International Journal On Engineering Technology and Sciences – IJETS™
ISSN(P): 2349-3968, ISSN (O): 2349-3976
Volume I, Issue III, July - 2014

Deduplication Avoidance Using Convergent Key Management in Cloud

P.Gokulraj

P.G. Student, Department of CSE, Nandha College of Technology, Erode.

gokulraj2106@gmail.com

K.Kiruthika Devi

Assistant Professor, Department of CSE, Nandha College of Technology, Erode.

krithime@gmail.com

Abstract— *Data deduplication is a technique for decreasing the amount of storage space for an organization needs to save its data. In many organizations, the storage systems have duplicate copies of many pieces of data and its information. For example, a equivalent file may be saved in several places by many users, or two or more files that aren't equivalent may still include much of the same data. Deduplication remove these extra copies by saving just one copy of the data and changing the other copies with pointers that lead back to the original copy. In this paper first attempt to conventionally address the problem of resolve dynamic and reliable key management in deduplication. In the baseline approach each user holds a separate master key for encrypting the convergent keys and redistributes them to the cloud. And we propose a method Deckey, a new construction in which users are not need to manage any keys on their own. Security analyses establish that Deckey is secure in terms of the exposition specified in the proposed security model. As a proof of concept, we implement Deckey using the Ramp secret sharing scheme and determine that Deckey acquire limited overhead on realistic environments.*

Keywords—Deduplication, deckey, convergent encryption, key management

1. INTRODUCTION

Cloud Storage is a service [1] where data is fully handled, managed, and backed up. It allows the user to store files so that the user can access them from any location. The provider company makes them available to the users online by keeping the upload the files on an external server. This gives companies using cloud storage services ease and convenience, but it can be costly. Users should also be aware that backing up their data is still required when using cloud storage services, because recovering data from cloud is much slower than local backup.

From a user's perspective, data outsourcing raises security and privacy affair. We must trust third-party cloud providers to properly enforce confidentiality, integrity checking, and access control mechanisms against any insider and outsider attacks. However, deduplication, while increasing storage and bandwidth proficiency [3], is incompatible with traditional encryption. Specifically, traditional encryption needs different users to encrypt their data with their own keys. Thus, identical data copies of various users will lead to different cipher texts, making deduplication impossible.

Convergent encrypt allowed one cloud storage provider to claim 'many storage' in addition to 'security'. Convergent encryption is an interesting concern between efficiency and privacy. Convergent encryptions [8] include a viable option to enforce data confidentiality reliability while realizing deduplication. It encrypts and decrypts a data copy with a convergent key, which is derived by the cryptographic hash value of the content of the data copy itself. After key generation and data encryption, users maintain the keys and send the ciphertext to the cloud service. Since encryption is deterministic, equivalent data copies will generate the same convergent key and the same ciphertext.

To understand how convergent encryption can be accomplished, we consider a baseline approach that implements convergent encryption based on a layered approach. That

is, the real data copy is first encrypted with a convergent key which are derived by the data copy, and the convergent key is then encrypted by a master key that will be kept locally and securely by each user. The encrypted convergent keys are stored, along with the parallel encrypted data copies, in cloud storage services. The master key can be used to recover the encrypted keys and hence the encrypted files. In this way, each user only needs to store the master key.

This paper makes the following contributions.

- A new construction Deckey is proposed to provide efficient and reliable convergent key management through convergent key deduplication and secret sharing. Deckey supports both file-level and block-level deduplication.

- Security analysis demonstrates that Deckey is secure in terms of the definitions specified in the proposed security model. In particular, Deckey remains secure even the adversary controls a limited number of key servers.
- We implement Deckey using the Ramp secret sharing scheme that enables the key management to adapt to different reliability and confidentiality levels. Our evaluation demonstrates that Deckey incurs limited overhead in normal upload/download operations in realistic cloud environments.

2. PRELIMINARIES

In this section, we formally define the Cryptographic primitives used in our secure deduplication.

2.1 SYMMETRIC ENCRYPTION

Symmetric encryption uses a common secret key to encrypt and decrypt information. A symmetric encryption scheme consists of three primitive functions:

- $\text{KeyGen}_{SE}(1^{\lambda}) \rightarrow k$ is the key generation algorithm that generates k using Security parameter 1^{λ}.

- $\text{Encrypt}_{SE}(k,M) \rightarrow C$ is the symmetric encryption algorithm that takes the secret k and message and then outputs the ciphertext C.

