Mobile Networks and Applications (2021) 26:842–850
https://doi.org/10.1007/s11036-019-01328-1

A Multi-grained Log Auditing Scheme for Cloud Data Confidentiality

Zhen Yang[1,2] · Wenyu Wang[3] · Yongfeng Huang[1,2] · Xing Li[1,2]

Published online: 14 August 2019

Abstract

With increasing number of cloud data leakage accidents exposed, outsourced data control becomes a more and more serious concern of their owner. To relieve the concern of these cloud users, reliable logging schemes are widely used to generate proof for data confidentiality auditing. However, high frequency operation and fine operation granularity on cloud data both result in a considerably large volume of operation logs, which burdens communication and computation in log auditing. This paper proposes a multi-grained log auditing scheme to make logs volume smaller and log auditing more efficient. We design a logging mechanism to support multi-grained data access with Merkle Hash Tree structure. Based on multi-grained log, we present a log auditing approach to achieve data confidentiality auditing and leakage investigation by making an Access List. Experiments results indicate that our scheme obtains about 54% log volume and 60% auditing time of fine-grained log auditing scheme in our scenario.

Keywords Multi-grained log · Log auditing · Log forensics · Cloud data confidentiality

1 Introduction

Cloud computing is a quite prosperous application area in recent years. Specifically, cloud storage is widely-appreciated for offering reliable and convenient data outsourcing services. Nevertheless, data leakage accidents are reported to increase with the ever-growing popularity of cloud storage. Although many researches enhanced cloud data security with encryption schemes [5], serious threats are still noticeable in cloud storage: 1) Data access control is transferred to Cloud Storage Provider (CSP) along with data outsourcing. 2) CSP faces open Internet environment where malicious attacks and unexpected system failures exist. These two factors both exacerbate user's worry about his outsourcing cloud data.

Users' worry on cloud data can be relieved by using cloud data security auditing. An auditor with expertise and special capability will check the cloud data security status for the user who outsourced the data [15]. For cloud data integrity and availability, auditor enables public auditing with data hash checking and fetching [17]. On the other hand, given the confidentiality of the cloud data is violated, there must be some unauthorized user who has accessed the cloud data. With the unauthorized access operation log, the cloud data confidentiality can be audited [11]. Furthermore, cloud data confidentiality auditing enables user accountability by investigating the responsible user who conducts the unauthorized access [10].

Generally, log based data confidentiality auditing can be divided into two stages: log integrity auditing and log operation legality auditing [13]. Since log integrity is precondition of log operation legality auditing, many reliable logging mechanisms have been proposed to ensure log integrity. These logging mechanisms universally adopted hash chain structure to make logs forward-security [9] and append-only properties [6]. As a result, each entry of log is tamper-resilient and resists malicious deletion. After log integrity is assured, log operation legality auditing need to be conducted to investigate data confidentiality status. Log legality auditing analyzes log content to detect illegal operations, which has similar process to DDoS attacks detections [3].

In the cloud, data is commonly stored as blocks, and data access can be multi-grained from file level to block level to support different operation demands. To reflect fine-grained cloud data access operations, log is recorded for every data block in a fine granularity [8]. Thus, massive

✉ Zhen Yang
eeyangzhen@gmail.com

1 Department of Electronic Engineering, Tsinghua University, Beijing 100084, China

2 Tsinghua National Laboratory for Information Science and Technology, Beijing 100084, China

3 Information Networking Institute, Carnegie Mellon University, Pittsburgh, PA, 15213, USA

cloud users make frequent multi-grained access on cloud data, resulting in massive fine-grained logs. The volume of the fine-grained logs becomes too large to efficiently audit. Some methods has been proposed to speed up the log integrity auditing process, including supporting selective verification [13] and using Rabin's fingerprint and bloom filter for every verification process [1]. However, there are still no efficiency improving method for log operation legality auditing.

In this paper, we propose a file-oriented multi-grained log auditing scheme to solve the problem aforementioned. We firstly design a multi-grained data operation log based on Merkle Hash Tree (MHT) structure [7], and use the state-of-the-art of reliable logging mechanisms to ensure reliable log auditing. Besides, to support multi-grained auditing on logs, we propose a refined AccessList structure together with two auditing algorithms to investigate data leakage with accuracy. In our scheme, the multi-grained logs and the multi-grained auditing target coordinate to achieve high efficiency in log auditing. Particularly, our contribution in this work can be summarized as the following three aspects:

1. We propose a log auditing scheme supporting multiple data granularity, which can hold data confidentiality auditing task on single data block, multiple data blocks and the whole file.
2. We build a data operation log structure which enables multi-grained data operation and improves information validity of logs.
3. We improve the auditing scheme efficiency remarkably, compared with auditing schemes based on the state-of-the-art fine-grained logging.

