Global Journal of Engineering Science and Research Management

REDUCTION IN STORAGE SPACE BY ELIMINATION OF REDUNDANT DATA

Sandhya .N .C*, Shashirekha .H
* Dept. of Computer science &Engg, VTU PG Centre, Mysore, Karnataka, INDIA.
Assistent Professor, Dept. of Computer science &Engg, VTU PG Centre, Mysore, Karnataka, INDIA.

KEYWORDS: Deduplication, distributed storage system, reliability, secret sharing.

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.

INTRODUCTION
With the explosive growth of digital data, deduplication techniques are widely employed to backup data and minimize network and storage overhead by detecting and eliminating redundancy among data. 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. Deduplication has received much attention from both academia and industry because it can greatly improves storage utilization and save storage space, especially for the applications with high deduplication ratio such as archival storage systems.

A number of deduplication systems have been proposed based on various deduplication strategies such as client-side or server-side deduplications, file-level or block-level deduplications. A brief review is given in Section 6. Especially, with the advent of cloud storage, data deduplication techniques become more attractive and critical for the management of ever-increasing volumes of data in cloud storage services which motivates enterprises and organizations to outsource data storage to third-party cloud providers, as evidenced by many real-life case studies [1]. According to the analysis report of IDC, the volume of data in the world is expected to reach 40 trillion gigabytes in 2020 [2]. Today's commercial cloud storage services, such as Dropbox, Google Drive and Mozy, have been applying deduplication to save the network bandwidth and the storage cost with client-side deduplication.

There are two types of deduplication in terms of the size: (i) file-level deduplication, which discovers redundancies between different files and removes these redundancies to reduce capacity demands, and (ii) blockleveldeduplication, which discovers and removes redundancies between data blocks. The file can be divided into smaller fixed-size or variable-size blocks. Using fixedsize blocks simplifies the computations of block boundaries, while using variable-size blocks (e.g., based on Rabin fingerprinting [3]) provides better deduplication Efficiency.

Though deduplication technique can save the storage space for the cloud storage service providers, it reduces the reliability of the system. Data reliability is actually a very critical issue in a deduplication storage system because there is only one copy for each file stored in the server shared by all the owners. If such a shared file/chunk was lost, a disproportionately large amount of data becomes inaccessible because of the unavailability of all the files that share this file/chunk. If the value of a chunk were measured in terms of the amount of file data that would be lost in case of losing a single chunk, then the amount of user data lost when a chunk in the storage system is corrupted grows with the number of the commonality of the chunk. Thus, how to guarantee high data reliability in deduplication system is a critical problem. Most of the previous deduplication systems have only been considered in a single-server setting. However, as lots of deduplication systems and cloud storage systems are intended by users and applications for higher reliability, especially in archival storage systems where data are

critical and should be preserved over long time periods. This requires that the deduplication storage systems provide reliability comparable to other high-available systems.

Furthermore, the challenge for data privacy also arises as more and more sensitive data are being outsourced by users to cloud. Encryption mechanisms have usually been utilized to protect the confidentiality before outsourcing data into cloud. Most commercial storage service provider are reluctant to apply encryption over the data because it makes deduplication impossible. The reason is that the traditional encryption mechanisms, including public key encryption and symmetric key encryption, require different users to encrypt their data with their own keys. As a result, identical data copies of different users will lead to different ciphertexts. To solve the problems of confidentiality and deduplication, the notion of convergent encryption [4] has been proposed and widely adopted to enforce data confidentiality while realizing deduplication. However, these systems achieved confidentiality of outsourced data at the cost of decreased error resilience. Therefore, how to protect oth confidentiality and reliability while achieving deduplication in a cloud storage system is still a challenge.

1.1 Our Contributions

In this paper, we show how to design secure deduplication systems with higher reliability in cloud computing. We introduce the distributed cloud storage servers into deduplication systems to provide better fault tolerance. To further protect data confidentiality, the secret sharing technique is utilized, which is also compatible with the distributed storage systems. In more details, a file is first split and encoded into fragments by using the technique of secret sharing, instead of encryption mechanisms. These shares will be distributed across multiple independent storage servers. Furthermore, to support deduplication, a short cryptographic hash value of the content will also be computed and sent to each storage server as the fingerprint of the fragment stored at each server. Only the data owner who first uploads the data is required to compute and distribute such secret shares, while all following users who own the same data copy do not need to compute and store these shares any more. To recover data copies, users must access a minimum number of storage servers through authentication and obtain the secret shares to reconstruct the data. In other words, the secret shares of data will only be accessible by the authorized users who own the corresponding data copy.

Another distinguishing feature of our proposal is that data integrity, including tag consistency, can be achieved. The traditional deduplication methods cannot be directly extended and applied in distributed and multi-server systems. To explain further, if the same short value is stored at a different cloud storage server to support a duplicate check by using a traditional deduplication method, it cannot resist the collusion attack launched by multiple servers. In other words, any of the servers can obtain shares of the data stored at the other servers with the same short value as proof of ownership. Furthermore, the tag consistency, which was first formalized by [5] to prevent the duplicate/ciphertext replacement attack, is considered in our protocol. In more details, it prevents a user from uploading a maliciously-generated ciphertext such that its tag is the same with another honestly-generated ciphertext. To achieve this, a deterministic secret sharing method has been formalized and utilized. To our knowledge, no existing work on secure deduplication can properly address the reliability and tag consistency problem in distributed storage systems.