- $\text{Decrypt}_{SE}(k,M) \rightarrow M$ is the symmetric decryption algorithm that takes the secret k and ciphertext and then outputs the original message M.

2.2 CONVERGENT ENCRYPTION

Convergent encryption [8] [2] provides data confidentiality in deduplication. A user (or data owner) derives a convergent key from each original data copy and encrypts the data copy with the convergent key. The user extracts a tag for the data copy, such that the tag will be used to find duplicates. We assume that the tag property [5] holds, i.e., if two data copies are the same, then their tags are also the same. To find duplicates, the user first sends the tag to the server side to check if the identical copy has been stored already. Note that both the convergent key and the tag are independently derived and the tag cannot be used to deduce the convergent key and compromise data confidentiality. The both encrypted data copy and its corresponding tag will be stored on the server side.

2.3 RAMP SECRET SHARING

Deckey uses the Ramp secret sharing scheme (RSSS) [4] [9] to store the convergent keys. Especially, the (n,k,r)-RSSS (where $n>k>r\geq 0$) generates n shares from a secret sharing such that 1) the secret can be recovered from any k shares but cannot be replaces from fewer than k shares, and 2) There is no information about the secret can be consider from any r shares. It is known that when $r=0$, the $(n,k,0)$-RSSS becomes the $(n,k,)$ Rabin's Information Dispersal Algorithm (IDA) [5]; when $r=k-1$, the $(n,k,k-1)$-RSSS becomes the $(n,k,)$Shamir's Secret Sharing Scheme (SSSS). The (n,k,r)-RSSS builds on two primitive functions:

- Share divides a secret S into ,k-,r)-pieces of equal size, generates r random pieces of the same size, and encodes the k pieces using a non-systematic k-of-n erasure code into n shares of the same size;

- Recover takes any k out of n shares as inputs and then outputs the original secret S.

To make the generated shares appropriate for deduplication, we replace the above random pieces with pseudorandom pieces in the implementation of Deckey. Deckey uses RSSS to provide a tunable key management mechanism to balance among confidentiality, reliability, storage overhead, and performance.

3. PROBLEM FORMULATION

3.1 SYSTEM MODEL

We first develop a data outsourcing model used by Deckey. There are three entities, namely: the user, the cloud service provider Storage (CSP-S), and the cloud service provider key-management (CSP-KM), as categorized below.

- **User:** A user is an entity that needs to outsource data storage to the S-CSP and access the data later. To store the upload bandwidth, the user only uploads different data but does not upload any duplicate data, which may be retained by the same user or different users.

- **CSP-S:** The CSP-S provides the data outsourcing service and stores data on behalf of the users. To reduce the cost of storage, the CSP-S removes the storage of redundant data via deduplication and keeps only unique data.

CSP-KM: A CSP-KM maintains convergent keys for users, and provides users with small storage and computation services to facilitate key management. In the fault tolerance of key management, we consider a quorum of CSP-KMs, each being a separate entity. Each convergent key is distributed across multiple CSPs-KM using RSSS (see Section 2).

In this, we specify a data copy to be either a whole file or a smaller-size block, and this point to two types of deduplication: 1) file-level deduplication, which remove the storage of any repeated files, and 2) block-level deduplication, which fragment a file into smaller fixed-size or variable-size blocks and delete the storage of any redundant blocks. We deploy our deduplication mechanism in both file level and block level. Specifically, to upload a file, a user first executes the file-level duplicate check. If the file finds to be a duplicate, then all its blocks must be duplicates; otherwise, the user performs the block-level duplicate to check and find the unique blocks to be uploaded. Each of the data copy is associated with a tag for the duplicate check. All data copies and tags will be stored in the CSP-S.

4. CONSTRUCTIONS

In this section, we represent a baseline approach that realizes convergent encryption in deduplication, and examine the limitations of the baseline approach in key management. To this end, we denote our construction Deckey, which aims to mitigate the key management overhead and provide fault tolerance guarantees for key management, while protecting the required security properties of secure deduplication.

4.1 SYSTEM SETUP

The system setup phase computes the necessary parameters in the following two steps:

S1: The following points are initialized: 1) a symmetric encryption scheme with the primitive functions ($KeyGen_{SE}$, $Encrypt_{SE}$, $Decrypt_{SE}$) and the user's master key k= $KeyGen_{SE}$ (1^{λ}) for scurity parameter 1^{λ}; 2) a convergent encryption scheme with the primitive functions ($KeyGen_{CE}$, $Encrypt_{CE}$, $Decrypt_{CE}$, $TagGen_{CE}$).

