Dependable Data Outsourcing Scheme Based on Cloud-of-Clouds Approach with Fast Recovery

Chun-I Fan, *Member, IEEE,* Jheng-Jia Huang, Shang-Wei Tseng, and I-Te Chen*

Abstract—Cloud computing is increasingly popular today. Cloud services such as data-outsourcing services provide a growing number of users access to cloud storage for large quantities of data, and enterprises are turning to cloud storage for cost-effective remote backup. In 2011, DEPSKY shows and overcomes four limitations hinder the effectiveness of cloud storage: loss of availability, loss and corruption of data, loss of privacy, and vendor lock-in. Unfortunately, DEPSKY lacks an error detection mechanism and comes with heavy computing costs. Therefore, we propose a new data-outsourcing scheme overcoming not only the four limitations, but also the shortcomings of DEPSKY. In this manuscript, we modify Nyberg's accumulator and apply it to our three proposed error-detection methods. Moreover, we specially design a fast recovery method that is faster than DEPSKY and alternative methods.

Index Terms—Data Privacy, Cloud Computing, Cloud-of-clouds Approach, Data Outsourcing, Dependable System

1 Introduction

CLOUD computing, an Internet-based computer technology, is becoming increasingly popular today. Significant research has been conducted on cloud computing [1], [2], [3], [4], [5], [6]. Infrastructure-as-a-Service (IaaS) [7], [8], [9] is providing computing resources, such as virtual hard disks, virtual central processing units (CPUs) and even virtual desktops for cloud service developers. Moreover, Software as a Service (SaaS) [10], [11], [12], [13] based on IaaS, is changing the way people use software.

Compared with the traditional way of using software, SaaS is more convenient and flexible for the users. With an increase in network bandwidth, and the development of computer technology, SaaS provides an enhanced user experience with which users can subscribe to high-quality software services over the Internet. Furthermore, cloud-storage services have becomes increasingly prevalent in daily life, enabling users to share data, backup documents, and even develop special systems under SaaS.

In recent years, several SaaS products have been introduced, such as Amazon S3, Amazon EC2, Microsoft Azure Blob Storage, Dropbox, and Google Drive. These online services provide ample storage space, historic data back-up and multimedia synchronization between multiple devices, with data files protected by cloud services for availability and reliability [14], [15]. However, the reliability and security of data files stored in the cloud remains serious concern for most users. In 2011, DEPSKY [16] addressed four important limitations to cloud-storage services, the details for which are descried in what follows.

Loss of availability: The unavailability of cloud service is a common phenomenon on the Internet. There are many reasons why cloud services might be unavailable, and a distributed denial-of-service (DDoS) attack is one of the common reasons. Amazon's EC2 service was attacked with DDoS in 2009 and 2014. The results were that several web services, i.e., GitHub-like services and Code space were unavailable simultaneously. However, cloud services are occasionally unavailable because of human negligence as well. This happened with Amazon's cloud service in 2011. The service was unavailable for approximately 24 hours merely because of a misconfigured network setting. Recently, Google's DNS service was hijacked affecting customers in Brazil for roughly 22 minutes. During that time, anyone seeking a website through Google was directed instead to a dangerous website. Such attacks remain a critical issue to address.

Loss and corruption of data: There are many cases where data is lost using cloud services. In 2009, Danger Inc., a subsidiary of Microsoft, experienced a major service disruption that resulted in the loss of contacts, calendar entries, to-do lists, and photos that were backed up on the server. The disruption was serious enough that T-Mobile shut down Sidekick service provided by Danger. In the same year, Ma.gnolia, a bookmark service, lost a half-terabyte of data, and the service was terminated in 2010. Cloud-service providers should be clearly aware that data-losses from their databases will affect their ability to continue providing a stable service.

Loss of privacy: Cloud-service providers may be trustworthy, but malicious outsiders and insiders are a serious problem. This is a critical concern when the data in question contains private information such as health records, billing records, and credit-card information. Two years in a row (in 2011 and 2012), both Sony and Microsoft were hacked, exposing customers' personal information. In 2013, Evernote's users' passwords were leaked,

- *C.-I. Fan is with the Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung 80424, Taiwan. E-mail: cifan@mail.cse.nsysu.edu.tw*
- *J.-J. Huang is with the Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung 80424, Taiwan. E-mail: jhengjia.huang@gmail.com*
- *S.-W. Tseng is with the Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung 80424, Taiwan. E-mail: f905201@gmail.com*
- *I-Te Chen is with the Department of Healthcare Administration and Medical Informatics, Kaohsiung Medical University, Kaohsiung 807, Taiwan. E-mail: itchen@kmu.edu.tw (*The corresponding author)*

requiring all users to subsequently change their passwords. As a result, a loss of privacy is a legitimate concern for anyone using cloud services.

Vendor lock-in: A vendor lock-in issue refer to a small number of cloud-service providers dominating the market. Users will be affected when the cloud-service provider adjusts the policies of the service. Some cloud-service providers might suddenly terminate the service or limit the transmission flow. Moreover, moving from different geographical regions or different provides is a concern. For example, some cloud-service providers charge more when users move their data from Europe to Asia than they do when data remains within Europe; moving from one provider to another one is also expensive because of the amount of data that is read and written.

There is research aimed at improving cloud-storage service [17], [18], [19], [20]; but until now, only DEPSKY [16] has overcome all of the above mentioned limitations. DEPSKY introduces the concept of a cloud-of-clouds approach [21] which is the best solution for overcoming these four limitations. The cloud-of-clouds approach builds a virtual cloud-storage system by combining several cloud-service providers (CSPs). Hence, users disperse data files across several CSPs. In addition, the virtual cloud-storage system works considerably similar to a file-distribution system or a Redundant Array of Independent Disks (RAID) technique.

DEPSKY enhances the cloud-of-clouds approach by implementing Krawczyks secret sharing scheme [22] to avoid storing clear data (unencrypted data) on the CSPs and to improve the storage efficiency. DEPSKY also offers an alternative to the straightforward solution of encrypting the clear data and storing a copy on each CSP. Therefore, DEPSKY addresses privacy loss and data corruption simultaneously. The cloud-of-clouds approach has made issues such as loss of availability and vendor lock-in redundant for cloud-service users.

As mentioned in DEPSKY, data shadows, which have been stored in a cloud, are not safe since the cloud service provider cannot guarantee the integrity property. They will possibly disappear or be broken when some situations happen, such as power loss, abnormal operations of the server, and malicious attacks. It will make data owners unable to collect enough unbroken shadows to reconstruct the data when the number of broken shadows more than $(n - t)$, where n is the number of all shadows created corresponding to a data file via a secret sharing function and t is the threshold of file reconstruction.

Error detection is a process that is executed before file reconstruction. It can detect and even localize broken shadows such that we can correctly select t unbroken shadows for file reconstruction. Lack of the error detection process, file reconstruction, which is the most time-consuming operation, will be repeatedly performed until all of the selected shadows are unbroken. In worst care, it needs to run $C(n, t)$ times of file reconstruction. Besides, error detection can also be occasionally or periodically performed to detect if too many shadows have been broken. If so, file reconstruction should be performed as soon as possible in order to fix the broken shadows by redistributing of the shadows. Otherwise, once the number of broken shadows more than $(n - t)$, the file cannot be reconstructed and broken shadows cannot be recovered permanently.

DEPSKY is limited by some unresolved problems. First, DEPSKY lacks an error detection mechanism to ensure the correctness of the outsourced shadows which is the encoded data files created by the file distribution procedure. This problem leads to users spending more time reconstructing the data files when using broken shadows. Second, the computing cost is high whenever DEPSKY distributes data files using its underlying secret sharing scheme. In fact, compared with other secret sharing schemes, Krawczyk's is not the most suitable option for data outsourcing. Third, there is heavy cost whenever DEPSKY fixes broken shadows. To fix a broken shadow, DEPSKY must retrieve sufficient shadows to reconstruct the original data file; subsequently, DEPSKY distributes the original data file again to recover the corruption. For these three reasons, we believe that DEPSKY could be improved for cloud data outsourcing environment.

Therefore, in this manuscript, we propose a new data-outsourcing scheme based on the cloud-of-clouds approach to deal with the above problems. The proposed scheme overcomes not only the four general limitations to cloud storage services, but also the three shortcomings in DEPSKY. We apply the (t, L, n) ramp secret sharing [23], [24] to reduce the storage cost of shadows in a cloud-of-clouds approach; and we design three special detection algorithms to find out the broken shadows for different situations. These three algorithms can be executed individually or cooperatively. Besides, it is possible that the cloud service provider is unable to fix an error after localizing it. To cope with this, we design a fast-recovery method to fix the broken shadow when a single broken shadow of a file exists. The fast-recovery method reduces the computation and transmission costs to the user since it is unnecessary to collect shadows and reconstruct the file for recovery. Also, we provide file reconstruction to fix the file when multiple broken shadows exist.

1.1 Contribution

To the best of our knowledge, we are the first ones to apply the (t, L, n) ramp secret sharing scheme to the cloud-of-clouds approach to build a data-outsourcing scheme. Furthermore, we also modify Nyberg's fast accumulator [25] to be our error detection fundamental. Moreover, we construct three different error-detection methods for different situations. Finally, we design a fast-recovery method to fix any errors that cannot be recovered by the cloud-service provider. Our fast-recovery method also preserves the privacy of cloud-service users. We have implemented a prototype of our scheme in JAVA and simulated its effectiveness in a practical setting. It is available for downloading at "https://cifan.g-mail.nsysu.edu.tw/files".

1.2 Organization

In Section 2, we introduce the techniques used in the proposed scheme. In Section 3, some related works, including information dispersal approaches and data-outsourcing approaches, are reviewed. In Section 4, we describe the proposed schemes, including the system model. The comparisons between the proposed scheme and the others are provided in Section 5. Analyses and the conclusion are shown in Section 6 and Section 7, respectively.

2 Preliminaries

In this section, we review several techniques which are used in the proposed scheme.

2.1 (t, L, n) Ramp Secret Sharing

In 1979, Shamir proposed the first secret sharing scheme that is based on polynomial interpolation [26]. The scheme is simple and clear but the space burden is extremely high for users. Among n participants, the size of the shadows is n times larger than the size of the corresponding original secret. To overcome this problem, the (t, L, n) ramp secret sharing scheme was proposed in [23], [24], [27], [28], which aims to reduce the space burden. In the the (t, L, n) ramp secret sharing scheme, we assume that n be a number of total shadows, t is the threshold of n, m is the original data, and $\frac{1}{L}$ is the rate of partial information of m. Thus, each shadow needed the $\frac{1}{L}$ partial information of m. Also, it has the following properties:

1) It can only reconstruct the secret from t shadows.
2) The n shadows are n/L times as long as the original secret, where $L \leq t \leq n$.

In fact, Shamir's method can also be transformed into a ramp secret sharing scheme like the following way [29]: Let secret $s = (s_0, ..., s_{L-1})$ and the shadow S_i is obtained from

$$S_i = s_0 + s_1 x_i + s_2 x_i^2 + ... + s_{L-1} x_i^{L-1} + a_L x_i^L + ... + a_{t-1} x_i^{t-1}$$

where $a_L, ..., a_{t-1}$ are random values.

2.2 One-Way Accumulator

In 1993, Benaloh et al. [30] introduced the principle of accumulated hash as an alternative to digital signatures. In accumulated hashing, the items are hashed together to an accumulated value such that it is possible to prove every item's membership separately in the accumulation regardless of the order of the items. The desired property, quasi-commutative, is obtained by considering function $h : X \times Y \rightarrow X$ for each $x \in X$ and for each $y_1, y_2 \in Y$ such that

$$h(h(x, y_1), y_2) = h(h(x, y_2), y_1).$$

Exponentiation modulo n is used in their native scheme where $e_n(x, y) = x^y \bmod n$. In fact, one-way accumulators are usually constructed by public key cryptographic algorithms, such RSA or ECC, and thus, the cost is high.