This paper makes the following contributions.

- Four new secure deduplication systems are proposed to provide efficient deduplication with high reliability for file-level and block-level deduplication, respectively. The secret splitting technique, instead of traditional encryption methods, is utilized to protect data confidentiality. Specifically, data are split into fragments by using secure secret sharing schemes and stored at different servers. Our proposed constructions support both file-level and block-level deduplications.
- Security analysis demonstrates that the proposed deduplication systems are secure in terms of the definitions specified in the proposed security model. In more details, confidentiality, reliability and integrity can be achieved in our proposed system. Two kinds of collusion attacks are considered in our solutions. These are the collusion attack on the data and the collusion attack against servers. In particular, the data remains secure even if the adversary controls a limited number of storage servers.
- We implement our deduplication systems using the Ramp secret sharing scheme that enables high reliability and confidentiality levels. Our evaluation results demonstrate that the new proposed constructions are efficient and the redundancies are optimized and comparable with the other storage system supporting the same level of reliability.

1.2 Organization
This paper is organized as follows. In Section 2, we present the system model and security requirements of deduplication. Our constructions are presented in Section 3 and Section 4. The security analysis is given in Section 5. The implementation and evaluation are shown in Sections 6, and related work is described in Section 7. Finally, we draw our conclusions in Section 8.

PROBLEM FORMULATION

2.1 System Model
This section is devoted to the definitions of the system model and security threats. Two kinds entities will be involved in this deduplication system, including the user and the storage cloud service provider (S-CSP). Both client-side deduplication and server-side deduplication are supported in our system to save the bandwidth for data uploading and storage space for data storing.

- 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.

We deploy our deduplication mechanism in both file and block levels. Specifically, to upload a file, a user first performs the file-level duplicate check. If the file is a duplicate, then all its blocks must be duplicates as well, otherwise, the user further performs the block level duplicate check and identifies the unique blocks to be uploaded. Each data copy (i.e., a file or a block) is associated with a tag for the duplicate check. All data copies and tags will be stored in the S-CSP.

2.2 Threat Model and Security Goals
Two types of attackers are considered in our threat model: (i) An outside attacker, who may obtain some knowledge of the data copy of interest via public channels. An outside attacker plays the role of a user that interacts with the S-CSP; (ii) An inside attacker, who may have some knowledge of partial data information such as the ciphertext. An insider attacker is assumed to be honest-but-curious and will follow our protocol, which could refer to the S-CSPs in our system. Their goal is to extract useful information from user data. The following security requirements, including confidentiality, integrity, and reliability are considered in our security model.

Confidentiality. Here, we allow collusion among the SCSPs. However, we require that the number of colluded S-CSPs is not more than a predefined threshold. To this end, we aim to achieve data confidentiality against collusion attacks. We require that the data distributed and stored among the S-CSPs remains secure when they are unpredictable (i.e., have high min-entropy), even if the adversary controls a predefined number of S-CSPs. The goal of the adversary is to retrieve and recover the files that do not belong to them. This requirement has recently been formalized in [6] and called the privacy against chosen distribution attack. This also implies that the data is secure against the adversary who does not own the data.

Integrity. Two kinds of integrity, including tag consistency and message authentication, are involved in the security model. Tag consistency check is run by the cloud storage server during the file uploading phase, which is used to prevent the duplicate/ciphertext replacement attack. If any adversary uploads a maliciously-generated ciphertext such that its tag is the same with another honestly-generated ciphertext, the cloud storage server can detect this dishonest behavior. Thus, the users do not need to worry about that their data are replaced and unable to be decrypted. Message authentication check is run by the users, which is used to detect if the downloaded and decrypted data are complete and uncorrupted or not. This security requirement is introduced to prevent the insider attack from the cloud storage service providers.

Reliability. The security requirement of reliability in deduplication means that the storage system can provide fault tolerance by using the means of redundancy. In more details, in our system, it can be tolerated even if a

certain number of nodes fail. The system is required to detect and repair corrupted data and provide correct output for the users.

THE DISTRIBUTED DEDUPLICATION SYSTEMS

The distributed deduplication systems' proposed aim is to reliably store data in the cloud while achieving confidentiality and integrity. Its main goal is to enable deduplication and distributed storage of the data across multiple storage servers. Instead of encrypting the data to keep the confidentiality of the data, our new constructions utilize the secret splitting technique to split data into shards. These shards will then be distributed across multiple storage servers.

