An Efficient Scheme for Secured Deduplication Using Dekey for Reliable Convergent Key Management

Prabha Devi. D[1], K. Sivachandran [2]
[1] PG Scholar, [2] Assistant Professor, Computer Science and Engineering Department,
Sasurie Academy of Engineering,

Abstract: **One of the most important data compression techniques for eliminating duplicate copies of repeating data is the Data Deduplication, and it is widely used in cloud storage to reduce the amount of storage space and also to save bandwidth. To protect the confidentiality of sensitive data along with deduplication, the convergent encryption technique has been proposed to encrypt the data before outsourcing. Convergent encryption has been proposed to enforce data confidentiality while making deduplication feasible. It encrypts/decrypts a data copy with a convergent key, which is obtained by computing the cryptographic hash value of the content of the data copy. After key generation and data encryption, users retain the keys and send the cipher text to the cloud. Since the encryption operation is deterministic and is derived from the data content, identical data copies will generate the same convergent key and hence the same cipher text. To prevent unauthorized access, a secure proof of ownership protocol is also needed to provide the proof that the user indeed owns the same file when a duplicate is found.**

1. INTRODUCTION

1.1 CLOUD COMPUTING

The advent of cloud storage motivates enterprises and organizations to outsource data storage to third-party cloud providers, as evidenced by many real-life case studies. One critical challenge of today's cloud storage services is the management of the ever-increasing volume of data. To make data management scalable, deduplication has been a well-known technique to reduce storage space and upload bandwidth in cloud storage. Instead of keeping multiple data copies with the same content, deduplication eliminates redundant data by keeping only one physical copy and referring other redundant data to that copy. Each such copy can be defined based on different granularities: it may refer to either a whole file (i.e., file level deduplication), or a more fine-grained fixed-size or variable-size data block (i.e., block-level deduplication).

Today's commercial cloud storage services, such as Dropbox, Mozy, and Memopal, have been applying deduplication to user data to save maintenance cost. From a user's perspective, data outsourcing raises security and privacy concerns. 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 improving storage and bandwidth efficiency, is incompatible with traditional encryption. Specifically, traditional encryption requires different users to encrypt their data with their own keys. Thus, identical data copies of different users will lead to different cipher texts, making deduplication impossible.

Convergent encryption provides a viable option to enforce data confidentiality while realizing deduplication. It encrypts/decrypts a data copy with a convergent key, which is derived by computing the cryptographic hash value of the content of the data copy itself. After key generation and data encryption, users retain the keys and send the cipher text to the cloud. Since encryption is deterministic, identical data copies will generate the same convergent key and the same cipher text. This allows the cloud to perform deduplication on the cipher texts. The cipher texts can only be decrypted by the corresponding data owners with their convergent keys. To understand how convergent encryption can be realized, I considered a baseline approach that implements convergent encryption based on a layered approach. That is, the original data copy is first encrypted with a convergent key derived by the data copy itself, 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 then stored, along with the corresponding encrypted data copies, in cloud storage. The master key can be used to recover the encrypted keys and hence the encrypted files. In this way, each user only needs to keep the master key and the metadata about the outsourced data.

1.2 OBECTIVE

Data management through cloud is being viewed as a technique that can save the cost for data sharing and management. A key concept for remote data storage is client-side deduplication, in which the server stores only a single copy of each file, regardless of how many clients need to store that file. That is only the first client needs to upload the file to the server. This design will save both the communication bandwidth as well as the storage capacity. 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.

2. PRELIMINARIES:

Cryptographic primitives used in this secured deduplication is described as follows:

2.1 Symmetric Encryption:

Symmetric encryption uses the same key to both encrypt and decrypt tha data. A symmetric encryption consists of three primitive functions namely:

(i) key Gen(k)
(ii) Encrypt(k,M)
(iii) Decrypt(k,C)

2.2 Convergent Encryption:

Convergent encryption results with data confidentiality in deduplication. From the given original text, the user derives the convergent key and as well as tag, this tag is used to detect the data duplicates which are described with the tag correctness property ie. If two data copies are similar, then their tag will be the same user first sends the tag to the server side to check if the identical copies has been stored already in the server side to detect the duplicate copies of data. Formally, a convergent encryption scheme can be
defined with four primitive functions:

(i) keyGen(k)
(ii) encrypt(k,M)
(iii) decrypt(k,C)
(iv) TagGen(M)

2.3 Proof of Ownership

PoW is used as a proxy for the entire file in client side deduplication which provides more security at the client side. By this one can prove that the user /client has the same file to the server with the support of Dekey techniqueIn more general, this PoW concept is used as an interactive algorithm that is to be run by both prover and a verifier.

