A Rapid Locating Protocol of Corrupted Data for Cloud Data Storage

Guangwei Xu[1], Yanbin Yang[1], Cairong Yan[1] and Yanglan Gan[1]
[1] School of Computer Science and Technology, Donghua University
Shanghai, 201620, China
[e-mail: gwxu@dhu.edu.cn]
*Corresponding author: Guangwei Xu

Received December 9, 2015; revised March 23, 2016; revised July 23, 2016; accepted August 18, 2016; published October 31, 2016

Abstract

The verification of data integrity is an urgent topic in remote data storage environments with the wide deployment of cloud data storage services. Many traditional verification algorithms focus on the block-oriented verification to resolve the dispute of dynamic data integrity between the data owners and the storage service providers. However, these algorithms scarcely pay attention to the data verification charge and the users' verification experience. The users more concern about the availability of accessed files rather than data blocks. Moreover, the data verification charge limits the number of checked data in each verification. Therefore, we propose a mixed verification protocol to verify the data integrity, which rapidly locates the corrupted files by the file-oriented verification, and then identifies the corrupted blocks in these files by the block-oriented verification. Theoretical analysis and simulation results demonstrate that the protocol reduces the cost of the metadata computation and transmission relative to the traditional block-oriented verification at the expense of little cost of additional file-oriented metadata computation and storage at the data owner. Both the opportunity of data extracted and the scope of suspicious data are optimized to improve the verification efficiency under the same verification cost.

Keywords: Data integrity, verification metadata, block-oriented verification, file-oriented verification

This research was supported by Key Laboratory of Ecology and Energy-saving Study of Dense Habitat (Tongji University), the National Natural Science Foundation of China (Nos. 61070032 and 61300100), and Shanghai Natural Science Foundation (Nos. 15ZR1400900 and 15ZR1400300).

http://dx.doi.org/10.3837/tiis.2016.10.005 **ISSN : 1976-7277**

1. Introduction

Remote data storage services are increasingly turning into an essential part of people's daily life due to convenient communication and devices, and no distance and storage capacity constraints. Recently, many applications built on remote data storage platforms are increasingly emerging. For instance, data owners (abbreviated to the owner) store their own data in the cloud for users to access, such as Twitter and The New York Times running on Amazon infrastructure. The interested users subscribe owners' data services to get the information they need. However, no storage service can be completely reliable and trusted, i.e., all storage services may potentially lose or corrupt owners' data [1]**.** The storage service providers (abbreviated to the server) may be driven or attacked by malicious motives due to the vulnerabilities such as software bugs, operation failures, and different management levels. Even worse, some servers might drop off owner's data that have not been accessed or rarely accessed to save the storage space and then declare that the data are still intact [2]**.** Many data integrity verification algorithms [3–10] have been proposed to check the correctness of data in the untrusted servers. In these algorithms, a public third-party auditor (abbreviated to the auditor) is requested to fairly verify owner's data integrity as either side of server or owner cannot be guaranteed to provide unbiased verification result. According to the service agreement [3], the users are naturally chosen as auditors to execute the data integrity verification.

However, the traditional verification algorithms are facing new problems as the volume of the big data stored in the cloud is very huge. Also, these problems put forward new requirements: 1) Minimum verification cost. Reffering to the existing storage service charge (e.g. Amazon EC2 pricing), all verification cost should be undertaken by the auditors. Hence, the auditors should pay the minimum charge for the data verification. 2) Maximum coverage of checked data. The existing verification algorithms mainly execute the block-oriented random verification [3, 11]. Thus, only a part of data stored in the cloud can be checked in each verification. The greater the ratio of checked data is, the more reliable the data storage is. 3) Better user experiences. These existing algorithms pay more attention to each block rather than each file due to the special cloud storage structure. Nevertheless, the users care more about the integrity of each file since they are influenced by the traditional operation system which is based on each file. Recently, with the wide deployment of various data services such as e-Business, e-Library, and e-Learning, the number of data stored in the cloud increases at geometric series, especially the small files with the size ranging from a few KBytes to dozens of MBytes. For instance, a shared parallel file system at the National Energy Research Scientific Computing Center of USA showed that it contains over 13 million files, 99% of which were under 64 MBytes and 43% of which were under 64 Kbytes. Also, referring to Zipf's law, the number of small files often occupies a greater proportion in data storage. Therefore, the checked data unit should be coordinated to meet users' experiences.

In this paper, we improve the file-oriented verification algorithm in the metadata generation and verification field to meet the need of small files' verification. To the best of our knowledge, we first propose a novel mixed verification method to verify the data integrity, which combines the file-oriented verification and the block-oriented verification. The mixed verification identifies not only coarse grained data units (i.e., files) but also fine grained data units (i.e., blocks) by different verification metadata. Also, it rapidly and accurately identifies all corrupted or lost blocks in the checked files. Specifically, a

file-oriented verification algorithm detects the potential corrupted or lost files to limit the scope of suspicious data in massive files, and then a block-oriented verification algorithm identifies all corrupted or lost data blocks in these suspicious files which have been checked. Our scheme optimizes the verification cost and efficiency. The theoretical analysis and simulations show that the file-oriented verification algorithm is an excellent verification method for massive small files, and the mixed verification algorithm is a high efficient method to locate the corrupted blocks in massive files.

The remaining of the paper is organized as follows. In section 2, we briefly summarize the current researches on the verification of data integrity. Section 3 describes the verification model, adversary model, and security goals. Section 4 improves a file-oriented verification metadata. Section 5 proposes a mixed verification scheme based on the block-oriented and file-oriented verification. Section 6 analyzes the security of algorithm. Section 7 evaluates the performance of algorithm by simulations. Finally, section 8 concludes the contributions of this paper and presents future extension to the work.

2. Related Work

Since Deswarte et al. [3] introduced a homomorphism-based method to solve the verification of remote data integrity, many representative results are constantly achieved. Ateniese et al. [12] and Juels et al. [6] proposed the Provable Data Possession (PDP) and Proofs of Retrievability (PoR) to check the correctness of data in an untrusted server respectively. These methods utilize the RSA cryptographic technology to generate the verification metadata (or metadata for short) and then audit the integrity of data without retrieving the entire data by sampling strategies. Seb´e et al. [4] subsequently proposed a checking protocol to support the verification of dynamic data. Zhou Hao et al. [5] adapted this protocol to satisfy the demand of public verifiability. Cong Wang et al. [13] combined the public key based on homomorphic authenticator with random masking to achieve the data privacy-preserving in the process of data integrity verification. Soon afterwards, they [14] improved the algorithm to cope with the data in multiple clouds. Zhu et al. [15] and Yang et al. [16] proposed a dynamic audit service for outsourced storage. Recently, Wang et al. [11] advanced a homomorphic authenticable ring signature (HARS) in the privacy-preserving public auditing for shared data in the cloud (Oruta). Liu et al. [10] prevented DSP from concealing the data corruption. Hwang [17] exploited the detecting illegal signature technology and public verification scheme based on BLS short signature technology, and proposed a data integrity verification scheme to report locations of corrupted blocks.