3.1 Building Blocks

Secret Sharing Scheme. There are two algorithms in a secret sharing scheme, which are Share and Recover. The secret is divided and shared by using Share. With enough shares, the secret can be extracted and recovered with the algorithm of Recover. In our implementation, we will use the Ramp secret sharing scheme (RSSS, to secretly split a secret into shards. Specifically, the (n, k, r)-RSSS (where $n > k > r \geq 0$) generates n shares from a secret so that (i) the secret can be recovered from any k or more shares, and (ii) no information about the secret can be deduced from any r or less shares. Two algorithms, Share and Recover, are defined in the (n, k, r)-RSSS.

- 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. It is known that when $r = 0$, the $(n, k, 0)$-RSSS becomes the (n, k) Rabin's Information Dispersal Algorithm (IDA) [9]. When $r = k-1$, the $(n, k, k-1)$-RSSS becomes the (n,k) Shamir's Secret Sharing Scheme (SSSS) [10].

Tag Generation Algorithm. In our constructions below, two kinds of tag generation algorithms are defined, that is, TagGen and TagGen'. TagGen is the tag generation algorithm that maps the original data copy F and outputs a tag $T(F)$. This tag will be generated by the user and applied to perform the duplicate check with the server. Another tag generation algorithm TagGen' takes as input a file F and an index j and outputs a tag. This tag, generated by users, is used for the proof of ownership for F.

Message authentication code. A message authentication code (MAC) is a short piece of information used to authenticate a message and to provide integrity and authenticity assurances on the message. In our construction, the message authentication code is applied to achieve the integrity of the outsourced stored files. It can be easily constructed with a keyed (cryptographic) hash function, which takes input as a secret key and an arbitrary-length file that needs to be authenticated, and outputs a MAC. Only users with the same key generating the MAC can verify the correctness of the MAC value and detect whether the file has been changed or not.

3.2 The File-level Distributed Deduplication System

To support efficient duplicate check, tags for each file will be computed and are sent to S-CSPs. To prevent a collusion attack launched by the S-CSPs, the tags stored at different storage servers are computationally independent and different. We now elaborate on the details of the construction as follows.

System setup. In our construction, the number of storage servers S-CSPs is assumed to be n with identities denoted by $id1, id2, \cdots, idn$, respectively. Define the security parameter as 1_ and initialize a secret sharing scheme SS = (Share, Recover), and a tag generation algorithm TagGen. The file storage system for the storage server is set to be $\perp$.

File Upload. 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 $\phi F = \text{TagGen}(F)$ to S-CSPs for the file duplicate check.

If a duplicate is found, the user computes and sends $\phi F;idj = \text{TagGen}'(F, idj)$ to the j-th server with identity idj via the secure channel for $1 \leq j \leq n$ (which could be implemented by a cryptographic hash function $Hj(F)$ related with index j). The reason for introducing an index j is to prevent the server from getting the shares of other S-CSPs for the same file or block, which will be explained in detail in the security analysis. If $\phi F;idj$ matches the metadata stored with ϕF , the user will be provided a pointer for the shard stored at server idj .

- Otherwise, if no duplicate is found, the user will proceed as follows. He runs the secret sharing algorithm

SS over *F* to get *{cj}* = Share(*F*), where *cj* is the *j*-th shard of *F*. He also computes $\phi F;idj$ = TagGen′(*F*, *idj*), which serves as the tag for the *j*th S-CSP. Finally, the user uploads the set of values $\{\phi F , cj , \phi F;idj\}$ to the S-CSP with identity *idj* via a secure channel. The S-CSP stores these values and returns a pointer back to the user for local storage.

File Download. To download a file *F*, the user first downloads the secret shares *{cj}* of the file from *k* out of *n* storage servers. Specifically, the user sends the pointer of *F* to *k* out of *n* S-CSPs. After gathering enough shares, the user reconstructs file *F* by using the algorithm of Recover(*{cj}*). This approach provides fault tolerance and allows the user to remain accessible even if *any* limited subsets of storage servers fail.

3.3 The Block-level Distributed Deduplication System

In this section, we show how to achieve the fine-grained block-level distributed deduplication. In a block-level deduplication system, the user also needs to firstly perform the file-level deduplication before uploading his file. If no duplicate is found, the user divides this file into blocks and performs block-level deduplication. The system setup is the same as the file-level deduplication system, except the block size parameter will be defined additionally. Next, we give the details of the algorithms of File Upload and File Download.

File Upload. To upload a file *F*, the user first performs the file-level deduplication by sending ϕF to the storage servers. If a duplicate is found, the user will perform the file-level deduplication, such as that in Section 3.2. Otherwise, if no duplicate is found, the user performs the block-level deduplication as follows. He firstly divides *F* into a set of fragments *{Bi}* (where $i = 1, 2, \cdots$). For each fragment *Bi*, the user will perform a block-level duplicate check by computing ϕBi = TagGen(*Bi*), where the data processing and duplicate check of block-level deduplication is the same as that of file-level deduplication if the file *F* is replaced with block *Bi*. Upon receiving block tags $\{\phi Bi\}$, the server with identity *idj* computes a block signal vector σBi for each *i*.

