, MCA Dept., QIS College of Engineering and Technology, Ongole

Abstract: *Data deduplication is a technique for eliminating duplicate copies of data, and has been widely used in cloud storage to reduce storage space and upload bandwidth. However, there is only one copy for each file stored in cloud even if such a file is owned by a huge number of users. As a result, deduplication system improves storage utilization while reducing reliability. Furthermore, the challenge of privacy for sensitive data also arises when they are outsourced by users to cloud. Aiming to address the above security challenges, this paper makes the first attempt to formalize the notion of distributed reliable deduplication system. We propose new distributed deduplication systems with higher reliability in which the data chunks are distributed across multiple cloud servers. The security requirements of data confidentiality and tag consistency are also achieved by introducing a deterministic secret sharing scheme in distributed storage systems, instead of using convergent encryption as in previous deduplication systems. Security analysis demonstrates that our deduplication systems are secure in terms of the definitions specified in the proposed security model. As a proof of concept, we implement the proposed systems and demonstrate that the incurred overhead is very limited in realistic environments.*

Keywords: Data deduplication, Cloud Storage, Security model.

I. INTRODUCTION

Distributed computing is a field of computer science that studies distributed systems. A distributed system is a software system in which components located on networked computers communicate and coordinate their actions by passing messages. The components interact with each other in order to achieve a common goal. There are many alternatives for the message passing mechanism, including RPC-like connectors and message queues. Three significant characteristics of distributed systems are: concurrency of components, lack of a global clock, and independent failure of components. An important goal and challenge of distributed systems is location transparency. Examples of distributed systems vary from SOA-based systems to massively multiplayer online games to peer-to-peer applications.

A computer program that runs in a distributed system is called a distributed program, and distributed programming is the process of writing such programs.

Distributed computing also refers to the use of distributed systems to solve computational problems. In distributed computing, a problem is divided into many tasks, each of which is solved by one or more computers, which communicate with each other by message passing.

The word *distributed* in terms such as "distributed system", "distributed programming", and "distributed algorithm" originally referred to computer networks where individual computers were physically distributed within some geographical area. The terms are nowadays used in a much wider sense, even referring to autonomous processes that run on the same physical computer and interact with each other by message passing. While there is no single definition of a distributed system, the following defining properties are commonly used:

- There are several autonomous computational entities, each of which has its own local memory.
- The entities communicate with each other by message passing.

In this article, the computational entities are called *computers* or *nodes*.

A distributed system may have a common goal, such as solving a large computational problem.[1] Alternatively, each computer may have its own user with individual needs, and

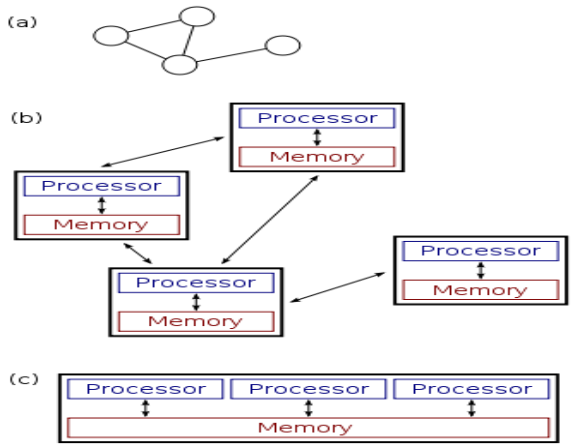
the purpose of the distributed system is to coordinate the use of shared resources or provide communication services to the users.

Other typical properties of distributed systems include the following:

- The system has to tolerate failures in individual computers.
- The structure of the system (network topology, network latency, number of computers) is not known in advance, the system may consist of different kinds of computers and network links, and the system may change during the execution of a distributed program.
- Each computer has only a limited, incomplete view of the system. Each computer may know only one part of the input.

Distributed systems are groups of networked computers, which have the same goal for their work. The terms "concurrent computing", "parallel computing", and "distributed computing" have a lot of overlap, and no clear distinction exists between them. The same system may be characterised both as "parallel" and "distributed"; the processors in a typical distributed system run concurrently in parallel. Parallel computing may be seen as a particular tightly coupled form of distributed computing, and distributed computing may be seen as a loosely coupled form of parallel computing. Nevertheless, it is possible to roughly classify concurrent systems as "parallel" or "distributed" using the following criteria:

- In parallel computing, all processors may have access to a shared memory to exchange information between processors.
- In distributed computing, each processor has its own private memory (distributed memory). Information is exchanged by passing messages between the processors.