However, the homomorphism-based method above costs a huge computational time to verify the big data due to the exponentiation operation of large integers. Another popular verification method is to utilize the hash-based method. The hash function is utilized to create one MAC (Message Authentication Code) for each data block to authenticate data block's integrity. To build a digest for all data blocks, Juels et al. [6] constructed a Merkle tree to merge these MACs together by hash or XOR. Also, the Bloom filters were used to check data correctness under data privacy-protecting [16]. Aditya et al. [7] replaced the individual hashes of data blocks with it to check the data integrity. Even though this method shortens the computational time of verification relative to the homomorphism-based method, it cannot resist the attack of the metadata replaying as it only supports the verification to execute once.

Considering the feature of data in the cloud storage, more and more researchers already have their focus on the fundamental object in verification changed from a file [19] to a block [11, 12]. Unfortunately, the verification cannot cover all blocks in the file since only a part of

the file is checked by the data random sampling in the traditional verification algorithms. Thus, some static data (i.e, unchanged data) such as precious document files may not be checked in the verification. Considering that these verification algorithms incur heavy computation cost of the auditor [2], we utilize the Bloom filter to generate the metadata and reduce the computation cost in the verificaiton process.

3. Models and problem statement

3.1 System model and assumptions

In this section, we give an overview of the data verification model based on PDP. Generally, once an owner stores her data into a server's storage space, the server has the responsibility to protect the data integrity. Otherwise, the owner will claim compensation against the server for the data corruption. Also, the owner evaluates the quality of storage service to decide which server to be chosen. In [20] and [2], the authors both mentioned an organizer to organize the verification. Considering that an additional trusted organizer isn't practical in cloud storage, and people who are not related to the storage service have no desire to verify the data due to the verification charge. We advise each user, who accesses the data, to act as the auditor to independently check the data integrity and then share the data verification results among each other. That is to say, the request of data verification is launched while a user doubts the integrity of accessed data. Without a doubt, the verification will cost the server and the auditor the additional resource of computation and transmission. For the server, the frequent and unnecessary verification request will affect the performance of the storage service. Certainly, the verification charge also limits users' verification request. Thus, the users have to balance the data verification between the verification cost and the suspect data.

In the verification model, there are three actors as shown in **Fig. 1**, i.e., server, owner, and auditor (user). The process of verification is composed of four phases. The first phase is to prepare the metadata for each data unit by the owner. The owner computes the local metadata via choosing appropriate metadata generation algorithm, and then uploads her data files and signed local metadata into the server. The second phase is to issue the request of data verification by the auditor. The third phase is to recompute and return the remote metadata by the server. The fourth phase is to compare the pairs of metadata (i.e., the local metadata and the remote metadata) by the auditor and then judge whether the corresponding data are intact.

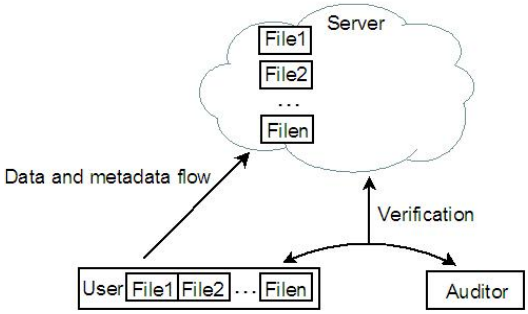


Fig. 1. The verification model of data integrity.

Assumption 1. *Three actors are independent of each other for their own interests and cannot collude to deceive the adversary.*

As the previous description, each auditor independently checks the data integrity and then

shares the data verification results. Once the number of auditors achieves a certain value, the data verification results can be trustworthy referring to the existing research and theoretical proof. Here, we do not discuss the collusion attack due to space limitation.

Let the storage period of owner's data be *T*. Without loss of generality, we select the *i*th file F_i to analyze the block-oriented verification, which is divided into *n* equal blocks, F_{ij}, $j \in [1, n]$. Let G_1, G_2 and G_T be multiplicative cyclic groups of prime order *p*, g_1 and g_2 be generators of G_1 and G_2 respectively. Let $e : G_1 \times G_2 \rightarrow G_T$ be a bilinear map, and $\psi: G_1 \rightarrow G_2$ be a computable isomorphism with $\psi(G_1)=G_2$. Let $H:\{0, 1\}^* \rightarrow G_1$ be a public hash function, which maps a string $\{0, 1\}^*$ into an element in G_1 (i.e., a point on an elliptic curve). Each block-oriented metadata (BoM) for each block is prepared and then used to execute the block-oriented verification. Referring to the algorithms in [5] and [11], we design an algorithm to verify the data block F_{ij}.

In first phase, the owner randomly picks a prime *z* as her private key, and computes $w = g_2^z$ as her public key. Then, the owner randomly chooses a positive integer *a*, and computes $\gamma = g_1^a$ and $\vartheta = g_1^{H(F_{ij}.id)F_{ij}}$, where $F_{ij}.id$ denotes the identity of F_{ij}. It sets $\sigma = \psi(\beta^{1/z})$.

In second phase, the auditor sends *chal*<*w*, $F_{ij}.id$ > to the server.

In third phase, after the server receives *chal*, it computes $\vartheta' = g_2^{H(F_{ij}.id)F_{ij}}$ as the proof of data integrity with respect to the block F_{ij}, and returns ϑ' to the auditor.

In fourth phase, once the auditor receives ϑ', she checks $e(\vartheta', g_2) \stackrel{?}{=} e(\sigma, w)$. If the equation holds, the given block F_{ij} is intact and the auditor outputs "success". Otherwise, it is corrupted and the auditor outputs "failure".

The outsourced data need to be repetitiously verified during the period *T* [15]. We define the duration of each verification from start to finish as one round, which is composed of two parts, i.e., the verification executing period t_e and the idle period t_d. The former is the duration of the verification executing. The latter is the idle time before the next verification. Assume it is enough to execute the verification *t* rounds in the storage period *T*.

Assumption 2. *All the verification information among the server, owner, and auditor is transmitted and distributed through authentic channels which can resist data tampering and intrusion*.

The existing data encryption technology provides many methods to authorize the data transmission channels and resist data tampering and intrusion. Due to space limitation, we do not discuss it here.

Assumption 3. *The probability of data corruption in the remote storage is very low.*

Without doubt, no owner is willing to outsource her data in the remote storage if she takes risks of huge data loss or corruption.

Definition 1 (The ratio of sampled data). In the verification, the sampled data occupies a certain proportion of all the stored data, which is defined as Ω. For example, if the total number of blocks is *N* and the number of sampled blocks is *n*, we have $\Omega=n/N$. The ratio directly impacts the verification cost.

3.2 Adversary model

The goal of the data verification is to protect the integrity and availability of the outsourced

data. The greatest threat is the escape of data corruption, i.e., the lost or corrupted data are concealed and aren't identified in the verification. The attacks of the adversary have three key methods. First, the server provides the false remote metadata to deceive the auditor while the stored data have been corrupted. Second, the server replays the remote metadata to pass the verification by utilizing the same remote metadata in the last verification. Third, the server easily escapes from the corrupted data being identified if less number of data are extracted in the verification. Here, the number of extracted data is limited by the verification charge.

3.3 Problem statement and design goals

The homomorphic-based technology is often applied in the verification as it can accurately identify the corrupted block and resist the attack of metadata replaying. However, the method has some problems.

First, the method brings about huge verification cost while the verification faces the challenge of the big data. Naturally, the method incurs more computation overhead for the server and the auditor than the hash-based operation since it involves the exponential computation of large integers [4], which is also shown in Section 7. With the increase of the number of rounds, the verification should be improved to balance the trade-off between the verification cost (i.e., the verification computation, transmission, and storage) and verification efficiency.