- If σBi=1, the user further computes and sends $\phi Bi;j$ = TagGen′(*Bi*, *j*) to the S-CSP with identity *idj* . If it also matches the corresponding tag stored, S-CSP returns a block pointer of *Bi* to the user. Then, the user keeps the block pointer of *Bi* and does not need to upload *Bi*.
- ii) If σBi=0, the user runs the secret sharing algorithm SS over *Bi* and gets *{cij}* = Share(*Bi*), where *cij* is the *j*-th secret share of *Bi*. The user also computes $\phi Bi;j$ for $1 \leq j \leq n$ and uploads the set of values $\{\phi F , \phi F;idj , cij , \phi Bi;j\}$ to the server *idj* via a secure channel. The S-CSP returns the corresponding pointers back to the user.

File Download. To download a file *F* = *{Bi}*, the user first downloads the secret shares *{cij}* of all the blocks *Bi* in *F* from *k* out of *n* S-CSPs. Specifically, the user sends all the pointers for *Bi* to *k* out of *n* servers. After gathering all the shares, the user reconstructs all the fragments *Bi* using the algorithm of Recover(*{·}*) and gets the file *F* = *{Bi}*.

FUTURE ENHANCEMENT

4.1 Distributed Deduplication System with Tag Consistency

In this section, we consider how to prevent a duplicate faking or maliciously-generated ciphertext replacement attack. A security notion of tag consistency has been formalized for this kind of attack [6]. In a deduplication storage system with tag consistency, it requires that no adversary is able to obtain the same tag from a pair of different messages with a non-negligible probability. This provides security guarantees against the duplicate faking attacks in which a message can be undetectably replaced by a fake one. In the previous related work on reliable deduplication over encrypted data, the tag consistency cannot be achieved as the tag is computed by the data owner from underlying data files, which cannot be verified by the storage server. As a result, if the data owner replaces and uploads another file that is different from the file corresponding to the tag, the following users who perform the duplicate check cannot detect this duplicate faking attack and extract the exact files they want. To solve this security weakness, [6] suggested to compute the tag directly from the ciphertext by using a hash function. This solution obviously prevents the ciphertext replacement attack because the cloud storage server is able to compute the tag by itself. However, such a method is unsuitable for the distributed storage system to realize the tag consistency. The challenge is that traditional secret sharing schemes are not deterministic. As a result, the duplicate check for each share stored in different storage servers will not be the same for all users. In [11], though they mentioned the method of deterministic secret sharing scheme in the implementation, the tag was still computed from the whole file or ciphertext, which means the schemes in [11] cannot achieve the security against duplicate faking and replacement attacks.

4.1.1 Deterministic Secret Sharing Schemes

We formalize and present two new techniques for the construction of the deterministic secret sharing schemes. For simplicity, we present an example based on traditional Shamir's Secret Sharing scheme. The description of (k, n)-threshold in Shamir's secret sharing scheme is as follows. In the algorithm of Share, given a secret $\alpha \in Zp$ to be shared among n users for a prime p, choose at random a $(k-1)$-degree polynomial function $f(x) = a0 + a1x + a2x2 + \cdots + ak-1xk-1 \in Zp[X]$ such that $\alpha = f(0)$. The value of $f(i)$ *mod* p for $1 \le i \le n$ is computed as the i-th share. In the algorithm of Recover, Lagrange interpolation is used to compute α from any valid k shares. The deterministic version of Shamir's secret sharing scheme is similar to the original one, except all the random coefficients $\{ai\}$ are replaced with deterministic values. We describe two methods to realize the constructions of deterministic secret sharing schemes below.

The First Method

Share. To share a secret $\alpha \in Zp$, it chooses at random a $(k-1)$-degree polynomial function $f(x) = a0 + a1x + a2x2 + \cdots + ak-1xk-1 \in Zp[X]$ such that $\alpha = f(0)$, $ai = H(\alpha \| i)$ and p is a prime, where $H(\cdot)$ is a hash function. The value of $f(i)$ *mod* p for $1 \le i \le n$ is computed as the i-th share and distributed to the corresponding owner. Recover. The description of algorithm Recover is the same with the traditional Shamir's secret sharing scheme by using Lagrange interpolation. The secret α can be recovered from any valid k shares. For files or blocks unknown to the adversary, these coefficients are also confidential if they are unpredictable. To show its security, these values can be also viewed as random coefficients in the random oracle model. Obviously, these methods can be also applied to the RSSS to realize deterministic sharing.