In 1996, Nyberg proposed a fast accumulator [25] that is not a trapdoor one and is different from other accumulators. It is based on a general hash function and a simple bit-wise operation so that it is faster than others. The following shows how Nyberg's accumulator works.

Let $N = 2^d$ be the maximum number of the accumulated items, where d is a positive integer. Let $x_1, ..., x_m$ be the items to be accumulated, where $m \leq N$. The one-way hash function h can map an arbitrary-length bit string to a string in a fixed length of r bits. For each x_i, the hashed value is $y_i = h(x_i)$. Besides, let the length of the accumulated hash value be l so that d can be expressed as $r = l \times d$. For each y_i, it is divided into l substrings of length d and denoted as $y_i = (y_{i,1}, ..., y_{i,l})$, where $y_{i,j}$ is the j^{th} bit string of length d. And then for every $y_{i,j}$, it is replaced by bit 1 if $y_{i,j} > 0$ and bit 0 otherwise. Afterward, y_i is transformed into a bit string of length l denoted as $b_i = \alpha_l(y_i)$.

Let H_a be Nyberg's accumulator and k be an initial value of length l used in H_a. The accumulated value Z for $x_1, ..., x_m$ can be expressed as

$$Z = H_a(...H_a(k, x_1), ..., x_m) = k \odot \alpha_l(h(x_1)) \odot ... \odot \alpha_l(h(x_m))$$

where $\odot$ is the logic multiplication operation, which is also known as the AND operation. Due to the above construction, it is quasi-commutative, one-way and also with absorbency, i.e., $H_a(H_a(k, x_1), x_1) = H_a(k, x_1)$.

However, there is a strict limitation in Nyberg's accumulator: the number of the accumulated items are limited which equals $N = 2^d$. In other words, the number of the accumulated items depends on the length of accumulated value. The more accumulated items are required, the longer length of accumulated value it will be. Therefore, it would not be perfectly suitable to apply Nyberg's accumulator to those environment that accumulated items are not fixed.

3 Related Works

In this section, information dispersal approaches and cloud data outsourcing approaches will be reviewed.

3.1 Information Dispersal Approaches

Information dispersal is a good approach to improve reliability, availability, and confidentiality of data in constructing a distributed storage system. The basic idea is decomposing a file into many encoded pieces called shadows and distributing them to different cloud storage providers. In the following, we review some common approaches of information dispersal.

3.1.1 Rabin's Secret Sharing Scheme

Rabin [31] defined the notion of information dispersal algorithms (IDA). His approach used non-systematic erasure codes rather than polynomial secret sharing introduced by Shamir. Basically, the data file F is viewed as a data vector $\overrightarrow{f}$ with k elements and one chooses an invertible $n \times k$ dispersal matrix D and the codeword vector is obtained as $\overrightarrow{c} = D \cdot \overrightarrow{f}$. Reconstruction is achieved by $\overrightarrow{f} = D^{-1} \cdot \overrightarrow{c'}$, whereas $\overrightarrow{c'}$ contains k out of n elements in the codeword vector $\overrightarrow{c}$ and D' is the corresponding submatrix of D.

It reduces the space requirement to be $\frac{n}{k}$ times as compared to Shamir's method. However, this approach does not provide information theoretic confidentiality guarantees and in fact provides extremely weak guarantees. If confidentiality is required, Rabin notes that it is recommended to encrypt the data prior to applying the IDA, which requires an additional concept for key distribution issues.

3.1.2 AONT-RS

In 2011, Resch et al. [18] designed the AONT-RS system by modifying Rabin's method [31] that enriches its confidentiality. The basic idea is to apply a variant of Rivest's all-or-nothing transformation (AONT) as a preprocessing step for the data before using Rabin's method. This approach is as follows: at first, consider $\overrightarrow{f}$ to be a vector with $k+1$ elements, whereas the $k+1^{th}$ element is a canary that is known as a fixed value for integrity checking. Let sk be the encryption key of a symmetric encryption function E. Compute the ciphertext $\overrightarrow{e_i} = \overrightarrow{f_i} \oplus E(i+1, sk)$. After encrypting all $(k+1)$ elements, one computes $\overrightarrow{e_{k+2}} = sk \oplus H(\overrightarrow{e_i} \parallel ... \parallel \overrightarrow{e_{k+1}})$, where H is a collision-resistant hash function. Note that only if all of the first $k+1$ elements of $\overrightarrow{e}$ are available, the key sk can be reconstructed and thus f is decrypted. The canary element is then used to check the integrity. Since the key is implicitly included into the data, the scheme does not require to share the key. After this preprocessing step, a systematic erasure code, Reed-Solomon code (RS-code), is applied. Consequently, the first $k+2$ rows of D compose a $k+2 \times k+2$ identity matrix. Then, one computes $\overrightarrow{c} = D \cdot \overrightarrow{e}$ and the resulting n elements of the codeword vector are stored at the different CSP. The systematic code is employed to save computations for encoding and the reconstruction of f works in the obvious way.

However, the design goal of AONT-RS is to make brute force hard. For example, it is aimed to make a 56-bit-secure system as secure as an 80-bit-secure system. Due to the fast progress of computing capabilities, this strategy is not very useful actually. In our proposed scheme, a 256-bit-secure encryption, such as AES (Advanced Encryption Standard), can be used to encrypt files before distributing them to cloud servers, where the key is long enough to withstand brute force attacks.

3.1.3 *IP-ECC*

An integrity-protected error correcting code (IP-ECC) was introduced with HAIL [17] and is a cryptographic primitive that acts as both a message authentication code (MAC) and an error-correcting (or erasure) code. This recent primitive achieves cross-server redundancy through ECC and at the same time represents a corruption resilient MAC on the underlying data. The construction of IP-ECC code is based on a $(n, l, a-l+1)$ RS-code, where l is the number of the elements in a file vector. To tag a file, it is encoded under the RS code and the last s, a parameter where $1 \leq s \leq n$, code symbols are applied to pseudo-random function (PRF) and transferred to a MAC on each of those s code symbols by using the UMAC construction, which is a combination of a keyed PRF and a keyed universal hash function. A codeword is considered to be valid if at least one of its last s symbols are valid MACs under UMAC on a decoded file.

3.2 Cloud Data Outsourcing Approaches

Subsequently, we review some approaches to cloud storages built on the top of existing cloud services with additional features for different design goals or users.

3.2.1 *Hail*

In 2009, Bowers et al. [17] introduced a concept for distributed cloud storages providing high availability by unifying the use of file redundancy within a cloud provider and across independent cloud providers. HAIL uses an IP-ECC by which it achieves cross-server redundancy through ECC and represents a corruption resilient MAC on the underlying data. IP-ECC is acting as a MAC and an error-correcting code. They additionally applied the proofs of retrievability (PoR) [32] to enable cloud storage providers to prove to a client that all the stored data are still available and reallocate corrupted data when failures are detected (test-and-redistribute technique).

In particular, when detecting an error in one CSP, the user recovers the corrupted shares from cross-server redundancy and resets the faulty CSP with a correct share. Their distributed PoR allows a set of CSPs to prove to a user via a challenge-response protocol that a user can recover a file with overwhelming probability. They do not need to store any check values at the client (like that within other PoR constructions), but obtain such check values from the collection of the CSP itself. HAIL requires the cloud to execute the procedure and this should be done in the cloud, not at some user side, since the access to all files is required. Otherwise, it would introduce too much latency.

However, HAIL does not support dynamic files and no efficient file updates are supported. On the protocol specification, there are five phases: key generation, encoding files, decoding files, verifying correctness of files, and redistribution of shares.

3.2.2 *Cleversafe*

Cleversafe's [33] patented object-based storage solution leverages information dispersal algorithms coupled with encryption to expand, virtualize, transform, slice, and disperse data across a network of storage nodes. In fact, Cleversafe implements the AONT-RS system [18]. This limitless scale storage system stores data much more efficiently than other traditional storage systems that need to maintain multiple copies of the same data. Cleversafe's unique information dispersal architecture uses a single instance of data with a minimal expansion of the data in order to maintain data integrity and availability. As a result, clients experience none of the associated incremental costs found in traditional RAID and replication methods, which makes it ideally scalable to the exabyte and beyond level. Cleversafe's slice-and-disperse technology delivers significantly higher levels of confidentiality, data integrity and information availability than alternative architectures.

Cleversafe's cost-effective storage system operates at the infrastructure layer to capture, store, and deliver unstructured data to enable data management and analytic applications to efficiently process big data delivering intelligence to the organization. Clients can access Cleversafe's storage solution through an internal private cloud deployment or through a managed services provider that offers the capability as a service. Thanks to Cleversafe technology, only a predefined threshold of slices is needed to retrieve the data. Individual servers or entire locations can be down, but you can still access your data.

Because Cleversafe implements the AONT-RS scheme, the weakness of AONT-RS, however, mentioned in previous section also exists. Moreover, how Cleversafe fixes broken data is to run again all procedures including reconstruction and distribution. It costs a user a lot to recover broken data.

3.2.3 *CDStore*

CDStore [20] proposed a deduplication storage solution with clouds system. It can let a cloud service provider release more storage cost with the same shadows, created from the same data. CDStore modifies the AONT-RS system [18] to CAONT-RS. The major change is that it sets the data hash value to be the random value of the AONT-RS system. Thus, all identical data will create the same shadows with the same n and k, where n is the number of clouds and k is the threshold value of n. CDStore can achieve many properties, which includes reliability, security, and cost efficiency guarantees. However, since CDStore modifies the AONT-RS scheme, the weakness of AONT-RS also exists.

3.2.4 *Wang et al.'s Scheme*

In [19], a flexible distributed storage integrity auditing mechanism was proposed by utilizing the homomorphic token and distributed erasure-coded data. The designed scheme allows a user to audit the cloud storage with very lightweight communication and computation cost. The auditing result not only ensures strong cloud storage correctness, but also simultaneously achieves fast data error localization. This system consists of four parts: file distribution preparation, challenge token precomputation, correctness verification and error localization, and file retrieval and error recovery.

However, Wang's scheme has a problem that precomputed detection tokens can only be used t times. After t times of detection, the detection tokens should be computed again. In other words, the user has to retrieve all outsourced data and run all procedures again. The cost might be too high for users.

3.2.5 DEPSKY: Dependable and Secure Storage in a Cloud-of-Clouds

DEPSKY [16] is a distributed cloud storage system which is capable to integrate different CSPs and abstract away from the details of the cloud storage providers by offering an object store interface to its users. In fact, DEPSKY is the first cloud storage system based on cloud-of-clouds approach. Also, DEPSKY requires no computation at CSPs and no communication between CSPs. For all stored data objects, integrity is verifiable by means of digital signatures embedded in the metadata of every object. DEPSKY provides two different algorithms, namely DEPSKY-A and DEPSKY-CA. Also, DEPSKY follows Krawczyk's secret sharing scheme [22], which combines Shamir's and Rabin's schemes, and applies it to the cloud-of-clouds environment.

The (t, L, n) ramp secret sharing scheme, which will be used in our construction, is based on Shamir's interpolation methods. In Shamir's scheme [26], the information is represented by the whole polynomial, which is essential for Shamir's scheme to provide perfect secrecy to guarantee the confidentiality of the information. DEPSKY imports the information-optimal erasure code algorithm (IOECA) [31] to reduce the storage cost of the shadow on each cloud. IOECA was introduced by Rabin's scheme [31] and used in Krawczyk's information secret-sharing scheme [22]. In Rabin's scheme [31], the information is represented by the free coefficient, not from the whole polynomial.