Second, the data random sampling in the traditional block-oriented verification reduces the probability of the small files being extracted. These verification algorithms randomly verify a part of all the data in the verification, and thus they are only a type of probabilistic proof of possession [22]. Assume there are two files (e.g., F_1 and F_2) which are divided into n_1 and n_2 ($n_1<n_2$) equal blocks respectively. Also, assume that the number of data random sampling is N_s. The probability of blocks being extracted in F_1 is less than that in F_2 due to $n_1 / N_s < n_2 / N_s$. If the size of F_1 is far smaller than F_2, the blocks in the file F_1 are hardly extracted in the verification. As the number of small files is very huge, their integrity directly influences the availability of the stored data. Thus, the verification should maximize the scope of checked data at each round to identify the lost or corrupted data with the minimum charge and the shortest time.

Finally, the integrity of one file is more important than several blocks. The traditional verification is oriented to each block instead of each file in the remote storage to accommodate the verification of dynamical data. These verification algorithms scarcely pay attention to each file, especially some precious and small files, due to the special cloud storage structure. For example, in the Google cloud storage, owners' files are aggregated and divided into fixed-size chunks, each with a basic storage unit of size 64 Mbytes. To conveniently read and write data, each chunk is broken up into several blocks of the size 64 Kbytes [19]. After owner's data are stored into the cloud, their storage structure changes from the traditional file system to the big data storage system. Here, the size of each small file is slightly smaller or bigger than one block. However, from the user's perspective, she is more interested in the integrity of each file instead of several blocks in the cloud storage, since a file is entirely convinced to be intact only if all the blocks in the file are at least verified to be intact once. Therefore, the checked data unit should also be coordinated to meet the user's need.

4. Verification metadata design

As the previous analysis, every block isn't verified in each round, especially some blocks in the small files. Hence, it is more significant for an entire file with a small size to be verified in

each round. It is more suitable for each file to be chosen as another basic verification data unit from the user's perspective. In the following, we generate each metadata for each small file, which is defined as the file-oriented metadata (FoM).

Recently, a space-efficient data structure Bloom filter (BF) offers a potential possibility to shorten the verification computation cost, which has widely been applied in many network applications such as web caching, IP traceback, and packet classification [23] , etc. The BF has the advantage in judging whether an element is a member of a set, while at the risk of a low probability of false positives. It is a bit array composed of m bits. Each position of the array is initially set to *0*. It provides several simple operations such as the element inserting and querying. Assume there are g files. Without loss of generality, we analyze the ith file F_i, $i=1,2,\ldots,g$. For the inserting operation, when a new element F_i is added, k array positions $H_1(F_i)$, $H_2(F_i)$,…, $H_k(F_i)$ are set to *1* after k different hash functions $H_1(.)$, $H_2(.)$,…, $H_k(.)$ are fed back to hash F_i. For example, three array positions are set to *1* by three hash functions in **Fig. 2**. For the querying operation, if an element y is queried whether it is in the set, it is mapped into k array positions by the same k hash functions. If any of k array positions is *0*, the element y does not belong to the set. Moreover, if all of k array positions are *1*, either the element is in the set, or these bits are disturbed by other elements in the insertion process (called as false positive). We utilize the BF to generate the metadata of small files and decrease metadata's computation and storage overhead since it has a strong space and computation time advantage over other data structures. Due to the space limitations, we do not discuss the security of the BF in this paper.

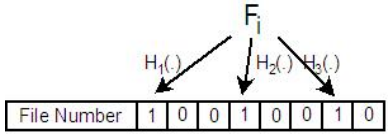


Fig. 2. File-oriented metadata based on the Bloom filter.

4.1 Metadata generation

Each file independently generates the metadata based on the BF by a set of hash functions, where the size of each metadata is similar to [7]. Even if the existing algorithms based on the BF can perform a simple verification, they don't resist the attack of the metadata replaying. Intuitively, if the verification persists for t rounds during the period T, it must generate t metadata for each file to resist the attack of the metadata replaying. However, even if the computation time based on the BF at each round decreases, the cost of metadata computation and storage increases remarkably while the verification executes t rounds or data blocks in every file are frequently updated [25]. Thus, we need to lower the value of t and then optimize the metadata generation by analyzing the frequency of files' verification in the period T.

To store these metadata and identify each other, each file's identification is appended and then bonded together with the corresponding file. The file's identification occupies $\log_2 g$ bits. Let the length of the BF be m bits. Each metadata at least occupies $m+\log_2 g$ bits to support the verification at one round, which is similar to the metadata for each block in [18]. If the amount of rounds achieves to t, all the metadata for the file F_i theoretically occupy

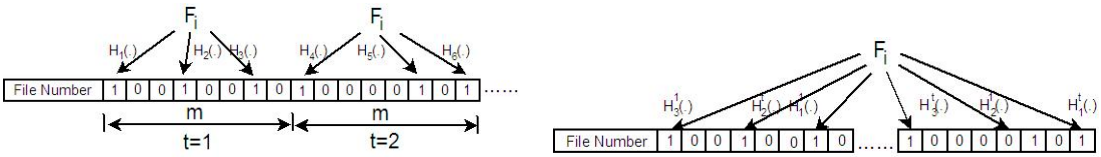
$$S(F_i) = t(m + \log_2 g) \qquad (1)$$

bits. The more the number of files is, the more space the metadata occupies. To decrease the storage space of the metadata, we optimize the structure of all the individual metadata for each file, i.e., t individual metadata for each file are merged into one metadata.

Each metadata for each file needs to support the verification t rounds. A segment of the metadata is extracted to execute the verification at one round, the size of which is $m' + \log_2 g$ bits. There are two optional schemes to design the metadata to support the verification from one round to t rounds. Referring to section 4.2, let $m = e^k / 2$ while the number of hash functions is k. Scheme one is to increase the length of the metadata from m bits to tm bits. Thus, each metadata occupies tm+ $\log_2 g$ (i.e., $te^k / 2 + \log_2 g$) bits, where tm bits are divided into t segments each with the length of m bits in **Fig. 3(a)**. Each segment which is formed by an independent BF with m bits and k hash functions supports the verification at one round. Scheme two is to increase the number of hash functions from k to tk, in **Fig. 3(b)**. For tk hash functions, the length of each metadata occupies $e^{tk} / 2 + \log_2 g$ bits while $m = e^k / 2$ for k hash functions. Obviously, comparing the two schemes, we find $e^{tk} / 2 + \log_2 g > te^k / 2 + \log_2 g$ due to $e^{tk} > te^k$. That is to say, the scheme one is appropriate to generate the metadata for each file. Thus, we let the length of each metadata be tm bits. At each round, it uses m bits of tm bits in the metadata to execute the verification, which are called a sub-metadata. Thus, each metadata is composed of t sub-metadata. The metadata of the file F_i occupies

$$S(F_i) = tm + \log_2 g \qquad (2)$$

bits.



(a) Extending the array from m bits to tm bits (b) Extending the number of hash functions from k to tk

Fig. 3. File-oriented metadata based on the Bloom filter.