The Second Method

Obviously, the first method of deterministic secret sharing cannot prevent brute-force attack if the file is predictable. Thus, we show how to construct another deterministic secret sharing construction method to prevent the brute-force attack. Another entity, called key server, is introduced in this method, who is assumed to be honest and will not collude with the cloud storage server and other outside attackers. System Setup. Apart from the parameters for the first deterministic secret sharing scheme, the key server chooses a key pair (pk, sk) which can be initialized as RSA cryptosystem. Share. To share a secret α, the user first computes $H(\alpha \| i)$ for $1 \le i \le k-1$. Then, he interacts with the key server in an oblivious way such that the key server generates a blind signature on each $H(\alpha \| i)$ with the secret key sk without knowing $H(\alpha \| i)$. For simplicity, we denote the signature as $\sigma i = \varphi(H(\alpha \| i), sk)$, where φ is a signing algorithm. Finally, the owner of the secret chooses at random a $(k-1)$-degree polynomial function $f(x) = a0 + a1x + a2x2 + \cdots + ak-1xk-1 \in Zp[X]$ such that $\alpha = f(0)$ and $ai = \sigma i$. The value of $f(i)$ *mod* p for $1 \le i \le n$ is computed as the i-th share and distributed to the corresponding owner. Recover. It is the same with the traditional Shamir's secret sharing scheme. In the second construction, the secret key sk is applied to compute the value of σi. Thus, for the cloud storage server and other outside attackers, they cannot get any useful information from the short value even if the secret is predictable [5]. Actually, the signature can be viewed as a pseudorandom function for α.

4.1.2 The Construction of Distributed Deduplication

System with Tag Consistency We give a generic construction that achieves tag consistency below.

System setup. This algorithm is similar to the above construction except a deterministic secret sharing scheme SS = (Share, Recover) is given.

File Upload. To upload a file F, the user first performs the file-level deduplication. Different from the above constructions, the user needs to compute the secret shares $\{Fj\}1 \le j \le n$ of the file by using the Share algorithm. Then, ϕFj = TagGen(Fj) is computed and sent to the j-th S-CSP for each j. It is the same as above if there is a duplicate. Otherwise, the user performs the block-level deduplication as follows. Note that each server idj also needs to keep ϕFj with the following information of the blocks. The file F is firstly divided into a set of fragments $\{Bi\}$ (where $i = 1, 2, \cdots$). For each block, the duplicate check operation is the same as the file-level check except file F is replaced with block Bi. Assume that the secret shares are $\{Bij\}$ for $1 \le j \le n$ and corresponding tags are ϕBij for block Bi, where $1 \le j \le n$. The tag ϕBij is sent to the the server with identity idj . A block pointer of Bi from this server is returned to the user if there is a match. Otherwise, the user uploads the Bij to the server idj via a secure channel and a pointer for this block will also be returned back to the user. The procedure of the file download is the same as the previous block-level deduplication scheme in Section 3.3. In this construction, the security relies on the assumption that there is a secure deterministic secret sharing scheme.

4.2 Enhanced Deduplication System with Proof of Ownership

Recently, Halevi [12] pointed out the weakness of the security in traditional deduplication systems with only a short hashing value. Halevi showed a number of attacks that can lead to data leakage in a storage system supporting client-side deduplication. To overcome this security issue, they also presented the concept of Proof of Ownership (PoW) to prevent these attacks. PoW [12] enables users to prove their ownership of data copies to the storage server. Specifically, PoW is implemented as an interactive algorithm (denoted by PoW) run by a prover (i.e., user) and a verifier (i.e., storage server). The verifier derives a short tag value $\phi(F)$ from a data copy F. To prove the ownership of the data copy F, the prover needs toi) compute and send ϕ' to the verifier, and ii) present proof to the storage server that he owns F in an interactive way with respect to ϕ'. The PoW is successful if $\phi' = \phi(F)$ and the proof is correct. The formal security definition for PoW roughly follows the threat model in a content distribution network, where an attacker does not know the entire file, but has accomplices who have the file. The accomplices follow the "bounded retrieval model" so they can help the attacker obtain the file, subject to the constraint they must send fewer bits than the initial min-entropy of the file to the attacker [12]. Thus, we also introduce Proof of Ownership techniques in our construction to prevent the deduplication systems from these attacks. Furthermore, we also consider how to achieve the integrity of the data stored in each S-CSP by using the message authentication code. We now show how to integrate PoW and the message authentication code in our deduplication systems. The system setup is similar to the scheme in Section 3.3 except two PoW notions are additionally involved. We denote them by POWF and POWB, where POWF is PoW for file-level deduplication and POWB is PoW for block-level deduplication, respectively.

File Upload. To upload a file F, the user performs a file-level deduplication with the S-CSPs, as in Section 3.3. If a file duplicate is found, the user will run the PoW protocol POWF with each S-CSP to prove the file ownership. More precisely, for the j-th server with identity idj , the user first computes $\phi F;idj$ = TagGen′(F, idj) and runs the PoW proof algorithm with respect to $\phi F;idj$. If the proof is passed, the user will be provided a pointer for the piece of file stored at j-th S-CSP. Otherwise, if no duplicate is found, the user will proceed as follows. He first divides F into a set of fragments $\{Bi\}$ (where $i = 1, 2, \cdots$). For each fragment Bi, the user will perform a block-level duplicate check, such as the scheme in Section 3.3.