The former protocol (DEPSKY-A) is designed to only improve availability by replicating a data file on several CSPs and all of the copies are plaintexts. The latter version (DEPSKY-CA) provides improved availability and confidentiality of data. The data are encrypted and encoded using an erasure code, and the shares for the encryption key are computed using threshold secret sharing. The key shares along with the slices of an encoded file are then stored at different CSPs. It also provides read freshness by integrating version numbers into the metadata of the data objects. The protocol design is for a single-writer multi-reader setting, but it also includes a lock/lease protocol to enforce consistency in the presence of multiple writers to the same data object.

However, the problem of DEPSKY is that DEPSKY has no error detection and error localizing protocols. Moreover, the cost of file distribution and error recovery is high due to the underlying secret sharing scheme.

4 Our Construction

The proposed dependable data outsourcing scheme, which can be applied to cloud-of-clouds environment consists of three parts including proper information dispersal algorithm as fundamental storing system, error detection algorithm and fast recovery algorithm. We use (t, L, n) ramp secret sharing scheme based on Shamir's interpolation methods to design our scheme.

To design error detection algorithm with functionality of batch detection, we modify Nyberg's accumulator and apply it to our proposed error detection algorithms. We will discuss error detection algorithms in 4.2. In general, the three error detection algorithms are optional. Each detection procedure can be executed anytime if necessary. Furthermore, the three error detection algorithms can either be performed individually or cooperatively.

The proposed recovery algorithm is valid when only one shadow of a file was broken. Although the recovery algorithm can only fix one broken shadow of a file, it can avoid redistributing the entire file to the servers when only one shadow was broken. The recovery algorithm will be much more efficient than file redistribution in one broken shadow situation.

A scenario of the execution of the proposed algorithms is described as follows. After distributing the shadows of a file to the servers, the user can execute the batch-detection algorithm anytime, even before doing file reconstruction. Once a broken shadow is detected, ring detection can be performed to find the file the broken shadow belongs to and then the single-detection algorithm can help the user to find the location (some CSP) of the broken shadow. If there is only one broken shadow, the user can fix it immediately by running fast-recovery algorithm.

4.1 The System Model

We design our data outsourcing scheme based on a cloud-of-clouds approach (DOS-COCA for short). In our scheme, there are three main roles, $\mathcal{USER}$, $\mathcal{PRIMARY}$-$\mathcal{CSP}$ and $\mathcal{SECONDARY}$-$\mathcal{CSP}$. The DOS-COCA involves six phases, including **File Distribution, Batch Detection, Ring Detection, Single Detection, Fast Recovery** and **File Reconstruction**. $\mathcal{PRIMARY}$-$\mathcal{CSP}$ is a semi-trust cloud service provider that is honest but curious. $\mathcal{PRIMARY}$-$\mathcal{CSP}$ provides the services for a user to outsource her/his data and execute her/his programs. That is, $\mathcal{PRIMARY}$-$\mathcal{CSP}$ provides storage as a service and platform as a service. $\mathcal{SECONDARY}$-$\mathcal{CSP}$ is the second one and the function is same as $\mathcal{PRIMARY}$-$\mathcal{CSP}$. $\mathcal{USER}$ is someone who would like to outsource her/his data to $\mathcal{PRIMARY}$-$\mathcal{CSP}$ securely that is privacy-preserving. $\mathcal{USER}$ is able to detect unexpected errors as well. Moreover, to achieve fast-recoverable, $\mathcal{USER}$ outsources parity information to $\mathcal{SECONDARY}$-$\mathcal{CSP}$ so that by the help of $\mathcal{SECONDARY}$-$\mathcal{CSP}$, $\mathcal{USER}$ is able to fix error at low cost. When a user executes error detection on shadows, some works, such as [17], [19], requires the CSPs to compute the checking tokens of shadows and communicate with the user such that she/he can verify the correctness of the shadows. In the proposed scheme, we outsource most of the error detection overheads to the CSPs in order to reduce the computation and communication costs for the user. Thus, it requires computations at CSPs and direct communications between CSPs for error detection. However, if CSPs are not allowed to directly communicate each other, the user can take the works instead by communicating with all of the CSPs individually. To efficiently fix a single error among the data shadows of a file, the parity CSP has been involved in the scheme, and a single parity CSP is enough to recover the single broken shadow. It has been denoted by $\mathcal{SECONDARY}$-$\mathcal{CSP}$ to just highlight that it stores the parity shadow, so any separate CSP can act as the role.

4.2 The Modified Nyberg's Accumulator

First at all, the notations that we used are listed in TABLE 1.

Quasi-commutation and absorbency are two important properties in Nyberg's fast accumulator and they are simple and efficient to perform member testing. Inspired by Nyberg's fast accumulator, we would like to modify it so that the modified accumulator

TABLE 1
The Notations

Notation	Meaning
n	the number of cloud service providers
L	the number of blocks of a file
t	threshold value for reconstruction
F_m	user's m^{th} file that is going to be distributed
MAC_{F_m}	the hash value of the m^{th} file
q	a large prime
CSP_i	the i^{th} primary cloud service provider
CSP_p	the secondary cloud service provider
$S_{(m,i)}$	the shadow of F_m stored in CSP_i
$S_{(m,p)}$	the parity shadow of F_m stored in CSP_p
$H_a()$	modified Nyberg's accumulator function
DK	user's detection key
BT	a token for batch detection
RT_m	a token for ring detection
$ST_{(m,i)}$	a token for single detection

would be eliminative instead of absorbency and of course is quasi-commutative as well. Thus, the modified accumulator is also based on a general hash function and a simple bit-wise operation **XOR** ($\oplus$) such that it is quasi-commutative, one-way, eliminative, and non-degenerate.

- **The Quasi-Commutative Property:**

$$\begin{aligned}&H_a(H_a(K,x_1),x_2)\\&= (K \oplus \alpha(h(x_1)) \oplus \alpha(h(x_2)))\\&= K \oplus \alpha(h(x_1)) \oplus \alpha(h(x_2))\\&= K \oplus \alpha(h(x_2)) \oplus \alpha(h(x_1))\\&= (K \oplus \alpha(h(x_2))) \oplus \alpha(h(x_1))\\&= H_a(H_a(K,x_2),x_1)\end{aligned}$$

- **The Eliminative Property:**

$$\begin{aligned}&H_a(H_a(H_a(k,x_1),x_2),x_2)\\&= (K \oplus \alpha(h(x_1)) \oplus \alpha(h(x_2)) \oplus \alpha(h(x_2)))\\&= K \oplus \alpha(h(x_1)) \oplus \alpha(h(x_2)) \oplus \alpha(h(x_2))\\&= K \oplus \alpha(h(x_1))\\&= H_a(K,x_1)\end{aligned}$$

- **Non-Degeneracy:**
For any two random bits b_1 and b_2, $Pr[b_1 \odot b_2 = 1] = \frac{1}{2^2}$ and $Pr[b_1 \odot b_2 = 0] = 1 - \frac{1}{2^2}$, where where $\odot$ is the logic multiplication operation, i.e. the **AND** operation. For any u random bits $b_1, b_2, b_3, \cdots, b_u$,

$$Pr[b_1 \odot b_2 \odot b_3 \odot \cdots \odot b_u = 0] = 1 - \frac{1}{2^u}.$$

Let $x_1, x_2, x_3, \cdots, x_u$ be u random v-bit strings.

$$Pr[x_1 \odot x_2 \odot x_3 \odot \cdots \odot x_u = \overbrace{000\cdots0}^{v \text{ bits}}] = (1 - \frac{1}{2^u})^v.$$

The probability $(1 - \frac{1}{2^u})^v$ will be very close to 1 when u is large and it will be 1 when $u \to \infty$. The above is the case in Nyberg's accumulator where we call it degeneracy. In our modified accumulator, $\odot$ is replaced by $\oplus$, which is the **XOR** operation. Thus, $Pr[b_1 \oplus b_2 = 1] = Pr[b_1 \oplus b_2 = 0] = \frac{1}{2}$ and $Pr[b_1 \oplus b_2 \oplus \cdots \oplus b_u = 1] = Pr[b_1 \oplus b_2 \oplus \cdots \oplus b_u = 0] = \frac{1}{2}$. Hence, we have that

$$Pr[x_1 \oplus x_2 \oplus \cdots \oplus x_u = 00\cdots0] = \frac{1}{2^v} = Pr[y = 00\cdots0 \mid y \text{ is a random } v\text{-bit string.}]$$

This is the case in the modified accumulator, where we call it non-degeneracy.

The main idea here is to replace the logical multiplication operation in Nyberg's fast accumulator with the XOR operation. In fact, we can decompose Nyberg's fast accumulator into three parts or steps. The first step is to hash accumulated items. The second step is to reduce the bit length of each hashed accumulated item and also perform another transformation to prevent the accumulated value from being zero. The third step is to do logical multiplication operation on the initial value and all accumulated items and then output an accumulated value. When we replace the logical multiplication operation by the XOR operation, the problem that the accumulated value gradually turns to zero will be solved automatically. In other words, the second step in Nyberg's fast accumulator is not necessary at all. Thus, we prefer to remove it.

Let H_a be the modified fast accumulator, k be an initial value of length r, h be a general collision-resistance hash function with an r-bit output. The accumulated value Z for $x_1, ..., x_m$ can be expressed as

$$Z = H_a(...H_a(k,x_1),...,x_m) = k \oplus h(x_1) \oplus ... \oplus h(x_m)$$

where $\oplus$ is the XOR operation and it is quasi-commutative, one-way, and also eliminative.

Remark: A hash function h with an r-bit output is used in Nyberg's accumulator and the proposed accumulator. In Nyberg's accumulator, to prevent the accumulated value from being zero, the r-bit hash value must be transformed to an l-bit value for accumulation and the number of accumulated items should be less than $N = 2^d$, where $r = l \cdot d$. Free from the above problem and transformation, the proposed accumulator takes the complete r-bit output of the hash function h into accumulation where $r = 256$ (e.g. SHA-256) or 512 (e.g. SHA-512) in practical implementation, which can accommodate the accumulated items in real situations.

4.3 The Proposed Scheme

The details of the proposed scheme are described as follows. Some notations are defined in Table 1.

⋆ **File Distribution** $\{F_m\}_{\forall m \in \{1,..,\theta\}} \to (BT, \{RT_m, \{S_{(m,i)}, ST_{(m,i)}\}_{\forall i \in \{1,..,n,p\}}\}_{\forall m \in \{1,..,\theta\}})$
In this phase, a user would like to outsource her/his data to CSP. User first splits her/his data and encrypts those slices of the data, called shadows. Besides, the user also creates a parity shadow so that a shadow is able to be recovered even if it goes wrong. Thereafter, the user computes error-detection tokens including a batch-detection token, a ring-detection token and single-detection tokens according to those shadows. Finally, the user outsources the shadow to each the CSPs and stores those error-detection tokens in the local site. Assume that a user would like to distribute her/his θ files $F_1, ..., F_\theta$ to CSPs and pre-compute some required tokens for error detection. The user first selects a large prime q and performs **Algorithm 1** File Distribution.