Certainly, the amount of the metadata grows with the increase of the number of files. Even if the BF provides convenient operations of inserting and querying, it does not support deleting operation. The dynamic blocks or files need the corresponding metadata to be updated and maintained, especially deleted. If a file needs to be updated or deleted, it only deals with the metadata corresponding to the file instead of interfering with each other. Thus, the file-oriented metadata based on the Bloom filter is suitable for small files.

4.2 Parameters analysis

The size of metadata. Comparing the equation (2) with (1), we know that the storage cost of the merged metadata is $(t-1)\log_2 g$ bits and obviously less than the unmerged metadata. However, with the increase of the number of files, the storage space of all the metadata is very considerable. Hence, the size of each metadata should be designed as small as possible. Moreover, to reduce the disturbance of the false positives in the BF, the length of bit array, m, can't be too short. Let the probability of false positives be f, m has a close relationship with the number of hash functions, k, and the number of inserted elements, q, by $m = -q \ln f / (\ln 2)^2$ and $k = \ln 2(m/q)$.

As each element in the BF is corresponding to each file, q is set to 1. Changing the equation

$k = \ln 2(m/q)$, we have $m = e^k / 2$. Therefore, *m* is optimized to get a lower boundary by

$$m = \min\{\frac{e^k}{2}, \frac{-\ln f}{(\ln 2)^2}\}. \quad (3)$$

It is a tradeoff between *k* and *f*. As *m* must be an integer, it should get value by $\lceil m \rceil$, where $\lceil . \rceil$ means the smallest integer greater than the current value.

The number of rounds. In general, there are two reasons to cause the data corruption or loss. One is data's natural corruption (abbreviated to natural corruption) with the increase of the storage time. The other is data's malicious corruption (abbreviated to malicious corruption) since the server deletes owner's data. The probability of both corruptions is time-related and equal to 0 at the start stage of data storage. Let the rate of the natural corruption for at least one data unit corrupted after a unit time be α and the malicious corruption be β. In [25], the authors found that the linear cumulative probability of at least one data unit mismatch after 17 months for enterprise class disks is 0.6×10^{-3}, where the data unit is a file system block of the length 4kB. In this case, α is equal to 0.35×10^{-4} per month. Thus, the initial probability of the natural corruption, p_0^α, is equal to 0, and the cumulative probability after j months, p_j^α, is equal to $p_0^\alpha + \alpha \times j$. By extension, we let the initial probability of the malicious corruption, p_0^β, equal to 0, and the cumulative probability after j months, p_j^β, equal to $p_0^\beta + \beta \times j$. Thus, the cumulative probability of at least one data unit corrupted after *j* months, p_j, is

$$p_j = 1 - (1 - p_j^\alpha)(1 - p_j^\beta) = p_j^\alpha + p_j^\beta - p_j^\alpha p_j^\beta \quad (4)$$

where β is initialized to 0 and then changed by the iterative computation after the first round. For example, once p_j achieves the threshold *p*, the verification must be executed once. If some corrupted data are identified in the verification, the actual value of p_j, $p_j^{'}$, is got via the number of corrupted data divided by the total number of extracted data at each round [26]. If $p_j^{'}$ is greater than p_j^α, we have $p_j^\beta = \frac{p_j^{'} - p_j^\alpha}{1 - p_j^\alpha}$ while applying the equation (4). Applying $p_j^\beta = \beta \times j$ and $p_j^\alpha = \alpha \times j$, we modify

$$\beta = \frac{p_j^{'} - \alpha j}{j(1 - \alpha j)}, \text{ if } p_j^{'} > \alpha \times j. \quad (5)$$

If $p_j^{'}$ is less than p_j^α, we still have β=0 and alter α=0.35×10^{-4} to $\alpha = p_j^{'}$/j. Certainly, if β>0, the malicious corruption happens in owner's data.

To analyze the minimum value of *t*, we assume that each block of 4k bytes must be verified every 2.8 months and 28.3 months while *p*=0.0001 and *p*=0.001 respectively once α achieves

to 0.35×10^{-4}. With the increase of each block's length, p_j also varies. Given a block B of the length |B|, which is an integer multiple of 4k bytes. In other words, the block B is composed of n_b blocks each with the length 4k bytes, i.e., n_b=|B|/4k. Thus, the cumulative probability of B's natural corruption after j months, $p_j^{\alpha}(B)$, is

$$p_j^{\alpha}(B)=1-(1-p_j^{\alpha})^{n_b} \tag{6}$$

Once $p_j^{\alpha}(B)$ achieves to p, the block B must be verified. Under the natural corruption, the duration of the block B corruption, $j(B)$, is

$$j(B)=(1-\sqrt[n_b]{1-p})/\alpha \tag{7}$$

months. For example, applying the equation (7), each block must be verified once after $j(B)$ months as shown in **Table 1**. The lower the threshold p is, the shorter the duration of each block corruption is. Similarly, the longer the length of each block is, the shorter the duration of each block corruption is.

Table 1. The relationship among data corruption, the block length, and p while α =0.35×10^{-4}

Block length	j(B) (months)	
	p=0.0001	p=0.001
4kB	2.8	28.3
8kB	1.4	14.2
16kB	0.7	7.1

We vary the values of α in **Table 2**. The results show that the smaller the value of α is, the longer the duration of each block corruption is.

Table 2. The relationship among data corruption, the block length, and α while p=0.0001

Block length	j(B) (months)		
	α =0.35×10-3	0.35×10-4	0.35×10-5
4kB	0.28	2.8	28.3
8kB	0.14	1.4	14.3
16kB	0.07	0.7	7.1

Similarly, we further analyze the file F_i which is divided into n blocks. Referring to the equation (6), the cumulative probability of the file F_i corrupted after j months is $p_j(F_j)=1-(1-(1-p_j)^N))^n$. If $p_j(B)$ achieves to p after j months, $p_j(F_j)$ is equal to

$$p_j(F_j)=1-(1-p)^n, \tag{8}$$

where j also meets the equation (7). Thus, the end time of the vth round, T_v, is

$$T_v=v\times j(B),\ v\in[1,t]. \tag{9}$$

Let v=1, T_1=$j(B)$, where T_1 indicates the interval of every round. We have

$$t=\lceil T/T_1\rceil. \tag{10}$$

To further analyze the number of rounds t in the period T, we choose a series of different length files with each block of the fixed length 4k bytes. Let p be equal to 0.0001. Applying the equations (7), (9) and (10), we get the different values of t in **Table 3** while T is set to a series of values. A longer T corresponds to a larger t. The relationship between T and t has nothing to do with the file length since the file corruption depends on the cumulative probability of every block's natural corruption. Thus, the interval of the verification between two continuous rounds must be adjusted according to the period T.

Table 3. The variety of t

T (years)	T_1(months)	t(rounds)
1	2.8	4
4	2.8	17
8	2.8	34

Assume there are 10,000 files stored in a server, and the average length of each file is 4MB. In [4], the server takes 591.1 milliseconds to verify one file. Thus, the value of t_e for all files at a round is approximately 1.64 hours without utilizing the parallel computation. To identify the corrupted blocks as early as possible, the number of rounds for the block B is adjusted according to the block's length and p in the period T by

$$t = \lceil T / j(B) \rceil. \tag{11}$$

The number of sub-metadata. The number of sub-metadata is closely related to the number of rounds. The verification isn't always executed for every file at each round since not all of the files are corrupted. Each FoM is composed of t sub-metadata, where the value of t should be optimized.