- If there is a duplicate in S-CSP, the user runs PoWB on input $\phi Bi;j$ = TagGen′(Bi, idj) with the server to prove that he owns the block Bi. If it is passed, the server simply returns a block pointer of Bi to the user. The user then keeps the block pointer of Bi and does not need to upload Bi.
- Otherwise, the user runs the secret sharing algorithm SS over Bi and gets $\{cij\}$ = Share(Bi), where cij is the j-th secret share of Bi. The values of (cij , $\phi Bi;j$) will be uploaded and stored by the j-th S-CSP. Finally, the user also computes the messageauthentication code of F as $macF = H(kF , F)$, where the keys are computed as $kF = H0(F)$ with a cryptographic hash function $H0(\cdot)$. Then, the user runs the secret sharing algorithm SS over $macF$ as $\{mfj\}$ = Share($macF$), where mfj is the j-th secret share of $macF$. The user uploads the set of values$\{\phi F , \phi F;idj , mfj\}$ to the S-CSP with identity idj via a secure channel. The server stores these values and returns the corresponding pointers back to the user for local storage.

File Download. To download a file F, the user first downloads the secret shares $\{cij , mfj\}$ of the file from k out of n storage servers. Specifically, the user sends all the pointers for F to k out of n servers. After gathering all the shares, the user reconstructs file F, $macF$ by using the algorithm of Recover($\{\cdot\}$). Then, he verifies the correctness of these tags to check the integrity of the file stored in S-CSPs.

SECURITY ANALYSIS

In this section, we will only give the security analysis for the distributed deduplication system in Section 4. The security analysis for the other constructions is similar and thus omitted here. Some basic cryptographic tools have been applied into our construction to achieve secure deduplication. To show the security of this protocol, we assume that the underlying building blocks are secure, including the secret sharing scheme and the PoW scheme. Thus, the security will be analyzed based on the above security assumptions. In our constructions, S-CSPs are assumed to follow the protocols. If the data file has been successfully uploaded and stored at servers, then the user who owns the file can convince the servers based on the correctness of the PoW. Furthermore, the data is distributedly stored at servers with the secret sharing method. Based on the completeness of the underlying secret sharing scheme, the file will be recovered by the user with enough correct shares. The integrity can be also obtained because the utilization of secure message authentication code. Next, we consider the confidentiality against two types of adversaries. The first type of adversary is defined as dishonest users who aims to retrieve files stored at SCSPs they do not own. The second type of adversary is defined as a group of S-CSPs and users.

Their goal is to get the useful information of file content they do not own individually by launching the collusion attack. The attacks launched by these two types of adversaries are denoted by Type-I attack and Type-II attack, respectively. Because the RSSS is used in our construction, the different level of confidentiality is achieved in terms of the parameter *r* given in the RSSS scheme, hich increases with the number of *r*. Thus, in the following security analysis, we will not explain this furthermore.

Confidentiality against a Type-I Attack

This type of adversary tries to convince the S-CSPs with some auxiliary information to get the content of the file stored at S-CSPs. To get one piece of share stored in a S-CSP, the user needs to perform a correct PoW protocol for the corresponding share stored at the S-CSP. In this way, if the adversary wants to get the *k*-th piece of a share he does not own, he has to convince the *k*-th SCSP by correctly running a PoW protocol. However, the user cannot get the auxiliary value used to perform PoW, if he does not own the file. Thus, based on the security of PoW, the security against a Type-I attack is easily derived.

Confidentiality against a Type-II Attack

As shown in the construction, the data is processed before being outsourced to cloud servers. A secure secret sharing scheme has been applied to split each file into pieces, where each piece is distributedly stored in a SCSP. Because the underlying RSSS secret sharing scheme is semantically secure, the data can not be recovered from pieces of shares that are less than a predefined threshold number. This means the confidentiality of the data stored at the S-CSPs is guaranteed even if some S-CSPs collude. Note that in the RSSS secret sharing scheme, no information will be leaked even if any *r* of *n* shares collude. Thus, the data in our scheme remains secure even if any *r* S-CSPs collude. We also need to consider the security against a colluding attack for PoW protocol because the adversary may also get the data if he successfully convinces the S-CSPs with correct proof in PoW. There are two kinds of PoW utilized in our constructions. These are blocklevel and file-level proof of ownership. Recently, the formal security definition of PoW was formally given in [12]. However, there was one tradeoff security definition. This definition relaxes the restriction that the proof fails unless the accomplices of the adversary send more than a threshold or more bits to the adversary, regardless of the file entropy. Next, we will present a security analysis of the proposed PoW in distributed deduplication systems. Assume there are *t* S-CSPs that would collude and try to extract a user's sensitive file *F*, where $t < k$. We will only present the analysis for file because the security analysis for block is the same. From this assumption, we can model it by providing an adversary with a set of tags $\{\phi F;idi1 , \cdots , \phi F;idit \}$, where $idi1, \cdots , idit$ are the identities of the servers. Furthermore, the interactive values in the proof algorithm between the users and servers with respect to these tags are available to the adversary. Then, the proof of PoW cannot be passed to convince a server with respect to another different tag $\phi F;id\_$, where $id \ast / \in \{idi1, \cdots , idik\}$. Such a PoW scheme with a secure proof algorithm can be easily constructed based on previously known PoW methods. For example, the tag generation TagGen(*F, idi*) algorithm could be computed from the independent Merkle-hash tree with the different cryptographic hash function $Hi(\cdot)$ [12]. Using the proof algorithm in the PoW scheme with respect to $\phi F;idi$, we can then easily obtain a secure proof of ownership scheme with the above security requirement. Finally, based on such a secure PoW scheme and secure secret sharing scheme, we can get the following security result for our distributed deduplication system from the above analysis.