All of the detection tokens are illustrated in Fig 1. After performing the file distribution procedure, the user is able

Algorithm 1 File Distribution

```
Input: ${F_m}_{\forall m \in \{1,..,\theta\}}$
Output: $BT, \{RT_m, \{S_{(m,i)}, ST_{(m,i)}\}_{\forall i \in \{1,..,n,p\}}\}_{\forall m \in \{1,..,\theta\}}$
1:  for $m = 1, ..., \theta$ do /* The loop can be executed in parallel. */
2:      Compute $F_m^* = (F_m \parallel MAC_{F_m})$
3:      Encode $F_m^* = (F_{(m,0)}, F_{(m,1)}, ..., F_{(m,L-1)})$ by using
          $(t, L, n)$ Ramp secret sharing based on Shamir's
          interpolation method.
4:      Select $(t-L)$ random values $a_{(m,k)}$'s in $Z_q$, $L \le k \le t-1$.
5:      for $i = 1, ..., n$ do
6:          Compute the $i$th shadow $S_{(m,i)}$ of $F_m^*$:
              $(y_{(m,i)})$
              $= f(x_{(m,i)})$
              $= F_{(m,0)} + F_{(m,1)}x_{(m,i)} + ... + F_{(m,L-1)}x_{(m,i)}^{L-1}$
              $+a_{(m,L)}x_{(m,i)}^{L} + ... + a_{(m,t-1)}x_{(m,i)}^{t-1} \bmod q$
            where $x_{(m,i)}$ is a random value;
            and set $S_{(m,i)} = (x_{(m,i)}, y_{(m,i)})$.
7:      end for
8:      Compute the parity shadow $S_{(m,p)}$ of $F_m^*$:
          $S_{(m,p)} = S_{(m,1)} \oplus S_{(m,2)} \oplus ... \oplus S_{(m,n)}$.
9:      Compute the ring detection token $RT_m$ for $F_m^*$:
          $RT_m =$
          $H_a(...H_a(DK, S_{(m,1)}), S_{(m,2)}), ..., S_{(m,n)}), S_{(m,p)})$.
10:     Compute the single detection token $ST_{(m,i)}$ for $S_{(m,i)}$:
          $ST_{(m,i)} = H_a(DK, S_{(m,i)})$ for $i = 1, ..., n$ and $p$.
11: end for
12: Compute the batch detection token:
        $BT =$
        $H_a(...H_a(DK, S_{(1,1)}), S_{(1,2)}), ..., S_{(\theta,n)}), S_{(\theta,p)})$.
13: for $i = 1, ..., n$ do
14:     Distribute the $\theta$ shadows $S_{(1,i)}, ..., S_{(\theta,i)}$ to $CSP_i$
15: end for
16: Distribute $\theta$ parity shadows $S_{(1,p)}, ..., S_{(\theta,p)}$ to $CSP_p$
17: Keep all of the detection tokens in local side
```

to execute file reconstruction directly. Especially, it is recommended that performing the batch detection or ring detection algorithms before executing file reconstruction can reduce the risk of reconstructing a broken file.

Storing the detection tokens locally is to protect the detection key DK. Instead, if the detection tokens are stored in an additional CSP, that the CSP is honest should be assumed. Under the assumption, storing the detection tokens in a CSP is a good approach.

⋆ **Batch Detection** $(BT, \{\{S_{(m,i)}\}_{\forall i \in \{1,..,n,p\}}\}_{\forall m \in \{1,..,\theta\}}) \rightarrow (True/False)$

In this phase, the user can detect if there is any broken shadow among all of the data files stored in the CSPs. The user first sends a challenge to a CSP and then it computes a response according to all of the shadows it keeps. After that, the CSP sends it to the other CSP as a challenge. The challenge and response procedure is repeated until all of the CSPs being included. The last CSP returns the corresponding response back to the user. The correctness of all the shadows can be confirmed by the user. To detect if all of the shadows distributed in cloud are correct, the user can perform **Algorithm 2** Batch Detection. Assume that the user is going to detect all files $F_1, ..., F_\theta$ and all

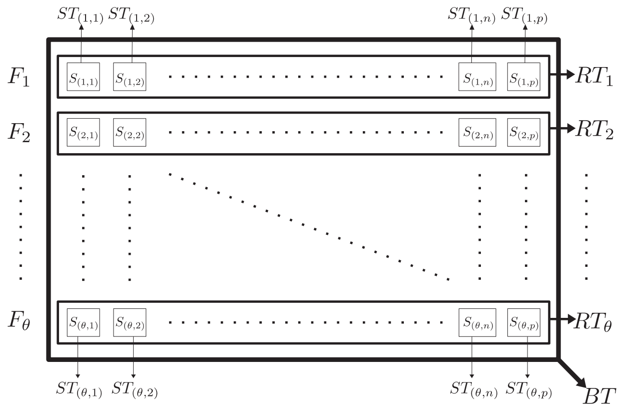

Fig. 1. Detection Tokens

of the CSPs are available.

Algorithm 2 Batch Detection

```
Input: $\{BT, \{S_{(m,i)}\}_{\forall i \in \{1,..,n,p\}}\}_{\forall m \in \{1,..,\theta\}}$
Output: $True/False$
1:  Select a random value $w$ and compute $\alpha = H_a(BT, w)$.
2:  Send $u_1 = H_a(DK, w)$ to $CSP_1$.
3:  for $i = 1, ..., n$ do
4:      $CSP_i$ computes
            $u_{i+1} = H_a(...H_a(u_i, S_{(1,i)}), ..., S_{(\theta,i)})$
5:      $CSP_i$ sends $u_{i+1}$ to the $CSP_{i+1}$.
6:  end for
7:  Set $(CSP_p, u_p) = (CSP_{n+1}, u_{n+1})$.
8:  $CSP_p$ computes
            $u' = H_a(...H_a(u_p, S_{(1,p)}), ..., ), S_{(\theta,p)})$
9:  $CSP_p$ sends $u'$ to the user.
10: if $u' = \alpha$ then
11:     Output $True$. (All shadows of the user are correct.)
12: else
13:     Output $False$. (At least one of the shadows is broken.)
14: end if
```

The user can freely set a running schedule for batch detection. For instances, it can be executed periodically or right before reconstructing the data files. After performing batch detection, if the result shows that some shadow is broken, the ring detection can be performed thereafter to find the file the broken shadow belongs to. Although some CSPs might be unavailable during the batch detection, the user is still able to finish the batch detection procedure with the help of the single detection tokens.

⋆ **Ring Detection** $(RT_m, \{S_{(m,i)}\}_{\forall i \in \{1,..,n,p\}}) \rightarrow (True/False)$

In this phase, the user can detect if there is any broken shadow of a single data file stored in the CSPs. The user first sends a challenge to a CSP and then it computes a response according to the shadow it keeps. After that, the CSP sends it to the other CSP as a challenge. The challenge and response procedure is repeated until all of the CSPs being included. The last CSP returns the corresponding response back to the user. The correctness of all the shadows can be confirmed by the user. To detect if the shadows of file F_m are correct, the user can run **Algorithm 3** Ring Detection. Set CSP_{n+1} to be CSP_p and assume that all of the CSPs are available.

Algorithm 3 Ring Detection

```
Input: $RT_m, \{S_{(m,i)}\}_{\forall i \in \{1,..,n,p\}}$
Output: $True/False$
 1: Select a random value $\tau$ and compute $\alpha = H_a(RT_m, \tau)$.
 2: Send $u_1 = H_a(DK, \tau)$ to $CSP_1$.
 3: for $i = 1, ..., n$ do
 4:     $CSP_i$ sends $u_{i+1} = H_a(u_i, S_{(m,i)})$ to $CSP_{i+1}$
 5: end for
 6: Set $(CSP_p, u_p) = (CSP_{n+1}, u_{n+1})$.
 7: $CSP_p$ sends $u' = H_a(u_p, S_{(m,p)})$ to the user.
 8: if $u' = \alpha$ then
 9:     Output $True$. (All shadows of $F_m$ are correct.)
10: else
11:     Output $False$. (At least one shadow of $F_m$ is broken.)
12: end if
```

The user is able to directly perform ring detection for F_m right before reconstructing F_m. If the result is false, the user can execute the single detection algorithm to find out the broken shadows. Although some CSPs might be unavailable during the ring detection procedure, the user is still able to complete the ring detection procedure with the help of the single detection tokens.

Remark: In the proposed scheme, the user can randomly permute the $(n+1)$ CSPs and re-number them with 1, 2, ..., n, and p, and then she/he sends the new indexes to the CSPs whenever she/he wants to execute Batch Detection or Ring Detection. The correctness is guaranteed due to the quasi-commutative property.

⋆ **Single Detection** $(S_{(m,i)}, ST_{(m,i)}) \rightarrow (True/False)$
In this phase, the user is able to detect whether an individual shadow is correct or not. She/he first sends a challenge to the server that stores a shadow he specifies. The server responds to the user based on the challenge. The user can learn if the specified shadow is correct after verifying the response. To check all shadows of a data file, the above process has to be repeatedly performed. To detect the broken shadows of file F_m, the user can perform **Algorithm 4** Single Detection. Assume that all of the CSPs are available.

Algorithm 4 Single Detection

```
Input: $S_{(m,i)}, ST_{(m,i)}$
Output: $True/False$
 1: Select a random value $\nu_i$.
 2: Send $\Gamma_i = H_a(DK, \nu_i)$ to $CSP_i$.
 3: $CSP_i$ sends $r_i = H_a(\Gamma_i, S_{(m,i)})$ back to the user.
 4: if $r_i = H_a(ST_{(m,i)}, \nu_i)$ then
 5:     Output $True$. (The shadow $S_{(m,i)}$ is correct.)
 6: else
 7:     Output $False$. (The shadow $S_{(m,i)}$ is broken.)
 8: end if
```

Basically, the user is able to find out the broken shadows after performing the single detection. It is convenient for the user to select enough correct shadows to execute the reconstruction algorithm. In addition, if there are some broken shadows, the user should first inform the corresponding CSPs to recover the shadows. If there exists only one broken shadow that the corresponding CSP cannot recover, the user can fix it by running the fast recovery algorithm.

The suggested detection sequence is batch detection, ring detection, and then single detection, noted by the batch-ring-single mode. The batch detection algorithm is performed first. If no error is found, all the shadows of the θ files of the user are unbroken, and it is unnecessary for the user to perform ring detection and single detection. Otherwise, ring detection is executed for each of the θ files to identify those files which contain broken shadows, and then single detection is performed for each shadow of each broken file to localize the broken shadows. Except the suggested mode (numbered by (1)), there are three additional detection modes which are (2) batch-single, (3) ring-single, and (4) single-only modes. It is necessary for the user to perform, in average, $O(1)$, $O(1)$, $O(\theta)$, and $O(n\theta)$ computations in the four different modes, respectively, to determine whether broken shadows exist in the θ files or not, where n is the number of the clouds. If there is no broken shadow, the detection processes will be completed and terminated. Once broken shadows exist, $O(\theta + bn)$, $O(n\theta)$, $O(bn)$, and $O(1)$ computations, in average, are required for the user to localize all the broken shadows in the four modes, respectively, where b $(\leq \theta)$ is the number of the broken files.

⋆ **Fast Recovery** $(\{S_{(m,i)}\}_{\forall i \in \{1,..,n,p\}}/\{S_{(m,k)}\}) \rightarrow S_{(m,k)}$
In this phase, the user can recover a broken shadow with the help of the parity shadow. The user first retrieves two correct shadows, combines them, and sends the combination to the CSP which has the broken shadow. And then, the CSP computes and sends some parameters to the next CSP where the process will be circularly performed until the last CSP sends parameters back to the CSP which has the broken shadow. Finally, the broken shadow will be recovered. If there is only one broken shadow, said $S_{(m,k)}$ of file F_m, **Algorithm 5** Fast Recovery can be performed to fix it. Assume that all of the CSPs are available.

The correctness is demonstrated below.

$$\begin{aligned} RS_p &= US \oplus S_{m,p} \oplus S_{m,1} \oplus S_{m,2} \oplus \cdots \oplus S_{m,k-1} \oplus \\ &\quad S_{m,k+1} \oplus \cdots \oplus S_{m,n} \\ &= US \oplus (S_{m,1} \oplus \cdots \oplus S_{m,n}) \oplus S_{m,1} \oplus S_{m,2} \oplus \\ &\quad \cdots \oplus S_{m,k-1} \oplus S_{m,k+1} \oplus \cdots \oplus S_{m,n} \\ &= US \oplus S_{m,k}. \end{aligned}$$

Hence, CSP_k can recover the shadow $S_{m,k}$ by computing $S_{m,k} = US \oplus RS_p$.