Considering $p_j^{\alpha}(B)$ achieves to p after j months. Applying the equation (7), $p_j(F_i)$ is equal to $1-(1-p)^n$ after j months. Assume it is enough for x sub-metadata in each FoM to verify the corresponding file t rounds, where x is an integer between 1 and t, i.e., $1 \le x \le t$. Hence, each FoM at least occupies $x*m+\log_2 g$ bits. Since the probability of block corruption in a file is uniform, it is enough to identify the corrupted file with the probability of 0.95 while

$$1.96\sqrt{tP(1-P)} - tP < x < 1.96\sqrt{tP(1-P)} + tP \tag{12}$$

and $1 \le x \le t$, where P is $P_j(F_i)$. The proof of the equation (12) is shown as follows.

Let X_i^v, $v \in [1,t]$, be the state of the ith file, F_i, at the vth round. If F_i is intact at the vth round, $X_i^v = 0$, otherwise, $X_i^v = 1$. Hence, $x = \sum_{v=1}^{t} X_i^v$ meets the binomial distribution $B(t,P)$. To ensure x being close to the amount of corrupted files by the probability of 0.95, we have the following equation using Chebyshev's inequality by

$$\Pr\{\left|\frac{x}{t} - P\right| < \varepsilon\} = 0.95, \tag{13}$$

while a positive number ε exists. In probability theory, $\frac{x-tP}{\sqrt{tP(1-P)}}$ is approximately subject to a normal distribution by the de Moivre-Laplace theorem, and then we have $\Pr\{\left|\frac{x-tP}{\sqrt{tP(1-P)}}\right| \le \frac{t\varepsilon}{\sqrt{tP(1-P)}}\} = 0.95$. Simplifying it, we have $2\phi(\frac{t\varepsilon}{\sqrt{tP(1-P)}} - 1) = 0.95$, i.e, $\phi(\frac{t\varepsilon}{\sqrt{tP(1-P)}}) = 0.975$. Due to $\phi(1.96) = 0.975$, we have $\varepsilon = 1.96\sqrt{P(1-P)}/\sqrt{t}$. Applying it into the equation (13), we have $\Pr\{\left|\frac{x}{t} - P\right| < \frac{1.96\sqrt{P(1-P)}}{\sqrt{t}}\} = 0.95$. Simplifying it, we have $\Pr\{1.96\pi - tP < x < 1.96\pi + tP\} = 0.95$, where $\pi = \sqrt{tP(1-P)}$. Thus, we have the equation (12).

Moreover, the metadata updating and deleting is also important. If any block in a file is updated or deleted, both the FoM and the BoM for the file must be updated or deleted. Before the block is updated or deleted, the file is first downloaded from the server to the local. For the block updating, the metadata of blocks and files are recomputed by the owner while the block is updated at the server, then the signed BoM are updated at the server and the FoM are replaced at the owner. For the block deleting, the FoM are recomputed and replaced at the owner. Simultaneously, the block and its BoM are deleted respectively at the server.

5. Verification protocol

5.1 File-oriented verification

Referring to the verification model, the owner firstly sets the appropriate initial parameters such as *m* and *t* respectively by applying the equation (3) and (11), and then computes the local metadata via choosing appropriate hash functions and random codes to generate a local FoM for each file. These FoM are stored in the local space.

Secondly, the auditor requests the local metadata of the distrusted files from the owner, and simultaneously sends the challenge information about these files (i.e., hash functions and random codes) to the server. In this case, *k* hash functions are chosen from *tk* ones in each verification. The suspicious files in the suspicious data verification need to be selected before they are challenged. These files are chosen according to the cumulative probability of the corrupted files over a period of time. For example, once $p_j(F_i)$ achieves $1-(1-p)^n$ applying the equations (4), (7) and (8) after j months, the file F_i is selected as the suspicious file.

Thirdly, the server recomputes each remote FoM of these files as the proof by the uniform hash functions, codes and initial parameters similar to the owner.

Finally, the auditor compares pairs of metadata (i.e., the local metadata and the remote metadata) and then judges whether these files are intact. For example, applying the querying operation of the Bloom filter, after *k* hash functions and random codes are chosen, if *k* array positions of a file's local metadata and remote metadata are all *1*, the file is determined to be not corrupted.

5.2 Mixed verification protocol description

Even if the file-oriented verification is superior to the block-oriented verification on the

verification cost as shown in section 7, it also has several defects. First, it results in the potential verification misjudgment due to the false positive of the BF. Second, it only identifies the corrupted files instead of the corrupted blocks. Even if a file is corrupted, not all blocks in the file are corrupted. Considering the efficiency of the file-oriented verification and the accuracy of the block-oriented verification, we propose a mixed verification protocol to deal with the data corruption in massive files.

Definition 2 (The suspicious data verification). In process of data service, the auditor suspects the integrity of some small files, and then challenges them. The auditor extracts the suspicious files according to the cumulative probability of the file corrupted over a period of time.

Definition 3 (The corruption locating verification). If the suspicious data verification fails, the auditor identifies the specific corrupted blocks in these suspicious files. On the basis of suspicious files, the auditor further identifies the corrupted blocks in these files.

Based on the above consideration, we design a mixed verification protocol to rapidly locate the corrupted blocks in massive files and reduce the verification cost. We abbreviate the block-oriented verification, the file-oriented verification, and the mixed verification to BLOCK, FILE, and MIXED respectively.

Before the MIXED executes, the application condition of the FILE should be determined at first. Even if the FILE is oriented to each file, it cannot be utilized to scan all files in the remote storage due to the verification cost. The application of the FILE is affected by two factors. One is the importance of files. The other is the size of files. The former is classified by the value of data. In the field of archival science, it has strict criteria to divide the value of files, which mainly includes the content of document and the source of document. The owner can accurately distinguish her files referring to the correlative criteria. Due to the space limitations, we do not discuss it in detail here. The latter is limited by the similarity of small files. For small files, many similarity measurement methods are proposed to evaluate whether any file belongs to the small file. Also, we do not discuss it here.

To perform the MIXED, we expand the verification model in Section 3 from four phases to seven phases. Specifically, the local FoM and BoM are generated in the first phase, and then the FoM are stored at the owner and the BoM are stored into the server. In the second phase, the users request the owner to send the local FoM corresponding to some distrustful accessed files, and then challenge them to the server. Here, these users are sent the local FoM simultaneously by the owners once the number of users meets the requirement that the verification results can be trusted. In the third phase, the server recomputes and returns the remote FoM to the users. In the fourth phase, the users compare the pairs of metadata to judge the distrustful files. Only if the certain number of users provides the same judgment on the same files, the verification results can be recognized. If the suspicious data verification fails, it executes the corruption locating verification in the three following phases. In the fifth phase, the user acts as an auditor to send $chal<w, F_{ij}.id>$ of all blocks in these suspicious files to the server. In the sixth phase, the server computes and returns ϑ' as proof to the auditor. In the final phase, the auditor checks $e(\vartheta', g_2) \stackrel{?}{=} e(\sigma, w)$ and then identifies the corrupted blocks in these suspicious files. For example, there exists one corrupted block in *g* files. After *g* files are checked by the FILE, one distrustful file *i* is detected. Then, applying the BLOCK, the corrupted block *j* is identified as shown in **Fig. 4**.