2.4 Ramp Secret Sharing

To store the convergent keys,Dekey uses the concept of Ramp secret sharing scheme (RSSS) . Specifically, the (n,k,r)-RSSS (where n > k > r > 0) generates n shares from a secret such that 1) the secret can be recovered from any k shares but cannot be recovered from fewer than k shares, and 2) no information about the secret can be deduced from any r shares. The (n,k,r)-RSSS builds on two primitive functions:

(i)**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 code1 into n shares of the same size.
(ii)**Recover** takes any k out of n shares as inputs and then outputs the original secret S.

3. SYSTEM MODEL:

3.1.1 Cloud System Set Up

The cloud server provides data storage and sharing services to data owners and data users. After verify the member connection under signature, member can able to access the particular owner's data with respect to owner's private key and identity (IDdata).So the cloud verifies whether the request member is in the revoke list which is send by group manager under signature if so, it provide permission to access the data else throw unauthorized member request. So the revoke list is updated once member leave or join the group by cloud.

3.1.2 User Registration, File Upload And Download

An entity, which has large data files to be stored in the cloud and relies on the cloud for data maintenance and computation, can be either individual consumers or organizations. User first chose a random parameter to construct the public and the private keys then he\she will sign the data using the private key to be uploaded to the cloud, then he/she send the signed data to the cloud server and deletes its local copy.

3.1.3 Convergent Encryption

Convergent encrypt allowed one cloud storage provider to claim 'infinite storage' in addition to 'security'. For another storage provider it resulted in controversy when some of the weaknesses became apparent. Convergent encryption is an interesting trade-off between efficiency and privacy. At the epicenter is an interesting question: Can a file provide the entropy for its own key? In this article I will explain the situation both practically and technically. Take the cryptographic primitives

HAHBED::::{0,1}∗{0,1}∗{0,1}lK×{0,1}∗{0,1}lK×{0,1}∗ →{0,1}lA→{0,1}lB→{0,1}∗→{0,1}∗,where HA and HB are cryptographic hash functions of length lA and lB respectively
and E and D are symmetric encryption and decryption functions with key length lK. This implies, HA(X1)HB(X2)D(K1,E(K2,X))=HA(X2)=HB(X2)=X⇔⇔ ⇔X1X1K1=X2=X2=K2
Where, the implications in the leftward direction are exact but in the rightward implications are only with cryptographic confidence. This will be important later in the security analysis.

For permanent storage, we will use an associative array with interface
StoreRetrieve::{0,1}lB×{0,1}∗{0,1}lB→∅→{0,1}∗
The convergent encryption store will have the interface as,
PutGet::{0,1}∗{0,1}lB×{0,1}lA→{0,1}lB×{0,1}lA→{0,1}∗
The Put operation takes a piece of information and stores it encrypted in the associative array. It is implemented as
Put(X)KX′H=:(H,K)=HA(X)=E(K,X)=HB(X′)Store(H,X′).
The Get operation is simply the inverse operation
Get(H,K)X′X=:X=Retrieve(H)=D(K,X′)
The Get operation can optionally verify the integrity of the data by checking K=HA(X) and/or H=HA(X′). The later has an advantage that will be shown below.

The implementation given above requires three passes through the data. The passes have data

dependencies so they cannot be parallelized. It is possible to develop a two-pass system. The hash H of the output stream X′ can be calculated while the output stream is being produced. This is similar to authenticated encryption but differs in the use of a hash instead of a message authentication code. But it is not possible to construct a one-pass system that deduplicates and has at least the security of convergent encryption. In a one-pass system, the first couple of bytes cannot rely on all the remaining bytes. But this is necessary to have a key with the entropy of the entire file.