The rest of the paper is organized as follows: Section 2 overviews the related work. Then we introduce the system and problem statement in Section 3, followed by the detailed description of our scheme in Section 4. Section 5 gives the performance evaluation. Finally, Section 6 gives the concluding remark of the whole paper.

2 Related work

Recently, cloud data security has attracted increasing attention of researchers. Some research works [14, 17] focused on the threat of cloud data integrity such as data loss, and put forward and completed cloud data integrity auditing scheme. These works promoted auditing approaches to be applied on logs: using logs auditing to monitor data confidentiality status.

As we state before, log auditing can be divided into two steps: reliable logging which protects and verifies log integrity, and logs forensics which audits legality of log operations.

2.1 Reliable logging

To protect log integrity, forward-security property was proposed to prevent attacker from tampering logs generated before attack happens. The common methods to implement forward-security include hash chain and digital signature [9]. While tampering on logs was prevented by forward-security, malicious deletion attack was still a problem for logging. In 2009, append-only property was proposed to protect log entry from any change after its generation [6]. They firstly presented FssAgg authentication technique to achieve it. Later, PKC-based secure logging called Log-FAS [19] achieved both forward-security and append-only properties, and improved log integrity auditing efficiency. CloudProof system [8] designed a reliable log named attestation with both forward-security and append-only properties. Recently, a log block structure [4] was proposed to store logs into blockchain. Thus, the immutability of blockchain can ensure the logs integrity.

However, logging scheme are still faced with incorrect log generation attacks of malicious intrusion. Automated logging mechanism was proposed firstly by leveraging java JAR file's programmable capability [11]. Then an extension work [10] implements JAR to enclose automated logging mechanism and builds a framework of accountability in case of data leakage. JAR-based automated logging is further adapted in CSP and achieves more efficient logging [18].

When cloud service starts, they just offered coarse granularity operation on the file. With developing cloud service, cloud data are accessed in fine granularity of data blocks. Some logging works adopted block oriented log record [8, 10] to enable fine-grained cloud data operation recording. A user often reads or writes multiple data blocks at a time. Such an operation will generate multiple logs if fine-grained logging is implemented. As a result, log volume becomes very huge. Because of huge volume of fine-grained logs, block-based logging approach [13] was proposed to support selective verification and enable the forward security and append-only properties with hash-chain-based structure for each log block. Then, based on the block-based logging [13], BLS signature and random mask technique [2] and binary auditing tree [12] were introduced into log auditing process to ensure privacy-preserving.

On the other hand, Cloud Log Assuring Soundness and Secrecy (CLASS) [1] was presented to secure logging by encrypting logs with individual user's public key. And Rabin's fingerprint and Bloom filter was used to generate proof of past log to speed up the log auditing.

2.2 Log forensics

Digital forensics are important means of information security and accountability. In 2015, digital forensics in cloud has been analyzed to propose an OCF (Open Cloud Forensics) Model and reliable forensics process based on actual civil lawsuit [21]. In an extension work [20], FECloud (Forensics-friendly Cloud) Framework is presented and offers underlying supporting for OCF model.

Until present, researches on log operation legality auditing still depends on CSP's functional support and user defined rules. In practical log auditing, users still face dilemma of complicated log auditing rules and low auditing efficiency.

3 System architecture & problem statement

3.1 System architecture

In the cloud data confidentiality auditing scenario that we concern, there are four participants: Data Owner, User, CSP and auditor. As is shown in Fig. 1, we discuss behaviors and introduce responsibilities of the four participants.

- Data Owner: he outsources his data to the CSP. Then he defines a sharing group and informs CSP of the group name list. If a user is authorized to read the data, he informs the user his authorization information.
- User: he makes every attempt to read data from CSP, no matter he is authorized or not. He may access the data in different granularity, such as one data block or all blocks of the data.
- CSP: CSP is responsible for keeping data and sharing data to user who can prove himself authorized. Ideally, only user in sharing group is allowed to read data from CSP. Data and sharing group list at CSP can be updated by Owner. For every operation such as data reading and writing, CSP makes corresponding logs with reliable logging mechanism.