⋆ **File Reconstruction**
In this phase, the user would like to reconstruct her/his data file. He first retrieves enough correct shadows from CSPs and then executes a procedure to reconstruct the data file. The user can reconstruct the file F_m by any t correct shadows as follows.
Step1: Retrieve any t correct shadows, said $S_{(m,i)} = (x_{(m,i)}, y_{(m,i)})$ for $i = 1, .., t$, from t CSPs, respectively,

Algorithm 5 Fast Recovery

```
Input: ${S_{(m,i)}}_{\forall i\in\{1,..,n,p\}} / \{S_{(m,k)}\}$
Output: $S_{(m,k)}$
 1: Set $p = n + 1$.
 2: Retrieve $S_{(m,(k-3 \bmod n+1)+1)}$ from $CSP_{(k-3 \bmod n+1)+1}$.
 3: Retrieve $S_{(m,(k-2 \bmod n+1)+1)}$ from $CSP_{(k-2 \bmod n+1)+1}$.
 4: Send $CSP_k$
        $RS_2 = S_{(m,(k-3 \bmod n+1)+1)} \oplus S_{(m,(k-2 \bmod n+1)+1)}$.
 5: $CSP_k$ selects a pseudo random shadow $US$.
 6: $CSP_k$ sends $RS_3 = US \oplus RS_2$ to $CSP_{(k \bmod n+1)+1}$.
 7: for $i = 4, ..., n$ and $p$ do
 8:     $CSP_{(k+i-4 \bmod n+1)+1}$ sends $CSP_{(k+i-3 \bmod n+1)+1}$
            $RS_i = S_{(m,(k+i-4 \bmod n+1)+1)} \oplus RS_{i-1}$.
 9: end for
10: $CSP_k$ recovers the broken shadow by computing
        $S_{(m,k)} = US \oplus RS_p$.
```

and reconstruct the polynomial by using Lagrange interpolation polynomial theorem.

$$\begin{aligned} f(x) &= \textstyle\sum_{j=1}^{t} y_{(m,j)} l_{(m,j)}(x) \\ &= F_{(m,0)} + F_{(m,1)}x + \ldots + F_{(m,L-1)}x^{L-1} \\ &\quad + a_L x^L + \ldots + a_{t-1}x^{t-1} \end{aligned}$$

where

$$l_{(m,j)}(x) = \prod_{\substack{1\le v\le t \\ v\ne j}} \frac{(x - x_{(m,v)})}{(x_{(m,j)} - x_{(m,v)})}$$

Finally, $F_m^* = (F_{(m,0)}, F_{(m,1)}, ..., F_{(m,L-1)})$.
Step2: The user can check the file F_m by the MAC value MAC_{F_m} where $F_m^* = F_m \parallel MAC_{F_m}$.

Step3: If MAC_{F_m} does not match the file F_m, the user can perform the single detection procedure to find out the broken shadows and inform the corresponding CSPs to fix them.

When executing the file reconstruction algorithm, it is recommended to perform the error detection processes beforehand in order to avoid selecting the broken shadows. In order to verify the correctness of File Reconstruction, it is sufficient to prove that the polynomial $f(x)$ in this phase is indeed the polynomial used to generate shadows in the File Distribution phase. That is, we need to prove that for $i = 1 \ldots t$, $f(x_{(m,i)}) = y_{m,i}$. Note that $l_{m,j}(x)$ has the following property,

$$l_{(m,j)}(x_{(m,i)}) = \begin{cases} 1, \text{if } j = i; \\ 0, \text{otherwise.} \end{cases}$$

Thus,we have that

$$\begin{aligned} & f(x_{m,i}) = \textstyle\sum_{j=1}^{t} y_{(m,j)} l_{(m,j)}(x_{(m,i)}) \\ & = (y_{(m,i)} \cdot 1) + \textstyle\sum_{j\in\{1,...,i-1,i+1,...,t\}} (y_{(m,j)} \cdot 0) \\ & = y_{(m,i)}. \end{aligned}$$

This concludes the proof for the correctness of File Reconstruction.

4.4 Updating Detection Tokens

Every user has three kinds of detection tokens, which are batch-detection token, ring-detection tokens and single-detection tokens. They will be renewed in the following situations. Also, we describe how to update the detection tokens in file creation, file deletion, file modification.

⋆ **File Creation:**
The user should update the batch-detection token and compute a new ring-detection token and new single-detection tokens for a new data file. Assume that file $F_{m'}$ is created and outsourced to the CSPs. Let the shadows of $F_{m'}$ are $(S_{(m',1)}, ..., S_{(m',n)}, S_{(m',p)})$.

Batch detection token: The batch detection token is updated as follows.
$BT' = H_a(...H_a(BT, S_{(m',1)}), ..., S_{(m',n)}), S_{(m',p)})$.

Ring detection token: A new ring detection token, $RT_{m'}$, is created as follows.
$RT_{m'} = H_a(...H_a(DK, S_{(m',1)}), ..., S_{(m',n)}), S_{(m',p)})$.

Single detection token: New single detection tokens $ST_{(m',i)}$'s of $S_{(m',i)}$'s are generated as follows. $ST_{(m',i)} = H_a(DK, S_{(m',i)})$ for $i = 1, ..., n$ and p.

⋆ **File Deletion:**
The user should update the batch-detection token and delete the ring-detection token and single-detection tokens corresponding to a deleted file. Assume that file F_m is about to be removed from the CSPs. The shadows $(S_{(m,1)}, ..., S_{(m,n)}, S_{(m,p)})$ should all be deleted.

Batch detection token: The batch detection token is renewed as $BT' = BT \oplus RT_m \oplus DK$.

Ring detection token: The ring detection token RT_m is deleted.

Single detection token: The single detection tokens $ST_{(m,i)}$'s, $i = 1, ..., n$ and p, are deleted.

⋆ **File Modification:**
The user should update the batch-detection token and compute a new ring-detection token and new single-detection tokens for the modified file. Assume that file F_m is modified to be F'_m. The old shadows $(S_{(m,1)}, ..., S_{(m,n)}, S_{(m,p)})$ should be replaced by the new shadows $(S_{(m',1)}, ..., S_{(m',n)}, S_{(m',p)})$.

Batch detection token: The batch detection token is renewed as follows.
$BT' = H_a(H_a(...H_a(BT, S_{(m',1)}), ..., S_{(m',n)}), S_{(m',p)}) \oplus RT_m \oplus DK$.

Ring detection token: The ring detection token RT_m is renewed as follows.
$RT_{m'} = H_a(H_a(...H_a(DK, S_{(m',1)}), ..., S_{(m',n)}), S_{(m',p)})$.

Single detection token: The single detection tokens $ST_{(m,i)}$'s corresponding to $S_{(m,i)}$'s are renewed as follows. $ST_{(m',i)} = H_a(DK, S_{(m',i)})$ for $i = 1, ..., n$ and p.

Remark: The proposed scheme supports multiple readers and multiple writers. The data owner of a file can share all of the shadows of the file with other users for reading via the clouds. Any of the readers can download the shadows from the clouds and then he can reconstruct the file for reading. Furthermore, the data owner can share the file with other users for writing. A writer can perform the reading operation first. He then updates the file, creates the shadows of the updated file, generate all detection tokens, and upload the shadows to the clouds to overwrite the original ones. The writer will take charge of error detection and recovery thereafter. In our method, the last writer will be always the one who takes over error detection and recovery for the file.

5 Comparison

5.1 Feature Comparison

Based on what we discussed in the Related Works section, we compare the proposed scheme with the other works [16], [17], [19], [20], [33] in some key properties. The comparison is summarized in TABLE 2. Data Confidentiality guarantees the privacy of users' file contents. Error Detection allows one to check the state of the shadows. Unlimited Times of Error Detection means that the data owner can freely execute the error detection without any times limit. Batch Error Detection for Multiple Files makes it possible for the data owner to execute once of error detection to obtain the state of the shadows of all files. Error Localization ensures that the location of any broken shadow can be found. Error Recovery implies that a limited number of broken shadows can be fixed. Dynamic Detection Token guarantees that the error detection parameters can be updated whenever the file contents are modified. The proposed scheme requires computations at CSPs and direct connections between CSPs in the error detection. Alternatively, the user can also act as an intermediate bridge if inter-CSP communication is not available.

TABLE 2
Feature Comparison

	[16]	[17]	[19]	[20]	[33]	Ours
Data Confidentiality	Y	Y	Y	Y	Y	Y
Error Detection	N	Y	Y	N	N	Y*
Unlimited Times of Error Detection	N	Y	N	N	N	Y
Batch Error Detection for Multiple Files	N	N	N	N	N	Y
Error Localizing	N	N	Y	N	N	Y
Error Recovery	Y	Y	Y	Y	Y	Y
Dynamic Detection Token	N	N	Y	N	N	Y

*It requires computations at CSPs and direct connections between CSPs.

The proposed scheme, [17], and [19] provide error detection whose computation costs are shown in TABLE 5. According to [34], [35], $\hat{h} \approx h \approx 0.4T_m$ where $\hat{h}$ is the cost of the modified Nyberg's accumulator operation and h is the cost of a general hash operation. The proposed scheme produces detection tokens with the computation cost $3\theta(n+1)\hat{h}$ which contains $\theta(n+1)\hat{h}$ for the batch detection token, $\theta(n+1)\hat{h}$ for the ring detection tokens, and $\theta(n+1)\hat{h}$ for the single detection tokens. In the process of error detection on the server side, the computation cost is $\theta(3n+2)\hat{h}$, where $\theta(n+1)\hat{h}$ for the batch detection, $\theta(n+1)\hat{h}$ for the ring detections, and $\theta n\hat{h}$ for the single detections. On the user side, the cost is $2(\theta n + \theta + 1)\hat{h}$, including $2\hat{h}$ for the batch detection, $2\theta\hat{h}$ for the ring detections, and $2\theta n\hat{h}$ for the single detections. The schemes of [17] and [19] are more efficient in error detection than the proposed scheme, but they lack the property of batch error detection for multiple files and some other properties shown in TABLE 2.

TABLE 3
Computation Cost of Error Detection

	[17]	[19]	Ours
Detection Token Construction	None	$(\theta n)M$	$3\theta(n+1)\hat{h}$ $\approx 1.2\theta(n+1)T_m$
Detection Cost on Server Side	$(\theta n)h$ $\approx 0.4(\theta n)T_m$	$(\theta n)M$	$\theta(3n+2)\hat{h}$ $\approx 0.4\theta(3n+2)T_m$
Detection Cost on User Side	$(\theta n + \theta)h$ $\approx 0.4(\theta n + \theta)T_m$	$(\theta n)T_m$	$2(\theta n + \theta + 1)\hat{h}$ $\approx 0.8(\theta n + \theta + 1)T_m$

T_m: the cost of a modular multiplication
M: the cost of a multiplication
h: the cost of a general hash operation
$\hat{h}$: the cost of the modified Nyberg's accumulator operation
θ: the number of files
n: the number of CSPs

5.2 Performance Comparison

It is mentioned that one of the drawback of DEPSKY is having high computation cost and communication cost in the file distribution and the error recovery. In this subsection, we compare the computation cost and communication cost with DEPSKY in file distribution and error recovery summarized in TABLE 4.