Theorem 1: The proposed distributed deduplication system achieves privacy against the chosen distribution attack under the assumptions that the secret sharing scheme and PoW scheme are secure. The security analysis of reliability is simple because of the utilization of RSSS, which is determined by parameters of *n* and *k*. Based on the RSSS, the data can be recovered from any *k* shares. More specifically, this reliability level depends on $n - k$.

EXPERIMENT

We describe the implementation details of the proposed distributed deduplication systems in this section. The main tool for our new deduplication systems is the Ramp secret sharing scheme (RSSS) [7], [8]. The shares of a file are shared across multiple cloud storage servers in a secure way. The efficiency of the proposed distributed systems are mainly determined by the following three parameters of *n*, *k*, and *r* in RSSS. In this experiment, we choose 4KB as the default data block size, which has been widely adopted for block-level deduplication systems. We choose the hash function SHA-256 with an output size of 32 bytes. We implement the RSSS based on the Jerasure Version 1.2 [13]. We choose the erasure code in the (*n, k, r*)-RSSS whose generator matrix is a Cauchy matrix [14] for the data encoding and decoding. The storage blowup is determined by the parameters *n, k, r*. In more details, this value is $n\ k{-}r$ in theory. All our experiments were performed on an Intelr Xeonr E5530

(2.40GHz) server with Linux 3.2.0-23- generic OS. In the deduplication systems, the (n, k, r)- RSSS has been used. For practice consideration, we test four cases:

- case 1: $r = 1$, $k = 2$, and $3 \leq n \leq 8$ (Figure 3(a));
- case 2: $r = 1$, $k = 3$ and $4 \leq n \leq 8$ (Figure 3(b));
- case 3: $r = 2$, $k = 3$, and $4 \leq n \leq 8$ (Figure 3(c));
- case 4: $r = 2$, $k = 4$, and $5 \leq n \leq 8$ (Figure 3(d)).

As shown in Figure 1, the encoding and decoding times of our deduplication systems for each block (per 4KB data block) are always in the order of microseconds, and hence are negligible compared to the data transfer performance in the Internet setting. We can also observe that the encoding time is higher than the decoding time. The reason for this result is that the encoding operation always involves all n shares, while the decoding operation only involves a subset of $k < n$ shares. The performance of several basic modules in our constructions is tested in our experiment. First, The average time for generating a hash function with 32-byte output from a 4KB data block is 25.196 usec. The average time is 30 ms for generating a hash function with the same output length from a 4MB file, which only needs to be computed by the user for each file. Next, we focus on the evaluation with respect to some critical factors in the (n, k, r)-RSSS. First, we evaluate the efficiency between the computation and the number of SCSPs. The results are given in Figure 2, which shows the encoding/decoding times versus the number of S-CSPs n. In this experiment, r is set to be 2 and the reliability level $n-k = 2$ are also fixed. From Figure 2, the encoding time increases with the number of n since more shares are involved in the encoding algorithm. We also test the relation between the computational time and the parameter r. More specifically, in Figure 3, it shows the encoding/decoding times versus the confidentiality level r. To realize this test, the number of S-CSPs $n = 6$ and the reliability level $n - k = 2$ are fixed. From the figure, it can be easily found that the encoding/decoding time increases with r. Actually, this observation could also be derived from the theoretical result. If we recall that a secret is divided into $k-r$ equalsize pieces in the Share function of the RSSS. As a result, the size of each piece will increase with the size of r, which increases the encoding/decoding computational overhead. From this experiment, we can also conclude it will require much higher computational overhead in order to achieve higher confidentiality. In Figure 4, the relation of the factor of $n - k$ and the computational time is given, where the number of S-CSPs and the confidentiality level are fixed as $n = 6$ and $r = 2$. From the figure, we can see that with the increase of $n-k$, the encoding/decoding time decreases. The reason for this result is based on the RSSS, where fewer pieces (i.e., k) will be required with the increase of $n - k$.

RELATED WORK

Reliable Deduplication systems Data deduplication techniques are very interesting techniques that are widely employed for data backup in enterprise environments to minimize network and storage overhead by detecting and eliminating redundancy among data blocks. There are many deduplication schemes proposed by the research community. The reliability in deduplication has also been addressed by [15], [11], [16]. However, they only focused on traditional files without encryption, without considering the reliable deduplication over ciphertext. Li et al. [11] showed how to achieve reliable key management in deduplication. However, they did not mention about the application of reliable deduplication for encrypted files. Later, in [16], they showed how to extend the method in [11] for the construction of reliable deduplication for user files. However, all of these works have not considered and achieved the tag consistency and integrity in the construction.

