Contents lists available at ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc

Lightweight integrity auditing of edge data for distributed edge computing scenarios[☆]

Liping Qiao [a], Yanping Li [a,b,*], Feng Wang [c,d], Bo Yang [e]

[a] School of Mathematics and Statistics, Shaanxi Normal University, Xi'an 710119, Shaanxi, China
[b] State key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, 100876, Beijing, China
[c] School of Computer Science and Mathematics, Fujian University of Technology, Fuzhou 350118, Fujian, China
[d] Fujian Provincial Key Laboratory of Network Security and Cryptology, Fujian Normal University, Fuzhou 350007, Fujian, China
[e] School of Computer Science, Shaanxi Normal University, Xi'an 710119, Shaanxi, China

ARTICLE INFO

Keywords:
Edge computing
Data integrity
Algebraic signature
Batch auditing
Provable dynamic update

ABSTRACT

With the increase of smart devices, edge computing is becoming a new computing paradigm that coexist with centralized cloud computing to process data distributed at the edge of the network. Based on this new paradigm, app vendors can store data replicas on geographically distributed edge servers to serve surrounding users to reduce access delay. However, since edge servers have limited storage space and computing ability, these edge data are vulnerable to various corruption. Therefore, how to efficiently audit the integrity of edge data has become an urgent problem to be solved. To tackle the Edge Data Integrity (EDI) problem, we propose a lightweight auditing scheme, namely EDI-SA. Firstly, inspired by the shuffle algorithm and the bucket sorting algorithm, we propose an improved sampling algorithm, which is a new lightweight challenge block sampling method. Secondly, based on algebraic signature, EDI-SA can achieve efficient aggregation verification and the signature computation cost is independent of the number of data blocks, which greatly reduces the computation overhead of app vendors and edge servers. Thirdly, EDI-SA supports batch auditing and provable dynamic update, app vendors can also uniquely locate the corrupted edge data. Finally, security and performance analyses demonstrate the security and effectiveness of EDI-SA.

1. Introduction

With the increase of smart devices (ipad, iphone), online applications such as the popular short video platforms like Tik Tok, ins, YouTube and Facebook have increased dramatically in recent years [1, 2]. However, when a large amount of data on these online applications are frequently accessed by numerous smart devices, huge network traffic load may cause network congestion and data access delay [3]. And due to the long distance between edge devices and the central cloud, latency and instability are extremely likely to occur, which seriously affects the user experience. Therefore, centralized cloud storage cannot meet the requirements of real-time and high-efficiency because of the transmission bandwidth limitations [4]. In this context, distributed edge storage has gradually become a research hotspot in academia and industry [5].

Edge computing is a new computing model in which computing and storage resources are deployed at the edge of a network closer to mobile devices or terminal users [6]. To go further, edge computing promotes the real-time interactivity of users and edge servers. Compared with cloud servers, edge servers have relatively limited computing and storage resources due to their size [5]. Generally, app vendors prestore massive data replicas on geographically distributed edge servers to support edge users to access data with low latency [7]. From app vendors' perspective, edge users access data on near-edge servers with low latency, rather than on remote clouds, which will further enhance the user experience.

Edge computing mainly faces four security challenges, namely, physical security, network security, application security and data security, where data security is the core. For app vendors, how to ensure the Edge Data Integrity (EDI) is a fundamental issue that needs to

☆ This work are partly supported by the National Natural Science Foundation of China [grant numbers U2001205]; the Open Foundation of State key Laboratory of Networking and Switching Technology (Beijing University of Posts and Telecommunications) [grant numbers SKLNST-2020-1-03]; the Natural Science Foundation of Fujian Province of China [grant numbers 2021J011066]; the Opening Foundation of Fujian Provincial Key Laboratory of Network Security and Cryptology Research Fund, Fujian Normal University (No. NSCL-KF2021-01).