Consider the computation cost of file distribution. DEPSKY imports Krawczyk's information secret sharing [22] to build a scheme, called DEPSKY-CA. In [22], Krawczyk combines the symmetric encryption, perfect (n, m) secret sharing scheme (PSSS) [26], and Rabin's generic information dispersal algorithm (GIDA) [31] to distribute the file. Thus, the computation cost of the file distribution in DEPSKY-CA contains (1 PSSS) + (1 GIDA) = $(n(t-1)T_m + n(t-1)T_a) + (n(t-1)T_m + n(t-1)T_a)$. In the proposed scheme, we adopt the (t, L, n) ramp secret sharing scheme based on Shamir's interpolation method to distribute the file. Thus, the computation cost of the file distribution in our scheme contains (1 PSSS) = $(n(t-1)T_m + n(t-1)T_a)$. Consider the computation cost of error recovery. DEPSKY-CA needs (1 PSSS) + (1 GIDA) = $(((2t-3)T_m + 1T_e)t + tT_m + (t-1)T_a) + (t^2)$ to reconstruct the file and (1 PSSS) + (1 GIDA) = $(n(t-1)T_m + n(t-1)T_a) + (n(t-1)T_m + n(t-1)T_a)$ to recover each shadow. The proposed scheme needs (1 PSSS) = $(((2t-3)T_m + 1T_e)t + tT_m + (t-1)T_a)$ to reconstruct the file and (1 PSSS) = $(n(t-1)T_m + n(t-1)T_a)$ to recover each shadow.

We carefully calculate the computation costs of DEPSKY-CA and the proposed scheme, where the costs are expressed as clock cycles according to $T_m \approx 66$ clock cycles [36] and shown in TABLE 4. Based on TABLE 4, we illustrate some real cases in four figures. Fig. 2. and Fig. 3. show file distribution costs for one file. Fig. 2. illustrates the computation costs under different threshold values with the same number (n = 10) of CSPs, while Fig. 3. shows the case under different numbers of CSPs with the same threshold t = 5. Fig. 4. and Fig. 5. show some real cases of error recovery costs. Fig. 4. illustrates the computation costs

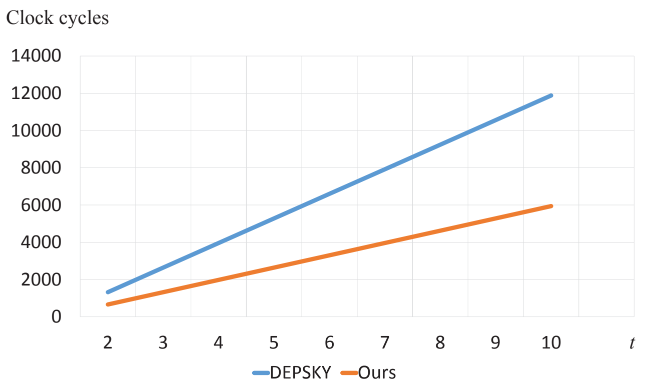

Fig. 2. Computation cost of file distribution with 10 CSPs (n = 10)

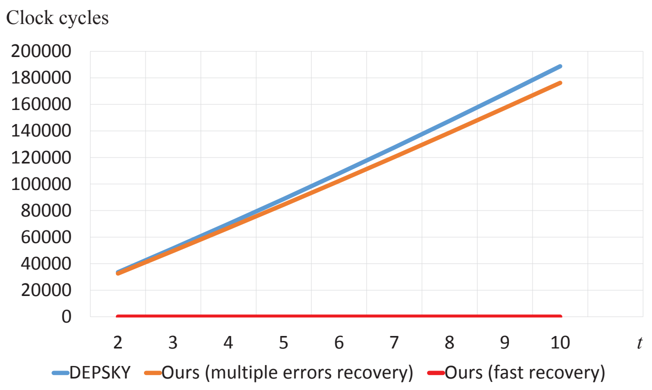

Fig. 4. Computation cost of error recovery with 10 CSPs (n = 10)

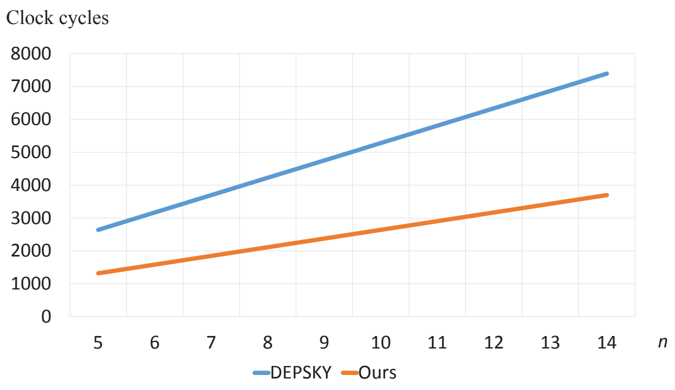

Fig. 3. Computation cost of file distribution with threshold value 5 (t = 5)

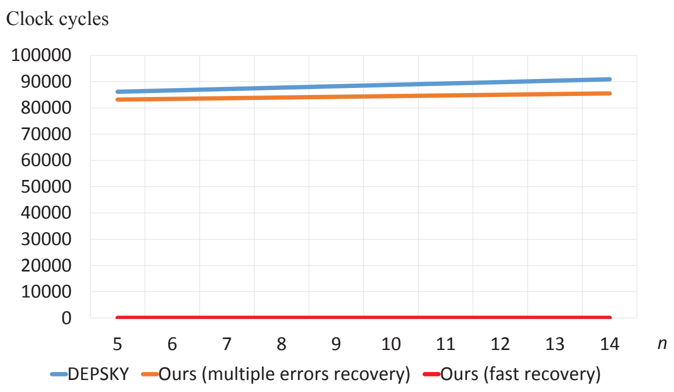

Fig. 5. Computation cost of error recovery with threshold value 5 (t = 5)

under different threshold values with the same number (n = 10) of CSPs, while Fig. 5. shows the case under different numbers of CSPs with the same threshold t = 5. From Fig. 2, to Fig. 5, we can observe that the computation cost of DEPSKY-CA is more than that of the proposed scheme. The major reason is that DEPSKY-CA imports Krawczyk's information secret sharing [22], which caused high computation costs when the number of CSPs or the threshold value is high.

Our scheme provides batch detection, ring detection, and single detection which can help users to detect and localize broken shadows. Also, the proposed error detection algorithms can be performed anytime based on security or management policies. If there is only one broken shadow of a file, Algorithm 5 (Fast Recovery), which executes XOR operations only, can be performed to fast fix it without reconstructing the file (i.e. the polynomial). If multiple broken shadows of a file are found, the unbroken shadows will be correctly chosen to reconstruct the file, and thus, the broken shadows can be fixed by redistributing all of the shadows. However, it will consume much more computation time than that of fast recovery. In DEPSKY, error detection implies file reconstruction no matter how many errors occur. Moreover, due to lack of error localizing in DEPSKY, the procedure of file reconstruction must be repeatedly executed until all of the chosen shadows are unbroken. Once the file is reconstructed, the broken shadows can be fixed by redistributing all of the shadows.

The storage costs of DEPSKY-CA [16] and the proposed schemes are shown in TABLE 4. The storage cost of DEPSKY-CA is $(256n + nL_s)$ and the proposed scheme needs storage cost $(n + 1)L_s$ on the clouds side. The difference between the two schemes in storage cost on the clouds side will depend on the values of L_s and n. For example, if L_s =1024 bits and 2048 bits, the proposed scheme will be better than DEPSKY-CA when $n > 4$ and $n > 8$, respectively. The proposed scheme requires storage cost $(n + 3)AH$ on the local side to store the tokens for error detections, while DEPSKY-CA does not since it does not provide error detections.

How to choose the parameters t and L of the (t, L, n) ramp secret scheme (RSS) is an issue. In RSS, when the program chooses a higher threshold t, it will face higher computation cost of file distribution and file reconstruction due to a higher degree of the polynomial. However, a higher threshold t allows the program to set the space efficient shadows due to a lower rate $\frac{1}{L}$ of the partial information of the data since $L \le t \le n$ [23], [24], [27].

6 System Analysis

In this section, we analyze our scheme in confidentiality and error detection. Moreover, the space requirement of the detection tokens is analyzed as well.

6.1 The Security of Ramp Secret Sharing Scheme

The (t, L, n) ramp secret sharing scheme is classified into strong ramp secret sharing and partially decryptable (PD for short) ramp secret sharing schemes [29]. Generally, the strong ramp secret sharing scheme indicates that no partial secret would leak out without enough shadows. In other words, a strong ramp secret sharing scheme is as secure as a perfect secret sharing scheme

TABLE 4
Performance Comparison

	[16][1]		Ours[2]	
File Distribution	Computation Cost: $2n(t-1)T_m + 2n(t-1)T_a$ $\approx 2n(t-1)T_m \approx 66(2n(t-1))$ clock cycles		$n(t-1)T_m + n(t-1)T_a$ $\approx n(t-1)T_m \approx 66(n(t-1))$ clock cycles	
	Communication Cost: $256n + nL_s$		$(n+1)L_s$	
Error Recovery	Computation Cost: $t^2 + ((2t-3)T_m + 1T_e)t + tT_m + (t-1)T_a$ $+2n(t-1)T_m + 2n(t-1)T_a$ $\approx (3t^2 + 238t + 2nt - 2n)T_m$ $\approx 66(3t^2 + 238t + 2nt - 2n)$ clock cycles		Multiple Errors	$((2t-3)T_m + 1T_e)t + tT_m + (t-1)T_a$ $+n(t-1)T_m + n(t-1)T_a$ $\approx (2t^2 + 238t + nt - n)T_m$ $\approx 66(2t^2 + 238t + nt - n)$ clock cycles
			Single Error	T_{xor} $< T_m <$ 66 clock cycles
	Communication Cost: $512n + 2nL_s$		Multiple Errors	$(t+n+1)L_s$
			Single Error	$3L_s$
Storage Cost (for a file)	The Clouds Side:	$256n + nL_s$	The Clouds Side:	$(n+1)L_s$
	The Local Side:	0	The Local Side:	$(n+3)AH$

[1]: the file size is not greater than $(t \cdot L_s)$.
[2]: the file size is not greater than $(L \cdot L_s)$.
T_e: the cost of a modular exponentiation in Z_q, $T_e \approx 240T_m$
T_m: the cost of a modular multiplication in Z_q. According to [36], $T_m \approx 66$ clock cycles
T_a: the cost of an addition in Z_q
T_{xor}: the cost of an exclusive-or computation
L_s: the bit-length of a shadow
AH: the bit length of the output of the one-way accumulator

while a PD ramp secret sharing scheme cannot achieve perfect secret sharing.

In [29], it has been shown that a ramp secret sharing scheme based on Shamir's interpolation method is not always strong. Consider a $(4, 2, 15)$-threshold ramp secret sharing scheme by using the following polynomial of degree 3 over the finite field $\mathbb{Z}_{17}$.

$$f(x) = S_1 + S_2 x + R_1 x^2 + R_2 x^3,$$

where $\boldsymbol{S} = \{S_1, S_2\}$ is a secret, and R_1 and R_2 are random values. The i^{th} shadow is given by $V_i = f(i)$. From the combination of the shadows, V_3, V_6 and V_{15}, we can get an equation

$$5S_2 = 7V_3 + 9V_6 + V_{15}.$$

It means that the partial secret S_2 is decrypted from the combination of the shadows, V_3, V_6 and V_{15}.

However, the ramp secret sharing scheme based on Shamir's interpolation method does not satisfy the definition of being a PD ramp secret sharing scheme described in [29]. Fortunately, the above example, the $(4, 2, 15)$ case, is the only partially decryptable case that has been found. For this reason, we consider that the ramp secret sharing scheme based on Shamir's interpolation method is still robust so far.

6.2 Error Detection

The error detection methods in our scheme are based on a modified Nyberg's accumulator. To analyze the correctness of the single detection, the ring detection, and the batch detection, it is necessary to analyze every situation in which the detection result should originally be wrong but end up in correctness. Also, considering the detection frequency, if the probability of having corrupted file shares is low, the detections could be executed only before a file reconstruction, which should suffice in catching any single corruptions before reconstruction. However, if the file shares are very susceptible to corruption, it is in the user's best interest to check on the integrity of the file shares periodically. Without the error detections, the user must rely on the chance to find a subset of shares without a corrupted share inside, and that may take several rounds of re-computation; whereas with the error detections, the corrupted share can be instantly identified, and file reconstruction will always succeed on the first try.