S2: The CSP-S initializes two types of storage systems: a rapid storage system for store the tags for efficient duplicate checks, and a file storage system for storing both encrypted data copies and convergent keys.

4.2 FILE UPLOAD

Suppose that a user uploads a file F . First, it performs file-level deduplication as follows.

S1: On input file F, the user execute and sends the file tag T(F)= $TagGen_{CE}$(F) to the CSP-S.

S2: Upon receiving T(F), the CSP-S checks whether there exists the same tag on the CSP-S. If so, the CSP-S replies the user with a response ‘‘file duplicate,’’ or ‘‘no file duplicate’’ otherwise.

S3: If the user receives the response ‘‘no file duplicate’’, then it jumps to S5 to proceed with block-level deduplication. If the response is ‘‘file duplicate,’’ then the user runs PoW_F on F with the CSP-S to prove that it actually owns the same file F that is stored on the S-CSP.

S4: If PoW_F is passed, the CSP-S simply returns a file pointer of F to the user, and no further information will be uploaded. If PoW_F fails, the S-CSP aborts the upload operation.

4.2 DECKEY

Deckey is designed to efficiently and reliably maintain convergent keys. Its idea is to implement deduplication in convergent keys and distribute the convergent keys across different CSPs-KM. Instead of encrypting the convergent keys on a per-user basis, Deckey constructs secret shares on the original convergent keys (that are in plain) and distributes the shares across multiple CSPs-KM. If multiple users share the same block, they can access the same convergent key. This significantly decreases the storage overhead for convergent keys. In addition, this approach gives fault tolerance and allows the convergent keys to remain accessible even if any subset of CSPs-KM fails. We now elaborate the details of Deckey as follows.

4.2.1 System Setup

The system setup phase in Deckey is similar to that in the baseline approach, but involves an additional step for initializing the key storage in CSPs-KM. In Deckey, we assume that the number of CSPs-KM is n.

S1: On input security parameter 1^{λ}, the user initializes a convergent encryption scheme, and two PoW protocols POW_F and POW_B for the file ownership proof and block ownership proof, respectively.

S2: The CSP-S initializes both the rapid storage system and the file storage system.

S3: Each CSPs-KM initializes a rapid storage system for block tags and a lightweight storage system for holding convergent key shares.

5. IMPLEMENTATION

In this section, we discuss the implementation details of Deckey. Deckey builds on the Ramp secret sharing scheme (RSSS) [3], [4] to distribute the shares of convergent keys across multiple key servers (see Section 2)

5.1 RSSS with Pseudorandomness

In Deckey, the RSSS secret is the hash key H_0 of a data block B, where H_0=hash(B). Recall from Section 2 that the Share function of the (n,k,r)-RSSS embeds r random pieces to achieve a confidentiality level of r. One challenge is that randomization conflicts with deduplication, since the random pieces cannot be deduplicated with each other. Instead of directly adopting RSSS, we here replace these random pieces with pseudorandom pieces in our Deckey implementation.

We generate the r pseudorandom pieces as follows. Let m = $[\frac{r}{[k-r]}]$. We first generate m additional hash values as H_1=hash(B+1), H_2=hash(B+2), . . . , H_m=hash(B+m). We then fill in the r pieces with the generated m additional hash values H_1. H_2,. . . ; H_m.

5.2 EVALUATION

In this section, we evaluate the encoding and decoding performance of Deckey on generating and recovering key shares, respectively. All our experiments were performed on an Intel Xeon E5530 (2.40 GHz) server with Linux 3.2.0-23-generic OS.

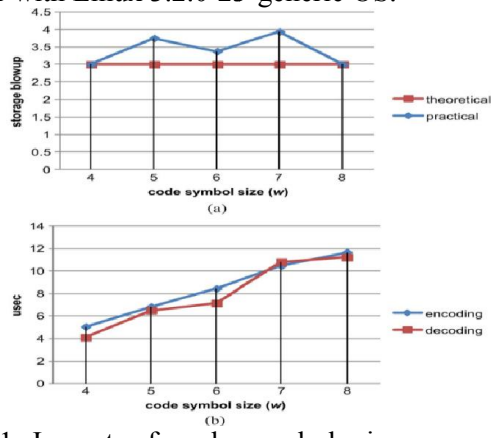

Fig. 1 Impact of code symbol size w on storage blowup and performance in the case of (6, 4, 2)-RSSS. (a) Storage blowup. (b) Encoding/decoding time