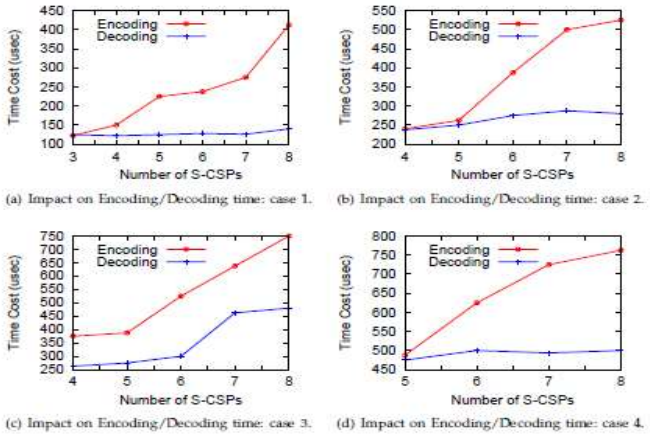

Fig. 1. The Encoding and Decoding time for different RSSS parameters.

ISSN 2349-4506
Impact Factor: 2.265

Global Journal of Engineering Science and Research Management

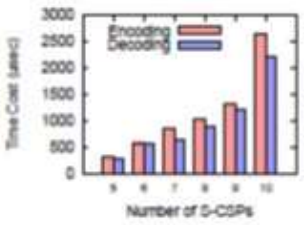

Fig. 2. Impact of number of S-CSPs n on encoding/decoding times, where $r = 2$ and $n-k = 2$.

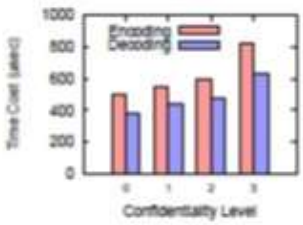

Fig. 3. Impact of confidentiality level r on the encoding/decoding times where $n = 6$ and $n - k = 2$.

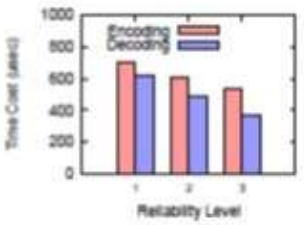

Fig. 4. Impact of reliability level $n-k$ on encoding/decoding times, where $n = 6$ and $r = 2$.

Convergent encryption. Convergent encryption [4] ensures data privacy in deduplication. Bellare et al. [6] formalized this primitive as message-locked encryption, and explored its application in spaceefficient secure outsourced storage. There are also several implementations of convergent implementations of different convergent encryption variants for secure deduplication (e.g., [17], [18], [19], [20]). It is known that some commercial cloud storage providers, such as Bitcasa, also deploy convergent encryption [6]. Liet al. [11] addressed the key-management issue in block-level deduplication by distributing these key across multiple servers after encrypting the files. Bellare et al. [5] showed how to protect data confidentiality by transforming the predicatable message into a

Unpredicatable message. In their system, another third party called the key server was introduced to generate the file tag for the duplicate check. Stanek et al. [21] presented a novel encryption scheme that provided differential security for popular and unpopular data. For popular data that are not particularly sensitive, the traditional conventional encryption is performed. Another two-layered encryption scheme with stronger security while supporting deduplication was proposed for unpopular data. In this way, they achieved better tradeoff between the efficiency and security of the outsourced data.

Proof of ownership. Harnik et al. [22] presented a number of attacks that can lead to data leakage in a cloud storage System supporting client-side deduplication. To prevent these attacks, Halevi et al. [12] proposed the notion of "proofs of ownership" (PoW) for deduplication systems, so that a client can efficiently prove to the cloud storage server that he/she owns a file without uploading the file itself. Several PoW constructions based on the Merkle Hash Tree are proposed [12] to enable client-side deduplication, which includes the bounded leakage setting. Pietro and Sorniotti [23] proposed another efficient PoW scheme by choosing the projection of a file onto some randomly selected bit-positions as the file proof. Note that all of the above schemes do not consider data privacy. Recently, Xu et al. [24] presented a PoW scheme that allows client-side deduplication in a bounded leakage setting with security in the random oracle model. Ng et al. [25] extended PoW for encrypted file, but they did not address how to minimize the key management overhead. *PoR/PDP.* Ateniese et al. [26] introduced the concept of proof of data possession (PDP). This notion was introduced to allow a cloud client to verify the integrity of its data outsourced to the cloud in a very efficient way. Juels et al. [27] proposed the concept of proof of retrievability (PoR). Compared with PDP, PoR allows the cloud client to recover his outsourced data through the interactive proof with the server. This scheme was later improved by Shacham and Waters [28]. The main difference between the two notions is that PoR uses Error Correction/Erasure Codes to tolerate the damage to portions of the outsourced data.

CONCLUSIONS

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.

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-reports/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.

ISSN 2349-4506
Impact Factor: 2.265

Global Journal of Engineering Science and Research Management

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.