* Corresponding author at: School of Mathematics and Statistics, Shaanxi Normal University, Xi'an 710119, Shaanxi, China.
E-mail addresses: qiaoliping@snnu.edu.cn (L. Qiao), lyp@snnu.edu.cn (Y. Li), w.h.feng@163.com (F. Wang), byang@snnu.edu.cn (B. Yang).

https://doi.org/10.1016/j.adhoc.2022.102906
Received 3 December 2021; Received in revised form 18 April 2022; Accepted 19 May 2022
Available online 28 May 2022

be solved urgently (Note that data replicas stored on edge servers are referred to as edge data). Nowadays, a considerable quantity of mature data integrity auditing schemes are proposed for centralized cloud storage [8–12], while they cannot be applied directly for edge computing model with high distribution and maintenance difficulty. Therefore, it is urgently to solve the EDI problem.

However, designing an EDI auditing scheme also faces many difficulties. First, there are multiple edge servers in distributed edge storage and app vendors may need to frequently update data in practice, such as software updates and VR game updates. Therefore, an EDI auditing scheme has more difficulty to manage and update outsourced data in a distributed edge multi-server scenarios than in the single cloud server scenarios. Second, since app vendors stores a lot of data replicas on the geographically distributed edge servers, an EDI auditing scheme should support app vendors to check the integrity of multiple edge data simultaneously. In other words, an EDI auditing scheme should support batch auditing. Third, compared with centralized cloud storage, edge servers have limited storage space and computing ability, and do not have large-scale server clusters and timely internal maintenance, which makes edge data vulnerable to various attacks such as maliciously or accidentally modified, deleted or forged. Therefore, an EDI auditing scheme should be able to locate the corrupted edge data in time and recover it to ensure its availability.

In order to address above issues, this paper designs a new auditing scheme called EDI-SA for distributed edge storage scenarios. Our EDI-SA also supports batch auditing, corruption localization and provable dynamic update. Specifically, the contributions can be summarized as follows:

(1) Inspired by the shuffle algorithm and the bucket sorting algorithm, we propose a new lightweight challenge block sampling method, called the improved sampling algorithm, which samples challenge blocks randomly and uniformly from edge servers.
(2) Based on algebraic signature, app vendors can efficiently aggregate and verify the integrity proofs returned from all edge servers. By making signature computation cost independent of the number of data blocks, i.e., the computation overhead of integrity verification and the communication overhead of integrity proofs will not increase with the number of auditing files, the computation overhead of app vendors and edge servers are especially lightweight.
(3) EDI-SA is highly scalable because it supports batch auditing and provable dynamic update. Besides, app vendors can effectively locate the corrupted edge data and recover them.
(4) We formally prove the security of EDI-SA. The performance is evaluated by numerical analysis and experimental simulation and the results demonstrate that EDI-SA is highly lightweight.

The rest is organized as follows. Section 2 presents a brief review of the related work. Section 3 describes the system model, the threat model and the design goals. Section 4 introduces algebraic signature technology. Section 5 specifically describes our EDI-SA, including the improved sampling algorithm, basic auditing of EDI-SA, extension to batch auditing, and provable dynamic update of edge data. Section 6 analyzes the security of EDI-SA. Section 7 presents the performance analysis of EDI-SA. Section 8 gives our conclusions.

2. Related work

As a new computing paradigm, edge computing has attracted extensive attention in recent years. Now, some related issues, such as distributed edge storage [13,14] have been widely studied. The scheme in [15] points out that edge computing greatly promotes the Internet of Things and mobile cloud computing, and will become the mainstream in the future. Therefore, it is of great significance to study EDI problem in distributed edge storage scenarios.