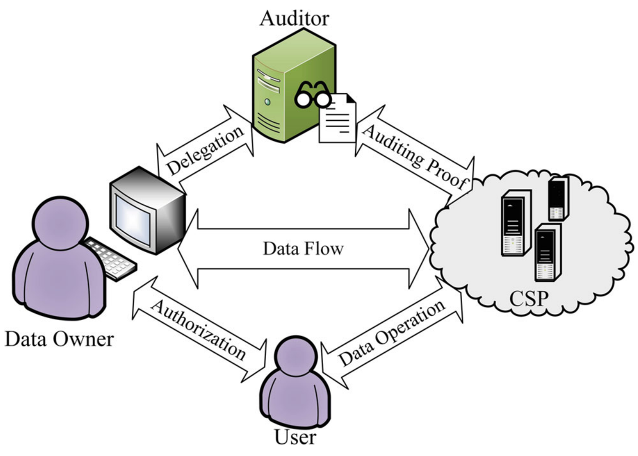

Fig. 1 Entities and behaviors in cloud data confidentiality auditing

- Auditor: Auditor is an entity who has auditing expert capabilities. When he gets auditing delegation from the data owner, he fetches the data operation logs from CSP to audit the data confidentiality.

3.2 Problem statement

Security threats arise when we review the architecture above.

As cloud storage environment is open on the Internet, malicious users may attack to pilfer cloud data. Reliable logs serve as proof for data confidentiality auditing. With reliable logging mechanisms [18], CSP makes logs to record data operation information in detail and these logs can resist tampering and malicious fabricating.

While reliable logs seems to be a practical solution for confidentiality auditing, the volume of logs can be considerably large because of the high frequency of data operations and many log entries generated in one opertion. In cloud data confidentiality auditing where a great number of logs are involved, the auditor should 1) fetch all logs from CSP, which is a heavy burden on communication; 2) perform auditing approach on these logs, which means a huge cost on computing. It is obviously a harsh task for normal auditors whose communication and computing ability fails to meet such demanding requirements.

4 The proposed scheme

This section presents our data confidentiality auditing scheme. We start from a file-oriented log which supports to reflect multiple data operation granularity. Then we show how to enable multi-grained log auditing for data confidentiality.

The scheme can be divided into two phases:

Logging Sharing users access cloud data on CSP by either reading or writing. With reliable logging mechanism, every conducted data access is recorded into authentic reliable logs.

Log auditing Auditor is able to fetch logs from CSP by providing Data Owner's auditing delegation message. Logs are then examined by auditor to check the existence of confidentiality violation.

4.1 File oriented multi-grained log

In order to support multiple granularity in log, we first design a special data field in our log format. With designed logging mechanism, multi-grained data access can be recorded in our format of log.

Operation Type	Data Hash	UserList Version	User HashID	Timestamp	ChainHash	Signature

Fig. 2 File oriented multi-grained log structure

4.1.1 Log format

The log format in our scheme is shown in Fig. 2. The data fields in the log format is explained as below.

Here $OperationType(OT)$ field denotes this log entry's operation, including *READ* and *WRITE*.

$DataHash$ is the operation orienting data's hash value. This field can uniquely identifies the accessed data blocks, no matter the access granularity is. This field is calculated by building a Merkle Hash Tree structure [7, 16] for each cloud file and detailed process will be introduced later in logging.

$UserListVersion(ULV)$ and $UserHashID(UHID)$ is used together to identify the access operation user uniquely. While the data owner generates pseudonym $UserHashID$ for users to preserve their privacy for log auditing, he also uses $UserListVersion$ to publish new version pseudonyms to enhance privacy.

$Timestamp(TS)$ field records the time of this operation.

$ChainHash$ is defined in next equation:

$$ChainHash_i = hash([OT|DH|ULV|UHID|TS]_i, ChainHash_{i-1}) \quad (1)$$

Here we use current log entry's information from $OperationType$ field to $Timestamp$ field and $ChainHash$ of last log entry to calculate the $ChainHash$ of current log entry. With $ChainHash$ enabling log entries' chain order, logs are protected from malicious deleting attacks.

$$Sig = sign([OT|DH|ULV|UHID|TS|ChainHash]_i, sk_{log}) \quad (2)$$

The $Signature(Sig)$ field is signed by log generator on current log entry's information from $OperationType$ field to $ChainHash$ field, which ensures the integrity of this log entry.

4.1.2 Multi-grained logging for data operations

In Fig. 3, we show a Merkle Hash Tree (MHT) of a file consisting of 7 blocks. MHT is a classic data structure to organize data storage, in which every leaf node corresponds to the hash value of a data block. The leaf nodes from left to right map one to one with file data blocks from start to end. Then every two hash values aggregate into the father node's hash value as a binary tree structure. With iterations of this computing, a series of data blocks' hash can be aggregated to one root hash. Commonly, we can uniquely identify a file by building a MHT with block hash.