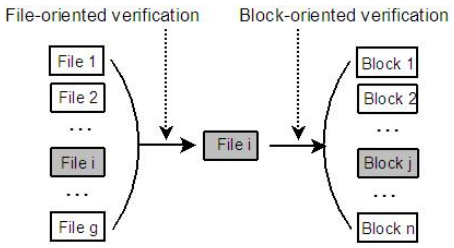


Fig. 4. The corrupted block is identified by the mixed verification.

6. Security analysis

The security of the algorithm includes the verification correctness, metadata unforgeability and non-replay, and data security. Since the security properties of the homomorphic-based algorithms (i.e., correctness, unforgeability, etc.) have been discussed in many researches, we don't discuss them in this paper. Any algorithm must correctly verify the data integrity. The attacks with regard to metadata are the metadata forging and the metadata replaying.

6.1 The verification correctness

The verification correctness is of crucial importance no matter which verification method is utilized, i.e., the FILE and the BLOCK.

Theorem 1: Only if a corrupted file is identified by the FILE, the BLOCK at least detects one corrupted block in the file, and vice versa.

Proof: A file's integrity is the cumulative integrity of all blocks in the file. If a file is judged to be corrupted after it is checked by the FILE, it must have at least one corrupted block referring to the description of the FILE. All pairs of the BoM aren't synchronously equal while they are verified by the BLOCK. Even if the number of corrupted blocks is ascertained, at least one corrupted block exists in the corrupted file. In other words, at least a pair of the BoM is unequal once after all pairs of the BoM for every block in the corrupted file are compared.

Conversely, if one block in a file is corrupted, the FILE must find out that a pair of the FoM is unequal to each other by the probability of 1-f . However, the probability can be limited in a small range. On the one hand, we can set a smaller f to improve the accuracy of the data verification. Assume the probability of each file misjudged is ξ . Referring to the probability of false positives of the BF f, let $\xi = f$. Applying the equation (3), we know that the accuracy of the data verification approximately achieves to 100%, i.e., $\xi = 0$, while f is close to 0. On the other hand, we increase the frequency of each file being verified during the period T to improve the correctness of the file verification. Even if the BLOCK provides the correct result of the verification for each block, it only verifies a small part of all data due to the limited computation overhead. Assume the frequency of each same file being repeatedly verified during the period T is t_0, which is equivalent to the number of hash functions increasing from k to t_0k in the BF. Correspondingly, the probability ξ drops down from f to $f^{e^{(t_0-1)}k}$ applying the equation (3). For example, ξ drops down from 1.0×10^{-3} to 1.6×10^{-164} while f =0.001 and t_0 increases from 1 to 2. The probability is very low and approximately achieves to 0. Without a doubt, we can compensate the lack of the FILE based on the BF about the verification correctness of each file via decreasing f and increasing the frequency of each file being verified. Therefore, the FILE also ensures the verification correctness of checked files.

6.2 The metadata unforgeability

The metadata generated by the owner is an important criterion in the verification, and cannot be forged by the adversary. For the BLOCK, as the metadata unforgeability has been proved in [5] and [11], we don't discuss it here. We only analyze the unforgeability of the FoM.

Theorem 2: The forged FoM cannot pass the verification if not all pairs of k sub-metadata (i.e., the local and remote file-oriented sub-metadata) are equal.

Proof: Each file-oriented sub-metadata is the value of an assay position in the BF with the length of m bits. The remote FoM are confronted with the potential spoofing attacks from the malicious server. As the server is unfeasible to guess all of k local sub-metadata, it can only forge a part of k remote sub-metadata randomly. Assume the server guesses ρ ones of k remote sub-metadata. The probability of each sub-metadata being forged is ρ / k. Only if

$$\rho < k(1-\sqrt[k]{1-f}), \tag{14}$$

the forged remote metadata can escape from being detected. The value of ρ is a very small value. The remote metadata forged by the server either escape from the corrupted data being detected or save too much computational resources. Thus, the forged remote metadata are infeasible while the verification is repeatedly executed.

6.3 The metadata non-replay

Similar to the previous reason, we only analyze the non-replay of FoM here.

Theorem 3: Even if a server stores the remote FoM at last round, it cannot pass the same files' verification utilizing the metadata relaying at the next round.

Proof: The attack of metadata replaying is from the server who stores the recomputed remote FoM. However, all the local metadata aren't utilized repeatedly at each different round. Certainly, the potential risk of the metadata replaying exists after all the local sub-metadata are run out after t rounds. However, the verification computation time and fee will restrict the user to unrestrainedly verify the accessed data in the server. Also, the ratio of data corruption is very low. Moreover, it is nonsignificant for the server to store these remote metadata while t meets the equations (10) and (12). Thus, the replaying attack of the remote FoM is infeasible.

7. Simulation

To further evaluate our algorithms, we test their performance with several metrics. In the experiment, each file or block is tested only once at each round. Let the amount of files be 200,000. To shorten the time of the experiment, we let T be 2 years. Applying the equation (3), the size of the BF is 16 bits and the number of hash functions is 4 while f =0.001. Let α =0.35 $\times 10^{-4}$ and β =0. Assume it is dangerous to the security of data while the threshold p achieves to 0.0001. Let the length of each block be 8kB. Applying the equation (8), $P_j(F_i)$ achieves to 0.05, 0.10,.and 0.19 while the file is 4MB, 8MB, and 16MB respectively. The owner and auditor implement these algorithms on a notebook computer equipped with the Intel Core Duo 1.6GHz CPU, 1.5 GB memory. The server executes the verification on a Dell PowerEdge R420 (Intel Xeon E5-2403 1.8GHz CPU, 8GB memory). HMAC-SHA1 is chosen as the hash function. All experiment results are tested 50 times and are taken their average.

7.1 Metadata computation time

Each sub-metadata. The computation time of each sub-metadata is the duration of a file mapping into the BF with m bits. We arrange the length of each file to be 1MB, 4MB, and 8MB respectively. The time is tested as shown in **Table 4**. It shows that a smaller file costs shorter time.

Table 4. The computation time of each sub-metadata

File length (bytes)	Time (ms)
1M	30
4M	124
8M	247

Each metadata. The time of each metadata generation depends on the length of each data unit and the number of rounds. For the BLOCK, a data unit is a block. For the FILE, a data unit is an entire file. For the MIXED, a data unit includes an entire file and all blocks in the file. To decrease the experiment time, we let the size of each file be 4MB, which is divided into 512 blocks. Applying the equations (7) and (11), t is 9 rounds. According to the equation (12), x should be set between 3 and 9. The time of each metadata generation for the BLOCK and FILE is respectively shown in the left of **Table 5**. As one FoM and one BoM can't constitute a MIXED metadata, the MIXED isn't listed. The time of the FoM is remarkably greater than the BoM since the BoM only computes one block instead of the entire file.

Table 5. The time of each metadata generation

Algorithms	Time(ms)	
	A data unit	A entire file
BLOCK	71	2,809
FILE	1,046	1,046
MIXED		3,855

For the sake of fairness, we compare the computation time of different metadata generation for an entire file. Based on the conditions above, the amount of the FoM is 1. The amount of the BoM is 512. The amount of the MIXED metadata includes 1 FoM and 512 BoM. Each FoM at least includes three sub-metadata while x=3. The time of three metadata generation is shown in the right of **Table 5**. The time of the BoM is remarkably longer than FoM. Certainly, the MIXED costs the most time since it computes 1 FoM as well as 512 BoM. Obviously, with the increase of the number of files, the difference will become more and more great. However, all these metadata are only precomputed at the owner.