Most of the existing integrity auditing schemes for cloud data are designed for single-server scenarios. For instance, the famous Provable Data Possession (PDP) scheme and Proof of Retrieval (PoR) scheme are both originally proposed for single-server scenarios. In addition, there are a few schemes are proposed for multi-server scenarios. For example, Chang et al. [16] proposed a data integrity auditing scheme to audit multiple data replicas on multiple cloud servers. Su et al. [17] proposed a decentralized self-auditing scheme for multi-cloud storage. Curtmola et al. [18] proposed a multiple-replica PDP scheme, which can verify the integrity of multiple replicas that stored on multiple servers. However, these schemes brings huge redundancy to communication and storage. In order to reduce redundancy, Masood et al. [19] proposed a distributed public auditing scheme which is based on peer-to-peer architecture. Yang et al. [20] proposed a certificateless multi-replica and multi-cloud public auditing scheme based on blockchain technology. Liu et al. [21] proposed a cloud auditing scheme for multi-copy dynamic data integrity based on red-black tree. However, the schemes in [19–21] all introduce a trusted Third-Party Auditor (TPA) to help users audit the integrity of their outsourced data and free them from heavy computation. In distributed edge storage scenarios, app vendors have powerful ability and do not want the third party to access to their data. Therefore, the traditional data integrity auditing schemes designed for multiple cloud storage are not directly suitable for distributed edge storage scenarios.

Recently, some data integrity auditing schemes are proposed for distributed edge storage scenarios. In order to ensure data security of distributed edge storage, Tian et al. [22] proposed a privacy-protected public auditing scheme in fog-to-cloud computing scenarios, which not only can effectively protect data privacy and identity privacy of data users based on the label conversion strategy, but also combine computing power of edge devices with storage resources of cloud server to optimize cloud auditing. However, it realizes the public integrity auditing of cloud data rather than the integrity verification of edge data. Shu et al. [13] proposed a distributed storage scheme based on binary Reed–Solomon code to ensure the reliability of data in distributed edge storage. Aral et al. [14] proposed a decentralized replica placement algorithm for edge computing, in which multiple replicas of each file are stored in different edge servers to guarantee data robustness. However, both schemes in [13,14] neither consider the integrity auditing of distributed edge data.

To address the above problem, Tong et al. [23] proposed a privacy protection scheme that supports users to verify the integrity of edge data. This scheme can effectively protect the privacy of data owner during the auditing process. However, it does not support batch auditing. Li et al. [6] proposed a probabilistic EDI auditing scheme named EDI-V that is based on the Variable Merkle Hash Tree (VMHT) and ensure auditing accuracy by shuffling data blocks. Unfortunately, EDI-V has two evident disadvantages. First, EDI-V needs the app vendor and edge servers to regenerate a new VMHT and its multiple subtrees each time, respectively. Second, the app vendor in EDI-V only verify the proofs returned by edge servers one by one instead of aggregating them. These greatly increases the storage and computation overhead of the app vendor and edge servers. Afterwards, in order to achieve aggregate verification of the app vendor, Cui et al. [24] used homomorphic signature technology to design an EDI auditing scheme called ICL-EDI, which supports the app vendor to verify the integrity of multiple edge data, but it does not support batch auditing and cannot resist forgery attacks, replace attacks and replay attacks. Subsequently, Li et al. [25] proposed an EDI-S auditing scheme based on elliptic curve and aggregate signature technology. EDI-S supports batch auditing and can locate the corrupted edge data. Recently, Li et al. [26] proposed a CooperEDI scheme to check the integrity of edge data in a distributed manner. CooperEDI employs a distributed consensus mechanism to form a self-management edge caching system and supports to repair damaged edge data. However, the schemes in [25,26] do not consider provable dynamic update.

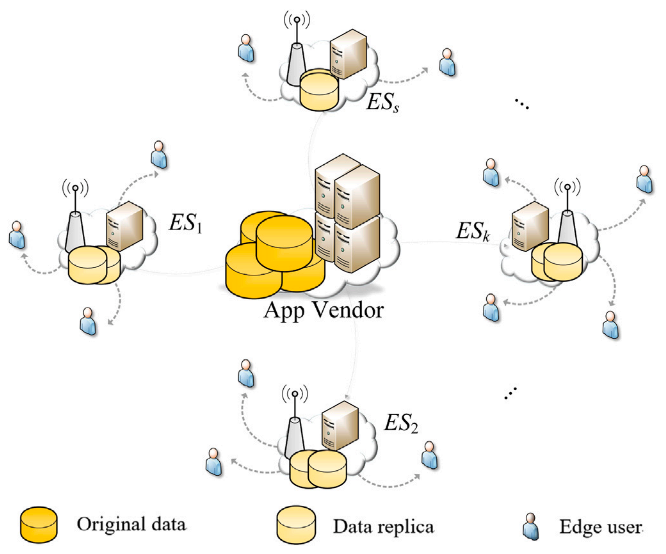

Fig. 1. The system model of EDI-SA.