Here we prove how our multi-grained logging will reduce the number of log entries by exploiting MHT. Generally speaking, it is done by aggregating the operation data blocks. When the log entry records an operation on a single data block, $DataHash$ is just the data block's BlockHash. When the log entry records an operation on the whole file, $DataHash$ is RootHash of the file's MHT. In case the log entry records an operation on multiple data blocks in the

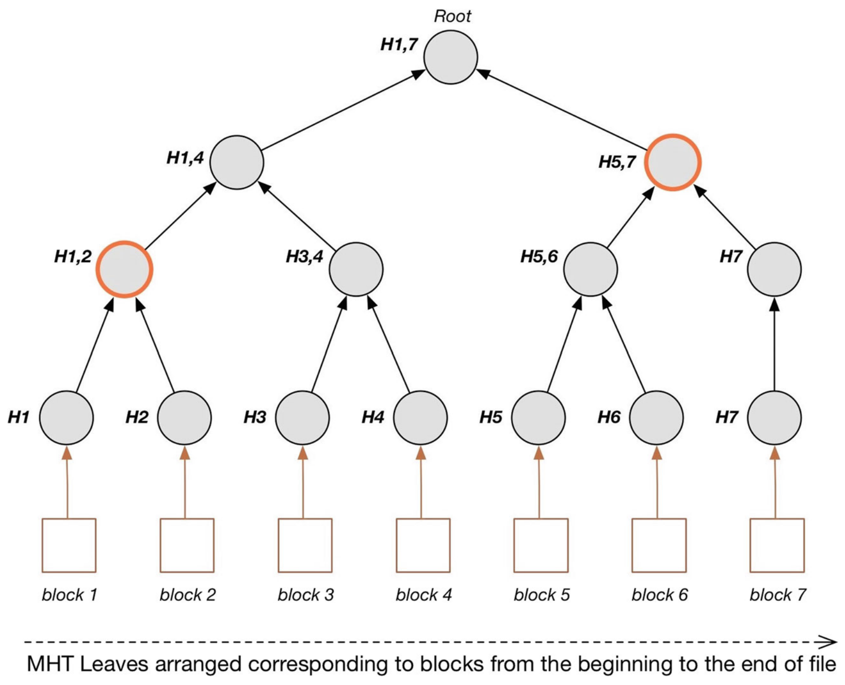

Fig. 3 An example of Merkle Hash Tree of a file with 7 blocks

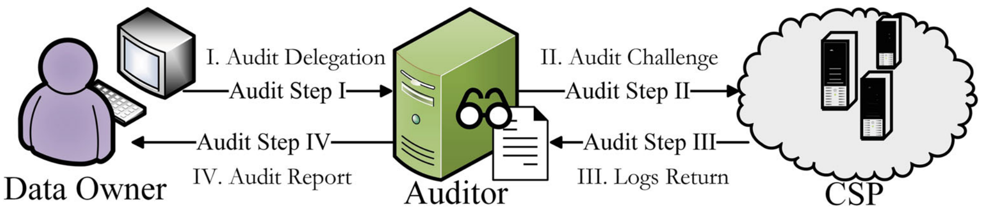

Fig. 4 Log auditing process

file, $DataHash$ is decided by inspecting MHT with the strategy illustrated in an example below. Given a 7-block file as Fig. 3 and operation on data block 1,2,5,6,7, we can aggregate these blocks' hash upwards the file's MHT. The aggregation will finally obtain some end nodes, which are $H1, 2$ and $H5, 7$ in Fig. 3. The operation is then recorded as 2 logs whose $DataHash$ fields are $H1, 2$ and $H5, 7$, respectively.

While we use MHT to build tree of file in the cloud, an access on some data blocks of the file will generate corresponding multi-grained logs. The log generator will first aggregates the data blocks with the file MHT and gets the $DataHash$ to uniquely identify the accessed data blocks. With other fields of log calculated, the access can be logged.

When data is modified in the cloud, the file MHT in CSP needs updating so that CSP could make logs with new block hash. MHT updating strategy can make use of common binary tree mechanism [17].

4.2 Multi-grained efficient log auditing

Log auditing process is shown in Fig. 4.

As Step I shows, the data owner first generates a log auditing request including delegation for auditor. After receiving auditing request, the auditor sends a log auditing challenge to CSP, which is shown as step II. If CSP verifies a legal challenge, he acts as step III: giving all logs of the file to the auditor. Then the auditor executes log auditing on the file logs and generates an auditing report for Data Owner in step IV.

4.2.1 Log auditing request

The design of request content is shown as Fig. 5 below.