7.2 Storage cost

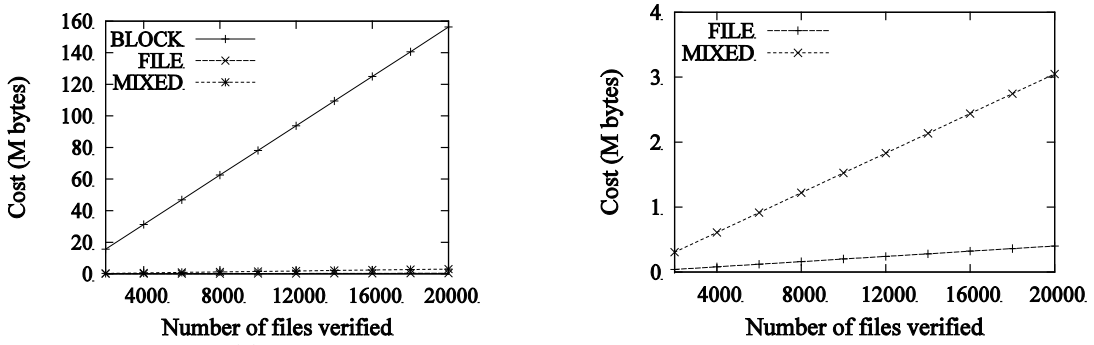
From the previous description, we know that the storage space of the MIXED metadata is $(gxm+\log_2 g)$ bits more than the BoM since it costs the additional storage space of the FoM at the owner. The server's storage space is equivalent with regard to the storage of the MIXED metadata and the storage of the BoM.

Let the length of each file be 4M bytes, and the number of files be 200,000. The amount of the FoM is 200,000, which occupies 4.5MB (x=3) to 6.8MB (x=9) applying the equations (2), (7), (8), (10) and (12). The amount of the BoM is 102,400,000, which occupies 1,562MB while the length of each BoM is 128 bits. For the MIXED, the amount of the metadata includes 200,000 FoM of length 4.5~6.8MB and 102,400,000 BoM of length 1,562MB. Therefore, the

storage space of the MIXED metadata is only approximately 0.43% more than the BoM.

7.3 Transmission cost

The transmission cost includes the overhead of the auditor sending the request of verification and the owner or the server sending the response. If there are no bad blocks in files, the cost is shown in **Fig. 5(a)**. The curve of the MIXED coincides with that of the FILE, and they are both lower than the BLOCK. The reason is that the MIXED doesn't execute the BLOCK while no bad block is identified by the FILE. Otherwise, the cost is shown in **Fig. 5(b)** while there exists a corrupted file every 59 files (only one corrupted block in each corrupted file). For the FILE, the costs are both equal in two figures. The MIXED is greater than the FILE, and but they are both far lower than the BLOCK since the MIXED only executes the BLOCK for a part of files each with a bad block after those corrupted files are identified by the FILE.



(a) No bad block (b) One bad block

Fig. 5. The transmission cost in the verification.

7.4 Verification execution time

The verification execution time includes the remote metadata recomputed and compared with the local metadata at one round. Similarly, let each file be 4MB. **Table 7** shows the time of three algorithms with no bad block or one bad block. No bad block means all blocks in the file are intact. One bad block means there exits a corrupted block in the file. We corrupt a block in the file at random. The BLOCK costs further more time than the FILE and the MIXED while there exists no bad block, and but the MIXED costs more time than the BLOCK while there exists one bad block. The reason is that the MIXED (no bad block) only executes the FILE (the suspicious data verification).

Table 7. The verification execution time of a entire file

Algorithms	Time(ms)	
	No bad block	One bad block
BLOCK	873	873
FILE	125	125
MIXED	125	998

7.5 Time of corrupted data being identified

The time of corrupted data being identified is the duration of all corrupted blocks identified. Let the file be 4MB. Applying the equation (8), $P_j(F_i)$=0.05 while p=0.0001. The cumulative

probability of n' files' natural corruption is $P(n' F_i)=1-(1- P_j(F_i))^{n'}$. While $P(n' F_i)$ achieves to 0.95, n' is approximately equal to 59, i.e., there is a corrupted file every 59 files by the probability of 0.95. We vary the number of files from 1 to 200,000. Hence, there are 3,390 files corrupted in 200,000 files by the probability of 0.95. The verification time of the MIXED and the FILE are close and both approximately 1/3 times less than that of the BLOCK in **Fig. 6**. If the verification is parallelly executed on multiprocessors or multidevices, the efficiency of the MIXED is more obvious.

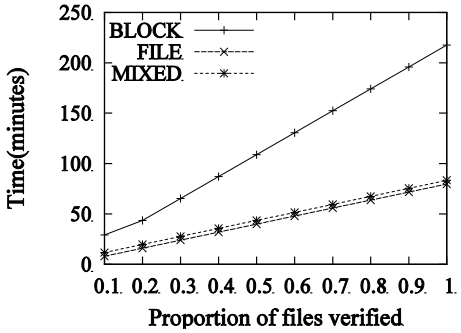


Fig. 6. The verification execution time.

7.6 Coverage of data verified

The coverage of data verified is the ratio of the verified data to the total data. Referring to Amazon EC2 pricing, the charge is based on the utilization of each server's computing unit within one hour. We select a machine as a computing unit with the time limited within one hour. The coverage is compared between the MIXED with no bad block and the BLOCK. While the MIXED executes the verification of 15,063 files (i.e., the size of 60,251MB), the BLOCK only covers 5,513 files (i.e., the size of 22,052MB). Thus, the MIXED provides all data with a greater opportunity to be checked at each round.

In other ways, however , we restrict the verification cost at each round. The duration that the BLOCK verifies a file can support the MIXED to execute the same file 3 times. It also implies that the value of k is increased from 4 to 12 at each round. In this case, the probability ξ is reduced from 2.0×10^{-6} to 3.3×10^{-16} applying the equation (3). The more the frequency of data verified is the lower the probability ξ is. Certainly, the MIXED improves all data's security by expending the coverage of data verified with less computation time.

5. Conclusion

In this paper, to identify the corrupted data in cloud data storage, we improve the file-oriented metadata generation and verification, and propose a mixed verification protocol to balance the trade-off between the verification cost and the efficiency. By the mixed verification, we can rapidly locate all corrupted data in the verified data. In the file-oriented verification, we design the structure of the file-oriented metadata based on the BF for each file and analyze the corresponding parameters in each metadata to resist the malicious attacks. In the mixed verification, we utilize the file-oriented verification to extend the coverage of checked data or decrease the verification charge by means of less cost of file-oriented metadata computation and transmission before the traditional block-oriented verification executes. The theoretic analysis and simulation results demonstrate that our protocol fastly identifies the corrupted

blocks at the low cost in massive small files. Even if the little storage space of file-oriented metadata is increased, it doesn't burden the server or the auditor since it is only stored at the owner. In the future, we will further optimize the metadata generation and the algorithm's performance.