Therefore, inspired by the existing excellent research achievements, we propose an EDI auditing scheme for distributed edge storage based on an improved sampling algorithm and algebraic signature. The improved sampling algorithm inspired by the shuffling algorithm and the bucket sorting algorithm is proposed to make the challenge more randomly and uniformly since the more random and uniform the index of challenge block is, the more accurate the integrity auditing is, which can effectively ensure the auditing accuracy of EDI-SA. Algebraic signature has additive homomorphism and can compress a large file block into a small bit string, resulting in a lower computation and communication overhead for app vendors and edge servers. Additionally, EDI-SA also supports batch auditing, corruption localization and provable dynamic update of edge data.

3. Problem statement

In this section, we introduce the system model, the threat model and the design goals.

3.1. System model

To allow edge users to access data with low latency, an app vendor employs multiple geographically distributed edge servers that close to edge users and pre-caches a lot of data replicas on them. Therefore, edge users can access data from nearby edge servers instead of a remote centralized server of the app vendor. As shown in Fig. 1, EDI-SA includes an app vendor and s edge servers ($s \geqslant 2$), denoted as $ES_1, ES_2, \dots, ES_s$, respectively.

(1) **App Vendor.** The app vendor has powerful computing resources, and stores massive data replicas on geographically distributed edge servers for reducing access delay and improving the user experience.
(2) **Edge Servers.** Edge servers have limited storage space and computing ability. And they are geographically close to edge users to facilitate users' access to data resources.

3.2. Threat model

Assume the app vendor is honest and trusted, i.e., the app vendor performs auditing task honestly. Assume all edge servers are independent of each other and do not collude, and each edge server may be dishonest, i.e., edge servers may forge, replace or replay integrity proofs to show the data replicas are intact when some of data replicas are actually corrupted or lost. Concretely, data replicas stored on edge servers (also called edge data) mainly face storage failures and three major attacks.

(1) **Unexpected Failures.** Faults such as hardware failures, software exceptions and cyber attacks may cause edge data to be corrupted.
(2) **Replay Attacks.** A dishonest edge server may use a correct data integrity proof previously generated to pass the new integrity auditing of the app vendor.
(3) **Replace Attacks.** A dishonest edge server may replace a corrupted block with another intact block stored by itself, or intercept integrity proof generated by another edge server as its own proof, so as to pass the integrity auditing of the app vendor.
(4) **Forgery Attacks.** A dishonest edge server may forge an integrity proof to pass the integrity auditing of the app vendor when some of edge data are actually corrupted or lost.

3.3. Design goals

Under the above system model and threat model, EDI-SA should meet the following four goals.

(1) **Correctness.** EDI-SA should ensure that the app vendor can correctly audit the integrity of edge data by verification equations.
(2) **Lightweight.** Given the resources limitation of edge servers, the auditing process should be performed with low computation overhead for edge servers. For the app vendor, the computation overhead is also as light as possible.
(3) **Security.** EDI-SA should prevent dishonest edge servers from performing replay attacks, replace attacks, and forgery attacks.
(4) **Data dynamics.** EDI-SA should support provable dynamic update since the app vendor may frequently updates its data in practice.

4. Preliminaries

In this section, we introduce algebraic signature in detail.

4.1. Algebraic signature

Algebraic signature has homomorphism and algebraic properties, and the operations are performed in Galois Field (GF). Suppose a file F is divided into n data blocks, i.e., $F = \{f_1, f_2, \dots, f_n\}$, the algebraic signature of F with the signature parameter α can be defined as follows.

$$Sig_\alpha(F) = Sig_\alpha(f_1, f_2, \dots, f_n) = \sum_{i=1}^{n} f_i \cdot \alpha^i \tag{1}$$

Where α is a primitive element of GF. More properties of the algebraic signature are shown below [27].