The figure on the right illustrates the difference between distributed and parallel systems. Figure (a) is a schematic view of a typical distributed system; as usual, the system is represented as a network topology in which each node is a computer and each line connecting the nodes is a communication link. Figure (b) shows the same distributed system in more detail: each computer has its own local memory, and information can be exchanged only by passing messages from one node to another by using the available communication links. Figure (c) shows a parallel system in which each processor has a direct access to a shared memory.

The situation is further complicated by the traditional uses of the terms parallel and distributed *algorithm* that do not quite match the above definitions of parallel and distributed *systems*; see the section Theoretical foundations below for more detailed discussion. Nevertheless, as a rule of thumb, high-performance parallel computation in a shared-memory multiprocessor uses parallel algorithms while the coordination of a large-scale distributed system uses distributed algorithms.

II RELATED WORK

Secure deduplication with efficient and reliable convergent key management

Data deduplication is a technique for eliminating duplicate copies of data, and has been widely used in cloud storage to reduce storage space and upload bandwidth. Promising as it is, an arising challenge is to perform secure deduplication in cloud storage. Although convergent encryption has been extensively adopted for secure deduplication, a critical issue of making convergent encryption practical is to efficiently and reliably manage a huge number of convergent keys. This paper makes the first attempt to formally address the problem of achieving efficient and reliable key management in secure deduplication. We first introduce a baseline approach in which each user holds an independent master key for encrypting the convergent keys and outsourcing them to the cloud. However, such a baseline key management scheme generates an enormous number of keys with the increasing number of users and requires users to dedicatedly protect the master keys. To this end, we propose Dekey , a new construction in which users do not need to manage any keys on their own but instead securely distribute the convergent key shares across multiple servers. Security analysis demonstrates that Dekey is secure in terms of the definitions specified in the proposed security model. As a proof of concept, we implement Dekey using the Ramp secret sharing scheme and demonstrate that Dekey incurs limited overhead in realistic environments.

Proofs of ownership in remote storage systems

Cloud storage systems are becoming increasingly popular. A promising technology that keeps their cost down is deduplication, which stores only a single copy of repeating data. Client-side deduplication attempts to identify deduplication opportunities already at the client and save the bandwidth of uploading copies of existing files to the server.

In this work we identify attacks that exploit client-side deduplication, allowing an attacker to gain access to arbitrary-size files of other users based on very small hash signatures of these files. More specifically, an attacker who knows the hash signature of a file can convince the storage service that it owns that file, hence the server lets the attacker download the entire file. (In parallel to our work, a subset of these attacks was recently introduced in the wild with respect to the Dropbox file synchronization service.) To overcome such attacks, we introduce the notion of proofs-of ownership (PoWs), which lets a client efficiently prove to a server that that the client holds a file, rather than just some short information about it. We formalize the concept of proof-of-ownership, under rigorous security definitions, and rigorous efficiency requirements of Petabyte scale storage systems. We then present solutions based on Merkle trees and specific encodings, and analyze their security. We implemented one variant of the scheme. Our performance measurements indicate that the scheme incurs only a small overhead compared to naive client-side deduplication.

III PROPOSED SYSTEM

We propose a secure deduplication scheme for encrypted data that has dynamic owner- ship management capability. The proposed scheme is constructed based partially on a randomized convergent encryption scheme [20] in order to randomize the encrypted data, which renders the proposed scheme secure against the chosen-plaintext attack while still allowing deduplication over the data. The proposed scheme is further integrated into the re-encryption protocol for owner revocation. The owner revocation is executed by re-encrypting the outsourced ciphertext and selectively distributing the re-encryption key to valid (that is, not revoked) owners by the cloud server. The following figure shows the overview of the proposed scheme and its corresponding security goals.

IV ARCHITECTURE & SYSTEM COMPONENTS

The architecture of the data deduplication system, which consists of the following entities.

Data owner: This is a client who owns data, and wishes to upload it into the cloud storage to save costs. A data owner encrypts the data and outsources it to the cloud storage with its index information, that is, a tag. If a data owner uploads data that do not already exist in the cloud storage, he is called an initial uploader; if the data already exist, called a subsequent uploader since this implies that other owners may have uploaded the same data previously, he is called a subsequent uploader. Hereafter, we refer to a set of data owners who share the same data in the cloud storage as an ownership group.