In the request, $AuditMode$ field is *LogAudit*, which means this request is to call for audit towards data logs. $FileName$ field is name of the file to which target data belongs. $AuthorizedUserList$ is all versions of authorized user HashID list for the file. $TargetDataHashTree$ is uniquely identifier of target data, which uses the file

Audit Mode	File Name	Authorized UserList	TargetData HashTree	Timestamp	Signature

Fig. 5 Auditing request content

MHT's subtree from any node downwards to leafnodes. $Timestamp$ shows this request's time. $Signature$ field is given by Data Owner with his secret key through signing on all the fields before.

Auditing request from Data Owner to auditor has the same content as the challenge from auditor to CSP. After CSP receives auditing challenge, he firstly reads $FileName$ field to locate file and extracts information of Data Owner. CSP then checks signature to confirm the identity of owner. If this legality checking passes, CSP will send all logs of the file to the auditor.

4.2.2 Log integrity checking

Log integrity are protected with $ChainHash$ and $Signature$. As file-oriented log is designed to chains all logs of a file with $ChainHash$, all logs of the file will be sent to the auditor to ensure log integrity. Here $ChainHash$ and $Signature$ is verified by Eqs. 1 and 2 respectively.

4.2.3 Data confidentiality auditing

In data confidentiality auditing, the auditor firstly requires data access information from authentic, integral, reliable logs and then compare access information with data authorization information.

Our auditing scheme supports multi-grained auditing targets, which means target data can be a data block, a series of blocks, or the whole file. To eliminate logs which is not related to our target data in auditing process, we conduct slicing on logs at first and generate an information refined *AccessList*.

AccessList is a link list recording the HashID of users who have accessed data and the version of HashID. With HashID and its version, we can accurately identify a unique user in the cloud environment. In order to improve auditing efficiency, *AccessList* need to follow two features: (1) no items in AccessList are identical; (2) items in AccessList are ordered as the version of HashID grows. Here we use $AL(len)$ to represent an *AccessList* which have len items, and the list's s^{th} item is expressed as $AL(len).pos(s)$ in which $0 < s \leqslant len$. In addition, the item's HashID version is $AL(len).pos(s).ver$. As the second feature of *AccessList* shows, a $newItem$'s insertion position of $AL(len)$ should meet the Eq. 3:

$$AL(len).pos(s).ver \leqslant newItem.ver < AL(len).pos(s+1).ver \quad (3)$$

After scanning the logs to generate *AccessList*, we use $AuthorizedUserList$ together to obtain unauthorized access information by which Data Owner could trace malicious user.

Algorithm 1 Access list generation.

```
Require: $TargetDataHashTree$, Logs
Ensure: $AccessList$
  while Logs not end do
    if $CurrentLog.DataHash.MHTsubtree \cap TargetDataHashTree \neq \varnothing$ then
      Current log entry is summarized to a $newItem$
      if $newItem \notin AL(len)$ then
        Insert $newItem$ into link list $AL(len)$ ordered by $newItem.ver$
        $len = len + 1$
      end if
    end if
    Current log entry moved to next one
  end while
  return $AL(len)$
```

AccessList generation As shown in Algorithm 1, we just generate AccessList with the logs which are concerned. We first extract $DataHash$ from current log entry and compare it with hash of audit target data. In the file's MHT, if the node of current $DataHash$'s subtree and $TargetDataHashTree$ have a non-empty intersection, this log entry is added into our audit scope. Then we extract this log entry's HashID and version and insert them into *AccessList* in ascending order with respect to $UserListVersion$.

Algorithm 2 Unauthorized access list generate.

```
Require: $AccessList, AuthorizedUserList$
Ensure: $UnauthorizedAccessList$
  while $AccessList$ not end do
    $s = AccessListcurrentposition$
    if $AL(len).pos(s) \in AuthorizedUserList$ then
      Remove $AL(len).pos(s)$ from link list $AL(len)$
      $len = len - 1$
    else
      $s = s + 1$
    end if
  end while
  return $AL(len)$
```

Leakage investigation With *AccessList* built, we use $AuthorizedUserList$ to remove the authorized accesses in *AccessList* and obtain unauthorized access on the target data we audit. The leakage investigation process is conducted as Algorithm 2. Return value is $UnauthorizedAccessList$ of the auditing target data. With the data leakage information in $UnauthorizedAccessList$, the auditor finally gives audit report to the data owner. If $UnauthorizedAccessList == \varnothing$, then target data's confidentiality is intact. Otherwise, $UnauthorizedAccessList$ stores the information of data leakage.

5 Results and discussion

In this section we briefly elaborates the security of our scheme and focus mainly on our performance including space and time efficiency.