Single Detection:
Consider that a shadow stored in a CSP is broken but the broken shadow passes the single detection. It implies that the accumulated value of the original shadow is the same as the accumulated value of the broken shadow. Our modified Nyberg's accumulator is based on a hash function, which also implies that the hash value of the original shadow is the same as the hash value of the broken shadow. However, the hash function is collision-resistance. Hence, there is no such case that a shadow stored in a CSP is broken.

Ring Detection:
A user performs the ring detection to check if all of shadows, $S_{(1,1)}, \ldots, S_{(1,n)}$, are correct. Assume that a hash value is in length of r bits.

- **Case1:** Consider that only one shadow is broken; however, all of the shadows pass the ring detection. It implies that the original accumulated value of all shadows is the same as the accumulated value of the broken shadow and the other shadows. This also implies that the hash value of the original shadow corresponding to the broken shadow is identical to that of the broken shadow. Therefore, the ring detection procedure can show that the broken shadow exists.
- **Case2:** If m shadows $S_{(1,1)}, \ldots, S_{(1,m)}$ are broken but all of the shadows pass the ring detection, then we have the

following

$$\begin{aligned}
&H_a(...H_a(DK,S^{'}_{(1,1)}),...,S^{'}_{(1,m)}),S_{(1,m+1)}),...,S_{(1,n)})\\
&= H_a(...H_a(DK,S_{(1,1)}),...,S_{(1,n)})\\
&\to H_a(...H_a(DK,S^{'}_{(1,1)}),...,S^{'}_{(1,m)})\\
&= H_a(...H_a(DK,S_{(1,1)}),...,S_{(1,m)})\\
&\to DK \oplus h(S^{'}_{(1,1)}) \oplus h(S^{'}_{(1,2)}) \oplus ... \oplus h(S^{'}_{(1,m)})\\
&= DK \oplus h(S_{(1,1)}) \oplus h(S_{(1,2)}) \oplus ... \oplus h(S_{(1,m)})\\
&\to h(S^{'}_{(1,1)}) \oplus h(S^{'}_{(1,2)}) \oplus ... \oplus h(S^{'}_{(1,m)})\\
&= h(S_{(1,1)}) \oplus h(S_{(1,2)}) \oplus ... \oplus h(S_{(1,m)}) = Z
\end{aligned}$$

Let $C_1 = h(S^{'}_{(1,1)}) \oplus h(S^{'}_{(1,2)}) \oplus ... \oplus h(S^{'}_{(1,m)})$ and $C_2 = h(S_{(1,1)}) \oplus h(S_{(1,2)}) \oplus ... \oplus h(S_{(1,m)})$. To pass the ring detection, C_1 must equal to C_2. That is, the i^{th} bit of C_1 must equal to the i^{th} bit of C_2. The probability is shown as follows.

$$\begin{aligned}
Pr[\bigoplus_{j=1}^{m} b_j = b] &= \frac{1}{2}\left(\frac{C_1^m + C_3^m + ...}{2^m} + \frac{C_0^m + C_2^m + ...}{2^m}\right)\\
&= \frac{1}{2}\left(\frac{2^{m-1}}{2^m} + \frac{2^{m-1}}{2^m}\right) = \frac{1}{2}
\end{aligned}$$

where b_j is the i^{th} bit of $h(S^{'}_{(1,j)})$ and b is the i^{th} bit of C_2. The hash value is in length of r bits, and thus the probability of C_1 exactly equals to C_2 is $\frac{1}{2^r}$. As a result, the probability of error detection failure is $\frac{1}{2^r}$.

Batch Detection:
A user performs the batch detection to check if all of the shadows, $S_{(1,1)},...,S_{(1,n)},S_{(2,1)},...,S_{(\theta,n)}$, are correct.

- **Case1:** If only one shadow is broken but all of the shadows pass the batch detection, the analysis is similar to that of Case 1 in the ring detection. Hence, the batch detection procedure can indicate that the broken shadow exists.
- **Case2:** Consider that m shadows $S_{(1,1)},...,S_{(1,m)}$ are broken but all of the shadows pass the batch detection. The analysis is similar to that of Case 2 in the ring detection. The probability of failure in error detection is $\frac{1}{2^r}$ no matter how many broken shadows exist simultaneously.

6.3 The Space Requirement of The Detection Tokens

In this subsection, we would like to compare the space requirement of the detection token stored in local site to the size of a data file. In our scheme, we use (t,L,n)-ramp secret sharing scheme and a secure hash function [37] $H : \{0,1\}^* \to \{0,1\}^h$. Assume that each shadow's size is $|q|$, where q is a large prime and let θ be the number of the data files. Due to the ramp secret sharing scheme, a data files' size is $L|q|$ and $\theta L|q|$ is the size of the total data files. In our scheme, n shadows will be generated for one data file and one single detection token is generated for a shadow. Besides, one ring detection token is generated for one data file and one batch token is generated for all of the data files. That is to say, there are $\theta n + \theta + 1$ detection tokens totally. Because the detection token is as large as a hash value, to store all of the detection tokens in local site requires $(\theta n + \theta + 1)h$. Hence, the reduction ratio of space requirement of storing the detection tokens instead of the original data file is $\frac{\theta L|q| - (\theta n + \theta + 1)h}{\theta L|q|}$.

The reduction ratio strongly relies on the shadow's size $|q|$. It will increase when $|q|$ increases. Considering some cases where h is in length of 256 bits, θ is 100, and (t,L,n) is $(4,3,5)$, the reduction ratio is $\frac{100\times3\times1024-(100\times5+100+1)\times256}{100\times3\times1024} \approx 50\%$ when $|q|$ = 1024. The reduction ratio will be 75% if $|q|$ = 2048, and it will be 87% if $|q|$ = 4096. In other words, when $|q|$ = 1024, 2048, and 4096, the space requirement for storing the detection tokens is about 50%, 25%, and 13%, respectively, that of storing all of the data files. If n is large enough and n almost equals to L, the reduction ratio will close to $1 - \frac{h}{q}$: those are about 75% if $|q|$ = 1024, 87% if $|q|$ = 2048, and 93% if $|q|$ = 4096 in the cases mentioned above.

Storing one duplicate or more duplicates of a data set is a typical way for error recovery of the data set. However, the additional storage costs at least the storage of the data set. The proposed scheme requires less than the storage cost of the data set to store the detection tokens.

7 Implementation and Simulation

We have implemented the proposed scheme to evaluate its effectiveness in a real world scenario. Our implementation covers the functional capabilities in Algorithm 1, Algorithm 2, Algorithm 3, Algorithm 4, and Algorithm 5. Our implementation is for simulation purposes and is run on a machine exclusively. The implementation is hosted on "https://cifan.g-mail.nsysu.edu.tw/files" for readers to download.

7.1 Simulation Environment

The simulation environment is set as TABLE 5:

TABLE 5
Simulation Environment

OS	64bit Windows 10
CPU	Quad Core-i7 at 3.4 GHz
RAM	16GB
Runtime	Java ver 1.8 JVM
File size	4.5 Megabytes

7.2 Simulation Results

In Fig. 6 and Fig. 7, we demonstrate the time taken by file distribution under different conditions. Fig. 6 shows the time taken under various threshold values t from 2 to 10 with a constant CSP number n = 10. Fig. 7 shows the time taken under various CSP numbers n from 5 to 13 with a constant threshold value t = 5.

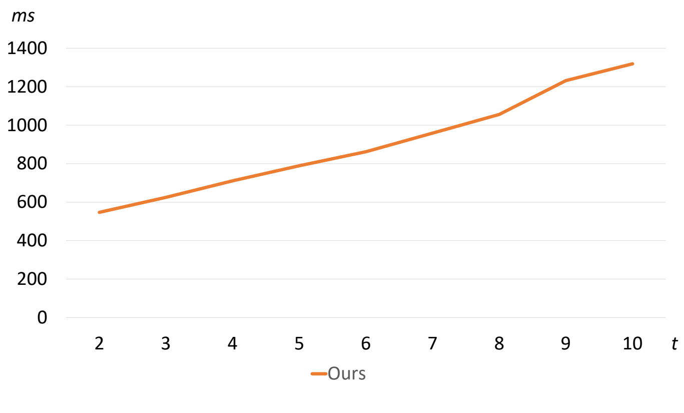

Fig. 6. Simulation result of file distribution with 10 CSPs (n = 10)

In Fig. 8 and Fig. 9 we demonstrate the time taken by error recovery under different conditions. We show both the time taken

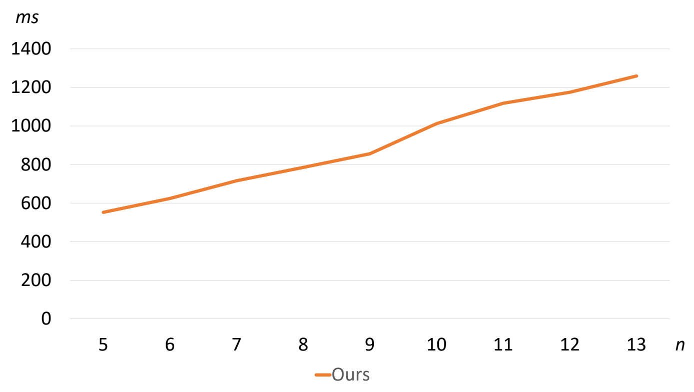

Fig. 7. Simulation result of file distribution with threshold value 5 ($t = 5$)

using fast recovery and full file reconstruction. Fig. 8 shows the time taken under various threshold values t from 2 to 10 with a constant CSP number $n = 10$. Fig. 9 shows the time taken under various CSP numbers n from 5 to 13 with a constant threshold value $t = 5$.

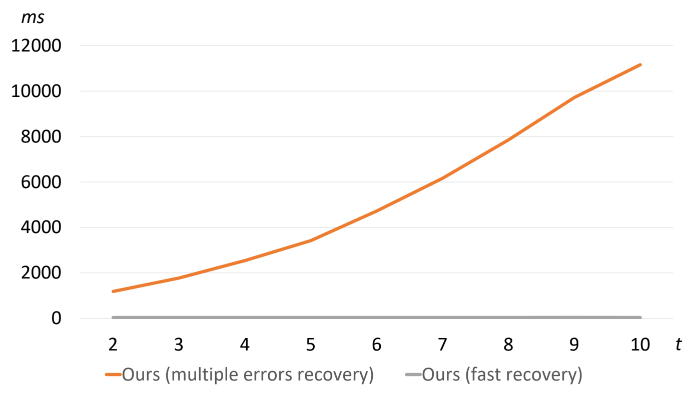

Fig. 8. Simulation result of error recovery with 10 CSPs ($n = 10$)

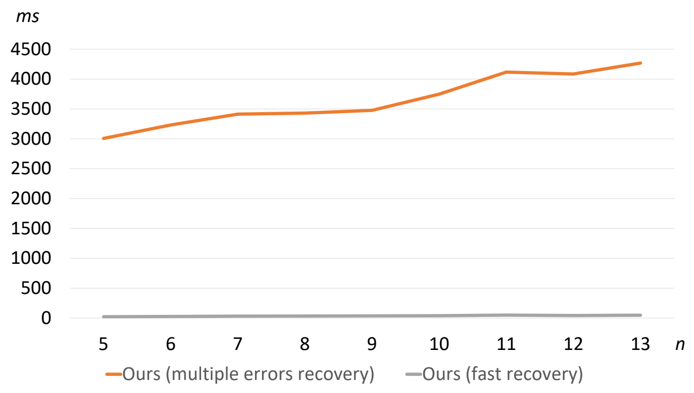

Fig. 9. Simulation result of error recovery with threshold value 5 ($t = 5$)

The time consumption illustrated from Fig. 6 to Fig. 9 is higher than the theoretical values; however this is to be expected as the implementation is executed in a real world environment and the computer cannot use up all the clock cycles on a single application. Another reason why our simulation result differs from the theoretical prediction is that the theoretical calculation did not take file size into consideration. However, there are also some calculations in our implementation that can be reused in further cycles, resulting in less computational cost.