Cloud service provider: This is an entity that provides cloud storage services. It consists of a cloud server and cloud storage. The cloud server deduplicates the outsourced data from users if necessary and stores the deduplicated data in the cloud storage. The cloud server maintains ownership lists for stored data, which are composed of a tag for the stored data and the identities of its owners. The cloud server controls access to the stored data based on the ownership lists and manages (e.g., issues, re-vokes, and updates) group keys for each owner-ship group as a group key authority. The cloud server is assumed to be honest-but-curious. That is, it will honestly execute the assigned tasks in the system; however, it would like to learn as much information about the encrypted contents as possible. Thus, it should be deterred from accessing the plaintext of the encrypted data even if it is honest.

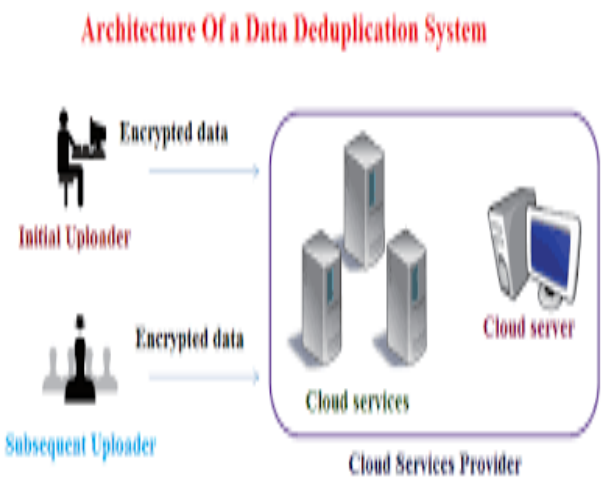


Fig: System Architecture

System Model

In this first module, we develop two entities: User and Secure-Cloud Service Provide.

User: The user is an entity that wants to outsource data storage to the S-CSP and access the data later. In a storage system supporting deduplication, the user only uploads unique data but does not upload any duplicate data to save the upload bandwidth. Furthermore, the fault tolerance is required by users in the system to provide higher reliability.

S-CSP: The S-CSP is an entity that provides the outsourcing data storage service for the users. In the deduplication system, when users own and store the same content, the S-CSP will only store a single copy of these files

and retain only unique data. A deduplication technique, on the other hand, can reduce the storage cost at the server side and save the upload bandwidth at the user side. For fault tolerance and confidentiality of data storage, we consider a quorum of S-CSPs, each being an independent entity. The user data is distributed across multiple S-CSPs.

Data Deduplication:

Data Deduplication involves finding and removing of duplicate datas without considering its fidelity. Here the goal is to store more data with less bandwidth.

- Files are uploaded to the CSP and only the Dataowners can view and download it.
- The Security requirements are also achieved by Secret Sharing Scheme.
- Secret Sharing Scheme uses two algorithms, share and recover.
- Data are uploaded both file and block level and the finding duplication is also in the same process.
- This is made possible by finding duplicate chunks and maintaining a single copy of chunks.

File Level Deduplication Systems:

To support efficient duplicate check, tags for each file will be computed and are sent to S-CSPs.

To upload a file F , the user interacts with S-CSPs to perform the deduplication.

More precisely, the user firstly computes and sends the file tag ϕF = TagGen(F) to S-CSPs for the file duplicate check.

If a duplicate is found the user computes and sends it to a server via a secure channel.

Otherwise if no duplicate is found the process continues, i.e secret sharing scheme runs and the user will upload a file to CSP.

To download a file the user will use the secret shares and download it from the SCSP's . This approach provides fault tolerance and allows the user to remain accessible even if *any* limited subsets of storage servers fail.

In this paper, $x \xleftarrow{\$} S$ denotes the operation of selecting an element x at random and uniformly from a finite set S and assigning it to x. For an algorithm , y ($x1$, . . .) denotes running on inputs $x1$, . . . and assigning the output to the variable y. 1^{λ} denotes a string of λ ones, if $\lambda \in \mathbb{N}$, which is the security parameter4. For two bit-strings a and b, we denote by $a\ b$ their concatenation.

Let = $u1$, , un be the universe of users. Let IDt be the identity of a user u_t. Let tt_i be a set of users that owns the data Mi, which is referred to as an ownership group. Let $L_i = T_i, tt_i$ be an ownership list for Mi, maintained by the cloud server, which consists of a tag Ti and tt_i for Mi. Let KG_i be the ownership group key that is shared among the valid owners in tt_i.