References

[1] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, J. Schindler, "An analysis of latent sector errors in disk drives," in *Proc. of SIGMETRICS'07*, pp. 289–300, 2007. Article (CrossRef Link)

[2] K. Yang, X. Jia, "An Efficient and Secure Dynamic Auditing Protocol for Data Storage in Cloud Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 9, pp. 1717–1726, 2013. Article (CrossRef Link)

[3] Y. Deswarte, J.-J. Quisquater, A. Saidane, "Remote Integrity Checking," in *Proc. of the 6th Working Conference on Integrity and Internal Control in Information Systems*, pp. 1–11, 2004. Article (CrossRef Link)

[4] F. Seb´e, J. Domingo-Ferrer, A. Martinez-Balleste, Y. Deswarte, J.-J. Quisquater, "Efficient Remote Data Possession Checking in Critical Information Infrastructures," *IEEE Transactions on Knowledge and Data Engineering*, vol.20, no. 8, pp. 1034–1038, 2008. Article (CrossRef Link)

[5] Z. Hao, S. Zhong, N. Yu, "A Privacy-Preserving Remote Data Integrity Checking Protocol with Data Dynamics and Public Verifiability," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 9, pp. 1432–1437, 2011. Article (CrossRef Link)

[6] A. Juels, B. S. Kaliski,, "Pors: proofs of retrievability for large files," in *Proc. of the 14th ACM conference on Computer and communications security*, pp. 584–597, 2007. Article (CrossRef Link)

[7] T. Aditya, P. Baruah, R. Mukkamala, "Space-efficient Bloom Filters for Enforcing Integrity of Outsourced Data in Cloud Environments," in *Proc. of 2011 IEEE 4th International Conference on Cloud Computing*, pp. 292–299, 2011. Article (CrossRef Link)

[8] J. Ni, Y. Yu, Y. Mu, Q. Xia, "On the Security of an Efficient Dynamic Auditing Protocol in Cloud Storage," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no.10, 2760–2761, 2014. Article (CrossRef Link)

[9] Y. Yu, L. Niu, G. Yang, Y. Mu, W. Susilo, "On the security of auditing mechanisms for secure cloud storage," *Future Generation Computer Systems*, vol. 30, pp. 127–132, 2014. Article (CrossRef Link)

[10] C. Liu, C. Yang, X. Zhang, J. Chen, "External integrity verification for outsourced big data in cloud and IoT: A big picture," *Future Generation Computer Systems*, vol. 49, pp. 58–67, 2015. Article (CrossRef Link)

[11] B. Wang, B. Li, H. Li, "Oruta: Privacy-Preserving Public Auditingfor Shared Data in the Cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, pp. 43– 56, 2014. Article (CrossRef Link)

[12] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, D. Song, "Provable data possession at untrusted stores," in *Proc. of the 14th ACM conference on Computer and communications security*, pp. 598–609, 2007. Article (CrossRef Link)

[13] C. Wang, Q. Wang, K. Ren, W. Lou, "Privacy- Preserving Public Auditing for Data Storage Security in Cloud Computing," in *Proc. of IEEE INFOCOM*, 2010. Article (CrossRef Link)

[14] C. Wang, S. S. Chow, Q. Wang, K. Ren, W. Lou, "Privacy-Preserving Public Auditing for Secure Cloud Storage," *IEEE Transactions on Computers*, vol. 62, no. 2, pp. 362–375, 2013. Article (CrossRef Link)

[15] Y. Zhu, G.-J. Ahn, H. Hu, S. S. Yau, H. G. An, C.-J. Hu, "Dynamic Audit Services for Outsourced Storages in Clouds," *IEEE Transactions on Services Computing*, vol. 6, no. 2, pp. 227–238, 2013. Article (CrossRef Link)

[16] K. Yang, X. Jia, "An Efficient and Secure Dynamic Auditing Protocol for Data Storage in Cloud Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 9, pp. 1717–1726, 2013. Article (CrossRef Link)

[17] M.-S. Hwang, C.-C. Lee, T.-H. Sun, "Data error locations reported by public auditing in cloud storage service," *Automated Software Engineering*, vol. 21, no. 3, pp. 373–390, 2014. Article (CrossRef Link)

[18] P. Williams, R. Sion, B. Carbunar, "Building castles out of mud: Practical access pattern privacy and correctness on untrusted storage," in *Proc. of the 15th ACM conference on Computer and communications security (CCS '08)*, pp. 139–148, 2008. Article (CrossRef Link)

[19] E. Stefanov, M. van Dijk, A. Oprea, A. Juels, "Iris: A Scalable Cloud File System with Efficient Integrity Checks," *Annual Computer Security Applications Conference*, pp. 229–238, 2012. Article (CrossRef Link)

[20] H. Wang, Y. Zhang, "On the Knowledge Soundness of a Cooperative Provable Data Possession Scheme in Multicloud Storage," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, pp. 264–267, 2014. Article (CrossRef Link)

[21] S. Ghemawat, H. Gobioff and S.-T. Leung, "The Google File System," in *Proc. of 19th ACM Symposium on Operating Systems Principles*, 2003. Article (CrossRef Link)

[22] H. Wang, "Identity-Based Distributed Provable Data Possession in Multi-Cloud Storage," *IEEE Transactions on Services Computing*, no. 99, pp. 1-12, 2014. Article (CrossRef Link)

[23] W. Jiang, V.K. Prasanna, "Scalable Packet Classification on FPGA," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 20, no. 9, pp. 1668-1680, 2012. Article (CrossRef Link)

[24] G. Ateniese, R. D. Pietro, L. V. Mancini, G. Tsudik, "Scalable and Efficient Provable Data Possession," in *Proc. of the 4th international conference on Security and privacy in communication networks*, 2008. Article (CrossRef Link)

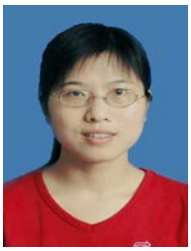
[25] L. N. Bairavasundaram, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, G. R. Goodson, B. Schroeder, "An analysis of data corruption in the storage stack," *ACM Transactions on Storage*, vol. 4, no. 3, pp. 8:1–8:28 , 2008. Article (CrossRef Link)

[26] G. Xu, M. Yu, J. Cheng, L. Li, Y. Shi, "Data Sampling Algorithms for Data Integrity Verification in Cloud Storage," *International Journal of Advancements in Computing Technology*, vol. 5, no. 9, pp. 1026–1034, 2013. Article (CrossRef Link)

Guangwei Xu is an associate professor in the School of Computer Science and Technology at Donghua University, Shanghai, China. He received the M.S. degree from Nanjing University, Nanjing, China, in 2000, and the Ph.D. from Tongji University, Shanghai, China, in 2003. His research interests include the verification of data integrity, data security, QoS and routing of the wireless Ad Hoc networks and sensor networks.

Yanbin Yang is a graduate in the School of Computer Science and Technology at Donghua University, Shanghai, China. His research interests include the verification of data integrity and data security.

Cairong Yan is an associate professor in the School of Computer Science and Technology at Donghua University, Shanghai, China. She received her Ph.D in Computer Science from Xian Jiaotong University of China in 2006. Her research interests are parallel and distributed computing, cloud computing, and big data processing.



Yanglan Gan is an associate professor in the school of computer science and technology at Donghua University, Shanghai, China. She received Ph.D. degree in computer science from Tongji University in 2012. Her research interests include Bioinformatics, and data mining.