5.1 Security analysis

Firstly, logs passed $ChainHash$ and $Signature$ verification are proved to be secure with both forward-security and append-only properties as many reliable logging mechanisms has stated [8, 10, 18].

Secondly, the hash pseudonyms used in our log format preserve data confidentiality and user privacy in the log auditing process. With the correctness of data confidentiality auditing algorithm based on $AccessList$, confidentiality of data or user privacy cannot be leaked.

5.2 Space efficiency performance

Our file-oriented multi-grained logs can reduce log volume to optimize the space efficiency. Recalling the MHT in our scheme, we can prove that our multi-grained logging makes fewer log entries than fine-grained logging if blocks accessed by user are continuous or at least some of them are continuous.

In Fig. 6, we evaluate log size for a file composed of 8 blocks as example. While fine-grained logs have equal entries with accessed data block number, our multi-grained logs have some different occasions with possibilities. For instance, if a user access 5 blocks at a time, there may be 2, 3 or 4 resulting logs. The existence of multiple values for one access size can be explained by the fact that users randomly choose blocks to access thus blocks may either be continuous or separated. The black curve shows that the number of resulting logs is equal to the number of blocks accessed if we conduct. When we implement multi-grained logging, the possible number of resulting logs is shown as individual points on blue line. With our assumption that blocks are accessed with equal probability, we are able to get

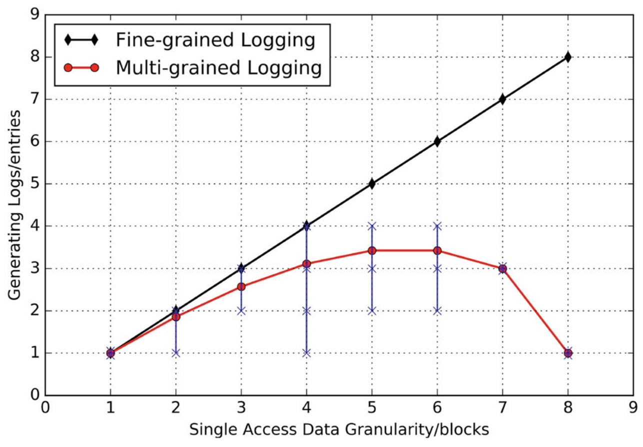

Fig. 6 Logs volume of one access on multi-blocks of 8-block file

the expectation of log entries with respect to the granularity of data access, as is shown by red curve in Fig. 6. Thus, the space cost of our multi-grained log is just 54% of that of the fine-grained log, as is shown in Fig. 6. Moreover, the improvement on efficiency by multi-grained logging is even remarkable when a user access more than half of blocks in a file, because of continuous blocks.

Extending the condition of B-blocks file, the log volume generated by one access is shown below: maximum log volume is B in fine-grained logs while its max value is $B/2$ in our multi-grained logs. In the assumption of uniformly distributed data access granularity, when B is a big number, our log generates approximately $B/4$ entries compared with $B/2$ entries of fine-grained log.

5.3 Time efficiency performance

We assume that CSP has great power such that MHT building and calculation cost in our scheme is only a small burden for CSP and can be neglected. Then performance of the auditing work is composed of three stages: 1) log integrity checking; 2) access list generation; 3) data confidentiality auditing.

We firstly give a theoretical analysis towards the complexity of each auditing step in Table 1, showing that our scheme achieves a preferable overall efficiency than the classic auditing scheme by entries when auditing certain blocks of a file.

Suppose the file logs have L entries in total and file size is B blocks. Among all log entries, a ratio of a are accessed by different hashIDs or versions. Auditing target data are supposed to be t continuous blocks. Commonly, ratio of logs related to target data can be approximately expressed as t/B. U is the number of all authorized users. Then we give out the time complexity of each auditing stage. Here "Hash" is hash calculation, "Sig" is signature calculation, "Ins" is link list insertion, "Del" is link list deletion, and "Comp" is comparison. As "Ins" and "Del" are operations of writing, these two operations consume far more time than "Comp" operation of reading. As t/B and a are both ratios less than 1, our scheme achieves better performance.

In addition to theoretical analysis, we conduct a set of experiments to simulate log auditing. In our experiment, auditor is a powerful computer with Intel Core-i5 5257U CPU and 8.0 GB RAM, and results of time are measured by repeating test for 5 times. The file has 8 blocks. Log volume and userlist size are large and access granularity is uniformly distributed. And we compare our scheme's multi-grained auditing time efficiency with the fine-grained log auditing scheme which is used in the state-of-the-art [18].