In this section, we define a secure deduplication framework for encrypted data with ownership management capability. The scheme consists of the follow- ing algorithms:

1. $KEK \xleftarrow{\$}$ KEKGen(U): The KEK generation algorithm takes a set of users U as input, and outputs. KEKs for each user in U for secure ownership group key distribution.

2. $C \xleftarrow{\$}$ Encrypt(M, 1λ): The encryption algorithm is a randomized algorithm that takes as input data M and a security parameter λ, and outputs a ciphertext C of the data. C consists of the encrypted message and its tag information for indexing.

3. $C' \xleftarrow{\$}$ ReEncrypt(C, tt): The re-encryption algorithm is a randomized algorithm that takes a ciphertext C and an ownership group tt, and outputs a re-encrypted ciphertext C′. Specifi- cally, it outputs a re-encrypted ciphertext such that only valid owners in tt can decrypt the message.

4. M Decrypt (C′, K, PK): The decryption algorithm is a deterministic algorithm that takes as input C′, message encryption key K, and a set of KEKs PK for encrypting an ownership group key tt_K, and outputs a message M , iff K is derived from M and ttK is not revoked for the ownership group tt (that is, the decryptor is in tt) for M .

Table 1 shows the comparison results of t he se-cure data deduplication schemes that is convergent encryption (CE) [15], leakage -resilient (LR) dedupli-cation [19], and randomized convergent encryption (RCE) [20] in terms of the data deduplication over encrypted data, tag consistency, and dynamic owner-ship management. Since all the schemes allow data owners to en-crypt their data and enable deduplication over them, they can guarantee the data confidentiality or privacy against the cloud server and unauthorized outside adversaries. With regard to data integrity, convergent encryption cannot guarantee the integrity of dedupli-cated data in the face of a poison attack, whereas the other schemes preserve it by adopting an additional mechanism that enables data owners to check the tag consistency of the received data. In the proposed scheme, upon every membership change in the ownership list (e.g., subsequently up-loading the same data, or modifying/deleting the existing data), access to the corresponding data is per-mitted to owners only for the time windows during which the owners maintain valid ownership of the data by re- encrypting it using an updated ownership group key and selectively distributing it. This re-solves the dynamic ownership management problem as opposed to the other schemes. The rekeying in the proposed scheme can be done immediately upon any ownership change. This enhances the security of the outsourced data in terms of backward/forward secrecy by reducing the windows of vulnerability

Table 1: Comparison of Secure deduplication schemes

Scheme	Encrypted deduplication	Tag consistency	Ownership Management
CE [15]	yes	no	No
LR [19]	yes	yes	No
RCE [20]	yes	yes	No
Proposed	yes	yes	Yes

V CONCLUSION

In this paper, we proposed the distributed deduplication systems to improve the reliability of data while achieving the confidentiality of the users' outsourced data without an encryption mechanism. Four constructions were proposed to support file-level and fine-grained block-level data deduplication. The security of tag consistency and integrity were achieved. We implemented our deduplication systems using the Ramp secret sharing scheme and demonstrated that it incurs small encoding/decoding overhead compared to the network transmission overhead in regular upload/download operations.

VI REFERENCES

[1] Amazon, "Case Studies," https://aws.amazon.com/solutions/casestudies/#backup.

[2] J. Gantz and D. Reinsel, "The digital universe in 2020: Big data, bigger digi tal shadows, and biggest growth in the far east," http://www.emc.com/collateral/analyst-eports/idcthe-digital-universe-in-2020.pdf, Dec 2012.

[3] M. O. Rabin, "Fingerprinting by random polynomials," Center for Research in Computing Technology, Harvard University, Tech. Rep. Tech. Report TR-CSE-03-01, 1981.

[4] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system." in *ICDCS*, 2002, pp. 617–624.

[5] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Dupless: Serveraided encryption for deduplicated storage," in *USENIX Security Symposium*, 2013.

[6] ——, "Message-locked encryption and secure deduplication," in *EUROCRYPT*, 2013, pp. 296–312.

[7] G. R. Blakley and C. Meadows, "Security of ramp schemes," in *Advances in Cryptology: Proceedings of CRYPTO '84*, ser. Lecture Notes in Computer Science, G. R. Blakley and D. Chaum, Eds. Springer-Verlag Berlin/Heidelberg, 1985, vol. 196, pp. 242–268.