3.1.4 Dekey

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

Thus, the overall system architecture is described below:

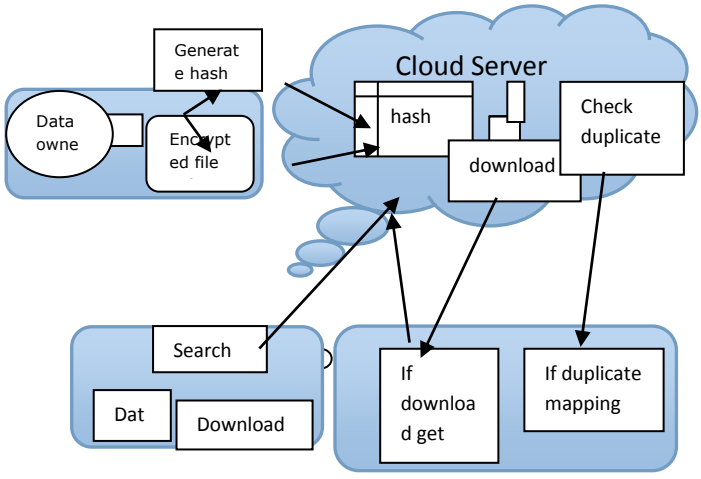

Fig: System Architecture of Secured deduplication using Dekey concept

4. CONCLUSION & FUTURE ENHANCEMENTS

4.1 CONCLUSION

This project proposes Dekey, an efficient and reliable convergent key management scheme for secure deduplication. Dekey applies deduplication among convergent keys and distributes convergent key shares across multiple key servers, while preserving semantic security of convergent keys and confidentiality of outsourced data. Dekey using the Ramp secret sharing scheme and demonstrate that it incurs small encoding/decoding overhead compared to the network transmission overhead in the regular upload/download operations is implemented.

4.2 FUTURE ENHANCEMENT

Convergent encryption enables identical encrypted files to be recognized as identical, but there remains the problem of performing this identification across a large number of machines in a robust and decentralized manner. This problem is solved by storing file location and content information in a distributed data structure called a SALAD: a Self-Arranging, Lossy, Associative Database. For scalability, the file information is partitioned and dispersed among all machines in the system; and for fault-tolerance, each item of information is stored redundantly on multiple machines. Rather than using central coordination to orchestrate this partitioning, dispersal, and redundancy, SALAD employs simple statistical techniques, which have the unintended effect of making the database lossy.

REFERENCES:

[1] P. Anderson and L. Zhang, ‘‘Fast and Secure Laptop Backups with Encrypted De-Duplication,’’ in Proc. USENIX LISA, 2010.
[2] M. Bellare, S. Keelveedhi, and T. Ristenpart, ‘‘Message-Locked Encryption and Secure Deduplication,’’ in Proc. IACR Cryptology.
[3] Jin Li, Xiaofeng Chen, Mingqiang Li, Jingwei Li, Patrick P.C. Lee, and Wenjing Lou, “Secure Deduplication with Efficient and Reliable Convergent Key Management, June 2014.
[4] W. Wang, Z. Li, R. Owens, and B. Bhargava, ‘‘Secure and Efficient Access to Outsourced Data,’’ in Proc. ACM CCSW, Nov. 2009
[5] A. Yun, C. Shi, and Y. Kim, ‘‘On Protecting Integrity and Confidentiality Of Cryptographic File System for Outsourced Storage,’’ in Proc. ACM CCSW, Nov. 2009.
[6] D. Meister and A. Brinkmann, ‘‘Multi-Level Comparison of Data Deduplication in a Backup Scenario,’’ in Proc. SYSTOR, 2009.
[7] R.D. Pietro and A. Sorniotti, ‘‘Boosting Efficiency and Security in Proof of Ownership for Deduplication,’’ in Proc. ACMSymp. Inf., Comput. Commun. Security, H.Y. Youm and Y. Won, Eds., 2012.
[8] G.R.Blakley and C. Meadows, ‘‘Security of Ramp Schemes’’.