In our simulation, we look into the scaling of the proposed scheme in the number of CSPs and the threshold value. The addition of CSPs or threshold value will result in a relatively steady increase in execution time. This proportional increase in execution time is consistent with our theoretical predictions.

8 Conclusion

Nowadays, an increasing number of users both individuals and enterprises utilize cloud services in their everyday lives. Hence, the cloud-storage service is a considerably popular service. Cloud computing offers a significant amount of storage space, historic data back up, and multimedia synchronization between multiple devices. However, there are four limitations to cloud storage:loss of availability, loss and corruption of data, loss of privacy, and vendor lock-in. Some research has been proposed to improve cloud-storage services, but few of them overcome all mentioned limitations. DEPSKY, the only proposed scheme to address four limitations simultaneously, is hindered by a number of drawbacks.

In addition, it is possible that the cloud-service provider will not be able to fix the errors. Therefore, we designed a better data-outsourcing scheme based on the cloud-of-clouds approach. Our scheme not only overcomes the four limitations to cloud storage, but also provides three special detection algorithms for different situations including a feature for determining whether an error exists and then, if one does, localizing it. We believe that our data-outsourcing scheme based on the cloud-of-clouds approach is dependable and can help users to take advantage of cloud-storage services. We have constructed an implementation of the proposed scheme and made it available for downloading at "https://cifan.g-mail.nsysu.edu.tw/files".

Acknowledgements

This work was supported by the Ministry of Science and Technology of the Taiwan under Grants MOST 105-2923-E-110-001-MY3, MOST 105-2221-E-110-053-MY2, and NSYSU-KMU Joint Research Project (NSYSUKMU 103-I001). The authors would like to thank Mr. Yun-Chun Cheng for his assistance in the implementation of the proposed scheme.

References

[1] U. Moghe, P. Lakkadwala, and D. Mishra, "Cloud computing: Survey of different utilization techniques," in *2012 CSI Sixth International Conference on Software Engineering (CONSEG)*, 2012, pp. 1–4.
[2] B. Rimal, C. Eunmi, and I. Lumb, "A taxonomy and survey of cloud computing systems," in *Fifth International Joint Conference on INC, IMS and IDC, 2009. NCM '09*, 2009, pp. 44–51.
[3] P. Jain, D. Rane, and S. Patidar, "A survey and analysis of cloud model-based security for computing secure cloud bursting and aggregation in renal environment," in *2011 World Congress on Information and Communication Technologies (WICT)*, 2011, pp. 456–461.
[4] J. Xue, D. Rane, and J.-J. Zhang, "A brief survey on the security model of cloud computing," in *2010 Ninth International Symposium on Distributed Computing and Applications to Business Engineering and Science (DCABES)*, 2010, pp. 475–478.
[5] R. Kanday, "A survey on cloud computing security," in *2012 International Conference on Computing Sciences (ICCS)*, 2012, pp. 302–311.
[6] Z. Minqi, Z. Rong, X. Wei, Q. Weining, and Z. Aoying, "Security and privacy in cloud computing: A survey," in *2010 Sixth International Conference on Semantics Knowledge and Grid (SKG)*, 2010, pp. 105–112.

[7] R. Ghosh, F. Longo, X. Wei, F. Frattini, S. Russo, and K. S. Trivedi, "Scalable analytics for iaas cloud availability," *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, pp. 57–70, 2014.

[8] T. Ling, Q. Jinghui, X. Lei, and Y. Yan, "Joint pricing and capacity planning for iaas cloud," in *2014 International Conference on Information Networking (ICOIN)*, 2014, pp. 34–39.

[9] S. Misra, P. V. Krishna, K. Kalaiselvan, V. Saritha, and S. M. Obaidat, "Learning automata-based qos framework for cloud iaas," *IEEE Transactions on Network and Service Management*, vol. 11, no. 1, pp. 15–24, 2014.

[10] D. Juan, D. J. Dean, T. Yongmin, G. Xiaohui, and Y. Ting, "Scalable distributed service integrity attestation for software-as-a-service clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 730–739, 2014.

[11] M.-H. Jeon, B.-D. Lee, and N.-G. Kim, "Adaptive media coding and distribution based on clouds," in *2014 IEEE 3rd Symposium on Network Cloud Computing and Applications (NCCA)*, 2014, pp. 101–104.

[12] A. R. Khan, A. Ahmed, and S. Ahmed, "Collaborative web based cloud services for e-learning and educational erp," in *2014 Recent Advances in Engineering and Computational Sciences (RAECS)*, 2014, pp. 1–4.

[13] A. Polyviou, N. Pouloudi, and S. Rizou, "Which factors affect software-as-a-service selection the most? a study from the customer's and the vendor's perspective," in *2014 47th Hawaii International Conference on System Sciences (HICSS)*, 2014, pp. 5059–5068.

[14] N. S. Sudharsan and K. Latha, "Improvising seeker satisfaction in cloud community portal: Dropbox," in *2013 International Conference on Communications and Signal Processing (ICCSP)*, 2013, pp. 321–325.

[15] Z. Yingwu and J. Masui, "Backing up your data to the cloud: Want to pay less?" in *2013 42nd International Conference on Parallel Processing (ICPP)*, 2013, pp. 409–418.

[16] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "Depsky: Dependable and secure storage in a cloud-of-clouds," in *Proceedings of the Sixth Conference on Computer Systems*, 2011, pp. 31–46.

[17] K. D. Bowers, A. Juels, and A. Oprea, "Hail: A high-availability and integrity layer for cloud storage," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, 2009, pp. 187–198.

[18] J. K. Resch and J. S. Plank, "Aont-rs: Blending security and performance in dispersed storage systems," in *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*, 2011, pp. 14–14.

[19] C. Wang, Q. Wang, K. Ren, N. Cao, and W. Lou, "Toward secure and dependable storage services in cloud computing," *IEEE Transactions on Services Computing*, vol. 5, no. 2, pp. 220–232, 2012.

[20] M. Li, C. Qin, and P. P. Lee, "CDStore: Toward reliable, secure, and cost-efficient cloud storage via convergent dispersal," in *Proceedings of the 2015 USENIX Annual Technical Conference*, 2015, pp. 111–124.

[21] D. Slamanig and C. Hanser, "On cloud storage and the cloud of clouds approach," in *2012 International Conference for Internet Technology And Secured Transactions*, 2012, pp. 649–655.

[22] H. Krawczyk, "Secret sharing made short," in *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, 1994, pp. 136–146.

[23] G. R. Blakley and C. Meadows, "Security of ramp schemes," in *Proceedings of CRYPTO 84 on Advances in Cryptology*, 1985, pp. 242–268.

[24] Y. Kawamoto and H. Yamamoto, "(k,l,n) ramp secret sharing systems for functions," *IEIC*, vol. J68-A, no. 9, pp. 945–952, 1985.

[25] K. Nyberg, "Fast accumulated hashing," in *Proceedings of the Third International Workshop on Fast Software Encryption*, 1996.

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

[27] H. Yamamoto, "Secret sharing system using (k, L, n) threshold scheme." in *Electronics and Communications in Japan (Part I: Communications)*, 1986.

[28] J. Kurihara, S. Kiyomoto, K. Fukushima, and T. Tanaka, "A fast (k,l,n)-threshold ramp secret sharing scheme," *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E92-A, no. 8, pp. 1808–1821, 2009.

[29] M. Iwamoto and H. Yamamoto, "Strongly secure ramp secret sharing schemes for general access structures," *Information Processing Letters*, vol. 97, no. 2, pp. 52–57, 2006.

[30] J. Benaloh and M. de Mare, "One-way accumulators: A decentralized alternative to digital signatures," in *Workshop on the Theory and Application of Cryptographic Techniques on Advances in Cryptology*, 1994, pp. 274–285.

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

[32] A. Juels, B. S. Kaliski, and S. Burton, "Pors: Proofs of retrievability for large files," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 2007, pp. 584–597.

[33] I. CLEVERSAFE, "Cleversafe dispersed storage," in *Community portal:www.cleversafe.org*, 2010.

[34] Z. Li, J. Higgins, and M. Clement, "Performance of finite field arithmetic in an elliptic curve cryptosystem," in *Ninth IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, 2001, pp. 249–256.

[35] K. Takashima, "Scaling security of elliptic curves with fast pairing using efficient endomorphisms," vol. E90-A, pp. 152–159, 2007.

[36] A. MRABET, N. EL-MRABET, R. LASHERMES, J. B. RIGAUD, B. BOUALLEGUE, S. MESNAGER, and M. MACHHOUT, "A systolic hardware architectures of montgomery modular multiplication for public key cryptosystems," Cryptology ePrint Archive, Report 2016/487, 2016, http://eprint.iacr.org/2016/487.

[37] Q. H. Dang, "Secure hash standard," *Federal Information Processing Standards Publication*, no. 180-4, 2015.

Chun-I Fan received the M.S. degree in computer science and information engineering from the National Chiao Tung University, Hsinchu, Taiwan, in 1993, and the Ph.D. degree in electrical engineering from the National Taiwan University, Taipei, Taiwan, in 1998. From 1999 to 2003, he was an Associate Researcher and a Project Leader with Telecommunication Laboratories, Chunghwa Telecom Company, Ltd., Taoyuan, Taiwan. In 2003, he joined the faculty of the Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan, where has been a Full Professor since 2010. His current research interests include applied cryptology, cryptographic protocols, and information and communication security. Prof. Fan is the Deputy Chairman of the Chinese Cryptology and Information Security Association, and the Chief Executive Officer (CEO) of 'Aim for the Top University Plan' Office at National Sun Yat-sen University. He was the recipient of the Best Student Paper Awards from the National Conference on Information Security in 1998, the Dragon Ph.D. Thesis Award from Acer Foundation, the Best Ph.D. Thesis Award from the Institute of Information and Computing Machinery in 1999, and the Engineering Professors Award from Chinese Institute of Engineers - Kaohsiung Chapter in 2016. Prof. Fan is also an Outstanding Faculty in Academic Research in National Sun Yat-sen University.

Jheng-Jia Huang was born in Kaohsiung, Taiwan. He received the M.S. degree in information management from National Kaohsiung First University of Science and Technology, Kaohsiung, Taiwan, in 2012. He now is a Ph.D. student in computer science and engineering from National Sun Yat-sen University, Kaohsiung, Taiwan, His current research interests include cloud computing and security, social network security and authentication, network and communication security, information security, and applied cryptography.

Shang-Wei Tseng was born in Changhua, Taiwan. He now is a MS student in computer science and engineering from National Sun Yat-sen University, Kaohsiung, Taiwan, His research interests include cloud computing and cloud storage, network and communication, and applied cryptography.

I-Te Chen was an associate professor of department of Healthcare Administration and Medical Informatics at Kaohsiung Medical University (KMU). From 2007, he is a Section Chief of Extension Education and Social Resources Center of KMU. He graduated from Mathematical Sciences of National Cheng-chi University in 1995 and received the MS and PhD degree in Computer Science & Information Engineering from National Chiao Tung University Taiwan in 1997 and 2005 respectively. He was a mathematical and computer teacher of Pei-Te vocation school in 1997 and an assistant researcher of Telecommunication Laboratories, Chunghwa Telecom Co., Ltd, Taiwan from Apr. 2001 to Sep. 2002. He holds US patents (US 8,793,296 B2 and US 8,533,493 B1) which are related to this paper. His current research interests include: (1) Cryptography (random number generator, Proxy Signature, Proxy Re-encryption etc.); (2) Watermarking; and (3) Medical Informatics.