[8] A. D. Santis and B. Masucci, "Multiple ramp schemes," *IEEE Transactions on Information Theory*, vol. 45, no. 5, pp. 1720–1728, Jul. 1999.

[9] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *Journal of the ACM*, vol. 36, no. 2, pp. 335–348, Apr. 1989.

[10] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[11] J. Li, X. Chen, M. Li, J. Li, P. Lee, and W. Lou, "Secure deduplication with efficient and reliable convergent key management," in *IEEE Transactions on Parallel and Distributed Systems*, 2014, pp. vol. 25(6), pp. 1615–1625.

[12] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems." in *ACM Conference on Computer and Communications Security*, Y. Chen, G. Danezis, and V. Shmatikov, Eds. ACM, 2011, pp. 491–500.

[13] J. S. Plank, S. Simmerman, and C. D. Schuman, "Jerasure: A library in C/C++ facilitating erasure coding for storage applications - Version 1.2," University of Tennessee, Tech. Rep. CS-08-627, August 2008.

[14] J. S. Plank and L. Xu, "Optimizing Cauchy Reed-solomon Codes for fault-tolerant network storage applications," in *NCA-06: 5th IEEE International Symposium on Network Computing Applications*, Cambridge, MA, July 2006.

[15] C. Liu, Y. Gu, L. Sun, B. Yan, and D. Wang, "R-admad: High reliability provision for large-scale de-duplication archival storage systems," in *Proceedings of the 23rd international conference on Supercomputing*, pp. 370–379.

[16] M. Li, C. Qin, P. P. C. Lee, and J. Li, "Convergent dispersal: Toward storage-efficient security in a cloud-of-clouds," in *The 6th USENIX Workshop on Hot Topics in Storage and File Systems*, 2014.

[17] P. Anderson and L. Zhang, "Fast and secure laptop backups with encrypted de-duplication," in *Proc. of USENIX LISA*, 2010.

[18] Z. Wilcox-O'Hearn and B. Warner, "Tahoe: the least-authority filesystem," in *Proc. of ACM StorageSS*, 2008.

[19] A. Rahumed, H. C. H. Chen, Y. Tang, P. P. C. Lee, and J. C. S. Lui, "A secure cloud backup system with assured deletion and version control," in *3rd International Workshop on Security in Cloud Computing*, 2011.

[20] M. W. Storer, K. Greenan, D. D. E. Long, and E. L. Miller, "Secure data deduplication," in *Proc. of StorageSS*, 2008.

[21] J. Stanek, A. Sorniotti, E. Androulaki, and L. Kencl, "A secure data deduplication scheme for cloud storage," in *Technical Report*, 2013.

[22] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage." *IEEE Security & Privacy*, vol. 8, no. 6, pp. 40–47, 2010.

[23] R. D. Pietro and A. Sorniotti, "Boosting efficiency and security in proof of ownership for deduplication." in *ACM Symposium on Information, Computer and Communications Security*, H. Y. Youm and Y. Won, Eds. ACM, 2012, pp. 81–82.

[24] J. Xu, E.-C. Chang, and J. Zhou, "Weak leakage-resilient client-side deduplication of encrypted data in cloud storage," in *ASIACCS*, 2013, pp. 195–206.

[25] W. K. Ng, Y. Wen, and H. Zhu, "Private data deduplication protocols in cloud storage." in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, S. Ossowski and P. Lecca, Eds. ACM, 2012, pp. 441–446.

[26] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proceedings of the 14th ACM conference on Computer and communications security*, ser. CCS '07. New York, NY, USA: ACM, 2007, pp. 598–609. [Online]. Available:http://doi.acm.org/10.1145/1315245.1315318

[27] A. Juels and B. S. Kaliski, Jr., "Pors: proofs of retrievability for large files," in *Proceedings of the 14th ACM conference on Computer and communications security*, ser. CCS '07. New York, NY, USA: ACM, 2007, pp.584–597.[Online]. Available:http://doi.acm.org/10.1145/1315245.1315317

[28] H. Shacham and B. Waters, "Compact proofs of retrievability," in *ASIACRYPT*, 2008, pp. 90–107.

Authors Profile:

Mr. **T. Sandeep** pursuing MCA 3rd year in Qis College and Engineering and Technology in Department of Master of Computer Applications, Ongole.

Mrs. **V. Tejaswini** is currently working as an Assistant Professor in Department of Master of Computer Applications in QIS College of Engineering & Technology.