5.3.1 Log integrity checking

Because of uniformly distributed granularity, we can get the expectation of entries number of logs. For instance, our multi-grained logging generates 19392 log entries for 8000 data accesses, which is 54% of the 36000 entries generated by fine-grained logging. In our experiment, it takes 3.58s to conduct integrity checking with $ChainHash$ and $Signature$ on these 19392 entries, which means an acceptable efficiency.

5.3.2 AccessList generation

Following our AccessList Generation algorithm, we conduct a set of experiments where data access scale is from 19392 to 96960 log entries. For every data access scale, our auditing granularity includes 1, 2, 4, 8 blocks. Because fine-grained log auditing [18] uses all log entries to generate AccessList, the auditing granularity of 8 blocks for this 8-block file is fine-grained log auditing. The relationship between the log volume and the corresponding time to make AccessList is shown in Fig. 7.

The results shows that our AccessList generation costs 60% time compared with fine-grained auditing on logs. For instance, for 40000 access of uniformly distributed data

Table 1 Log auditing time complexity

	Integrity	AccessList Generation	Investigation
Classic scheme	LHash $+L$Sig	LIns	LUComp $+\frac{U}{a}$Del
Our scheme	LHash $+L$Sig	$((log B + 2t)L + \frac{at^2L^2}{B^2})$Comp$+\frac{atL}{B}$Ins	$\frac{atLU}{B}$Comp $+U$Del

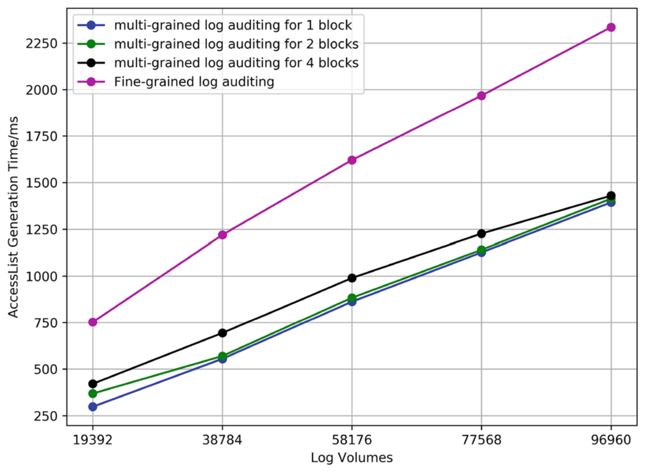

Fig. 7 AccessList generation efficiency

granularity, we have 96960 log entries. Then AccessList generation cost 1.394s for auditing 1 block, and time comes to 2.335s for auditing all blocks.

5.3.3 Data leakage investigation

We test with 5 different userlist whose sizes lie from 4×10^3 to 2×10^4 with equal spacing. We assume that each userlist is composed of 90% authorized users and 10% unauthorized users. Accordingly, our authorized userlists are of length from 3.6×10^3 to 1.8×10^4. For each user group, we ask some sublists of users to access data and then generate a set of access list with different sizes.

In Fig. 8, auditing time for all 5 test cases show the linear growth of audit time with the size of access list and authorized userlist respectively. Experiment results verify the high efficiency of this process, as it takes only 3.5 seconds to audit an access list of 2×10^4 items against a 1.8×10^4 authorized userlist.

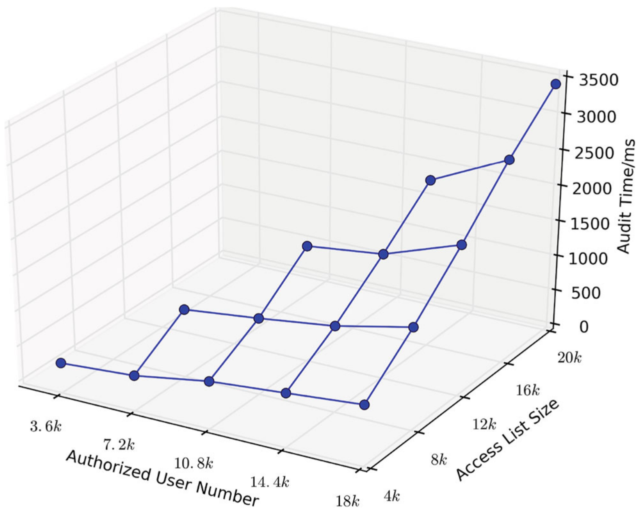

Fig. 8 Confidentiality audit efficiency

6 Conclusion

In consideration of that existing log auditing schemes are inefficiency, a multi-grained log auditing scheme is proposed in this paper to monitor cloud data confidentiality status and investigate data leakage. The scheme introduces Merkle Hash Tree structure to support multiple data granularity in logs. Based on our multi-grained log, we present a multi-grained data confidentiality auditing. Through our log auditing algorithms based on AccessList, we achieve significant efficiency amelioration than existing fine-grained logs based auditing.