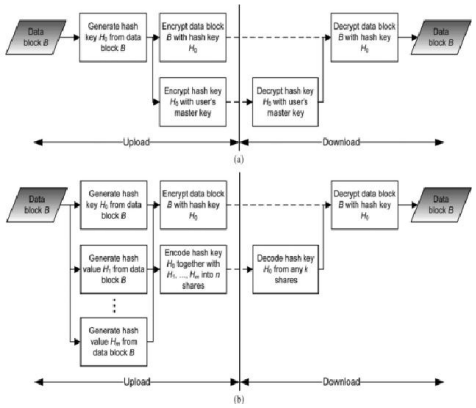

Fig. 2 Flow block diagrams of core modules in two different approaches. (a) Baseline approach (keeping the hash key with an encryption scheme). (b) Deckey (keeping the hash key with (n,k,r)-RSSS)

5.3 OVERALL RESULTS

With (n,k,r)-RSSS being used, we test all the following cases:$3 \le n \le 8, 2 \le k \le n-1$,and $1 \le r \le k-1$. We can see that the encoding and decoding times of Deckey for each hash key (per 4 KB data block) are always on the order of microseconds, and hence are negligible compare to the data transfer performance in the Internet .

6. RELATED WORK

6.1 CONVERGENT ENCRYPTION

Convergent encryption [8] ensures data privacy in deduplication. Bellare et al. [2] formalize this primitive as a message-locked encryption, and analyze its application in space-efficient secure outsourced storage. There are also several implementations of different convergent encryption variants for secure deduplication. It is known that some commercial cloud storage providers, such as Bitcasa, also deploy convergent encryption [2]

7. CONCLUSION

We propose Deckey is an efficient and reliable convergent key management scheme for secure deduplication. Deckey assign deduplication among convergent keys and distributes convergent key shares across many key servers, while protecting semantic security of convergent keys and confidentiality of outsourced data. We implement Deckey using the Ramp secret sharing scheme and implement that it incurs small encoding/decoding overhead com-pared to the network transmission overhead in the regular upload/download operations.

REFERENCES

[1] Ed. Michael Waidner"On the Security of Cloud Storage Services" SIT Technical Reports, SIT-TR-2012-001 March 2012, Fraunhofer Institute for Secure Information Technology SIT.

[2] M. Bellare, S. Keelveedhi, and T. Ristenpart, ''Message-Locked Encryption and Secure Deduplication,'' in Proc. IACR Cryptology ePrint Archive, 2012, pp. 296-3122012:631.

[3] Nesrine Kaaniche, Maryline Laurent " A Secure Client Side Deduplication Scheme Cloud Storage Environments" 6TH International Conference On New Technologies, And Security Year 2014

[4] A.D. Santis and B. Masucci, ''Multiple Ramp Schemes,'' IEEE Trans. Inf. Theory, vol. 45, no. 5, pp. 1720-1728, July 1999.

[5]M.O. Rabin, ''Efficient Dispersal of Information for Security, Load Balancing, Fault Tolerance,'' J. ACM, vol. 36, no. 2, pp. 335-348, Apr. 1989.

[6]S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, ''Proofs of Ownership in Remote Storage Systems,'' in Proc. ACM Conf. Comput. Commun. Security, Y. Chen, G. Danezis, and V. Shmatikov, Eds., 2011, pp. 491-500.

[7]R.D. Pietro and A. Sorniotti, ''Boosting Efficiency and Security in Proof of Ownership for Deduplication,'' in Proc. ACM Symp. Inf., Comput. Commun. Security, H.Y. Youm and Y. Won, Eds., 2012, pp.81-82.

[8]J.R. Douceur, A. Adya, W.J. Bolosky, D. Simon, and M. Theimer, ''Reclaiming Space from Duplicate Files in a Serverless Distrib-uted File System,'' in Proc. ICDCS, 2002, pp. 617-624.

[9]G.R. Blakley and C. Meadows, ''Security of Ramp Schemes,'' in Proc. Adv. CRYPTO, vol. 196, Lecture Notes in Computer Science, G.R. Blakley and D. Chaum, Eds., 1985, pp. 242-268.

[10]"Data Deduplication in Cloud Exxplained" http://www.computerworld.com/article/2474479/data-center/data-deduplication-in-the-cloud.