Acknowledgments Supported by the National Key Research and Development Program of China (No.2016YFB0800402); the National Natural Science Foundation of China (No.U1405254, No.U1536207).

References

1. Ahsan MM, Wahab AWA, Idris MYI, Khan S, Bachura E, Choo KKR (2018) Class: cloud log assuring soundness and secrecy scheme for cloud forensics. IEEE Trans Sustain Comput
2. Chen Z, Tian H, Lu J, Nan F, Cai Y, Wang T, Chen Y (2017) Secure logging and public audit for operation behavior in cloud storage. In: 2017 IEEE international conference on computational science and engineering (CSE) and embedded and ubiquitous computing (EUC), vol 1. IEEE, pp 444–450
3. Cheng R, Xu R, Tang X, Sheng VS, Cai C (2018) An abnormal network flow feature sequence prediction approach for ddos attacks detection in big data environment. Comput Mater Continua 55(1):095–119
4. Li C, Hu J, Zhou K, Wang Y, Deng H (2018) Using blockchain for data auditing in cloud storage. In: International conference on cloud computing and security. Springer, pp 335–345
5. Liu Y, Peng H, Wang J (2018) Verifiable diversity ranking search over encrypted outsourced data. Comput Mater Continua 55(1):037–057
6. Ma D, Tsudik G (2009) A new approach to secure logging. ACM Trans Storage (TOS) 5(1):2
7. Merkle RC (1987) A digital signature based on a conventional encryption function. In: Proceedings of the conference on advances in cryptology (CRYPTO'87). Springer, pp 369–378
8. Popa RA, Lorch JR, Molnar D, Wang HJ, Zhuang L (2011) Enabling security in cloud storage slas with cloudproof. In: USENIX annual technical conference, vol 242
9. Stathopoulos V, Kotzanikolaou P, Magkos E (2006) A framework for secure and verifiable logging in public communication networks. In: International workshop on critical information infrastructures security. Springer, pp 273–284
10. Sundareswaran S, Squicciarini A, Lin D (2012) Ensuring distributed accountability for data sharing in the cloud. IEEE Trans Dependable Secure Comput 9(4):556–568

11. Sundareswaran S, Squicciarini A, Lin D, Huang S (2011) Promoting distributed accountability in the cloud. In: 2011 IEEE international conference on cloud computing (CLOUD). IEEE, pp 113–120
12. Tian H, Chen Z, Chang CC, Huang Y, Wang T, Huang ZA, Cai Y, Chen Y (2018) Public audit for operation behavior logs with error locating in cloud storage. Soft Comput :1–14
13. Tian H, Chen Z, Chang CC, Kuribayashi M, Huang Y, Cai Y, Chen Y, Wang T (2016) Enabling public auditability for operation behaviors in cloud storage. Soft Comput, pp 1–13
14. Wang C, Chow S, Wang Q, Ren K, Lou W (2013) Privacy-preserving public auditing for secure cloud storage. IEEE Trans Comput 62(2):362–375
15. Wang C, Wang Q, Ren K, Lou W (2010) Privacy-preserving public auditing for data storage security in cloud computing. In: 2010 proceedings IEEE on Infocom. IEEE, pp 1–9
16. Wang Q, Wang C, Li J, Ren K, Lou W (2009) Enabling public verifiability and data dynamics for storage security in cloud computing. In: Computer security–ESORICS 2009. Springer, Berlin, pp 355–370
17. Wang Q, Wang C, Ren K, Lou W, Li J (2011) Enabling public auditability and data dynamics for storage security in cloud computing. IEEE Trans Parallel Distrib Syst 22(5):847–859
18. Yang Z, Wang W, Huang Y (2017) Ensuring reliable logging for data accountability in untrusted cloud storage. In: 2017 IEEE international conference on communications (ICC). IEEE
19. Yavuz AA, Ning P, Reiter MK (2012) Efficient, compromise resilient and append-only cryptographic schemes for secure audit logging. In: International conference on financial cryptography and data security. Springer, pp 148–163
20. Zawoad S, Hasan R (2016) Trustworthy digital forensics in the cloud. Computer 49(3):78–81
21. Zawoad S, Hasan R, Skjellum A (2015) Ocf: an open cloud forensics model for reliable digital forensics. In: 2015 IEEE 8th international conference on cloud computing (CLOUD). IEEE, pp 437–444