Proposition 1. *The algebraic signature of the sum of n blocks in file F is equal to the sum of algebraic signatures of each of the corresponding blocks.*

$$\sum_{i=1}^{n} Sig_\alpha(f_i) = Sig_\alpha\left(\sum_{i=1}^{n} f_i\right) \tag{2}$$

Proof. Suppose each data block in file F is divided into m sectors, f_{ij} represents the jth sector of the ith data block in file F.

$$\begin{aligned}\sum_{i=1}^{n} Sig_\alpha(f_i) &= \sum_{i=1}^{n}\Big(\sum_{j=1}^{m} f_{ij}\Big)\cdot \alpha^i \\ &= \sum_{j=1}^{m}\Big(\sum_{i=1}^{n} f_{ij}\Big)\cdot \alpha^i \\ &= Sig_\alpha\Big(\sum_{i=1}^{n} f_i\Big)\end{aligned}$$

Proposition 2. *Assume both files F and G are divided into n data blocks, i.e., $F = \{f_1, f_2, \dots, f_n\}$ and $G = \{g_1, g_2, \dots, g_n\}$. The algebraic signature of the sum of files F and G equals to the sum of algebraic signatures of each file.*

$$Sig_\alpha(F) + Sig_\alpha(G) = Sig_\alpha(F+G) \tag{3}$$

Proof.

$$\begin{aligned}Sig_\alpha(F) + Sig_\alpha(G) &= \sum_{i=1}^{n} f_i \cdot \alpha^i + \sum_{i=1}^{n} g_i \cdot \alpha^i \\ &= \sum_{i=1}^{n}(f_i + g_i)\cdot \alpha^i \\ &= Sig_\alpha(F+G)\end{aligned}$$

5. The EDI-SA

In this section, we first improve a sampling algorithm to make the challenge index more randomly and uniformly. Second, a basic scheme EDI-SA is put forward to efficiently check the integrity of edge data with low computation and communication, where the app vendor can simultaneously audit multiple data replicas of a single file from all edge servers. Thirdly, an extension scheme which supports batch auditing of multi-file and data provable dynamic update is proposed to make EDI-SA more practical and scalable.

Note that the signature parameter α of each round auditing is newly generated and regarded as a fresh factor (one-time random number), which can effectively prevent dishonest edge servers from replay attacks. Additionally EDI-SA eliminates the linear relationship between signature computation cost and the number of data blocks n. The computation cost is constant for edge servers and is linear related to the number of edge servers s for the app vendor. In practice, s is much smaller than n so as to EDI-SA can greatly reduce the computation and communication overhead.

For simplicity, we give some notations in Table 1.

5.1. The improved sampling algorithm

Inspired by the shuffling algorithm and the bucket sorting algorithm, we propose an improved sampling algorithm to reduce the computation overhead of EDI-SA. A traditional shuffling algorithm is, given a sequence from 1 to n, randomly shuffle this sequence to ensure that each number has the same probability of appearing in any position, i.e., the probability of each permutation in $n!$ permutations is the same. A traditional bucket sorting algorithm works by dividing an array into a limited number of buckets. Inspired by this, we can treat the s edge servers as a limited number of buckets and the index of n data blocks as an array, and further divide n data blocks into s edge servers.

Specially, when EDI-SA audits the integrity of s replicas of file F (assume F is divided into n data blocks) on s edge servers, the array $\{1, 2, \dots, n\}$ is first randomly sorted by using a pseudo-random permutation function $Per(x, y)$, where x is the random number and y is the total number of data blocks to be permuted. Then the sorted indexes are divided into s subsets, satisfying the intersection of s subsets is empty and the union is $\{1, 2, \dots, n\}$. These s subsets are used as

Table 1
The notations in our scheme.

Notation	Description
ES_k	The kth edge server
F	The original data F and its replicas are all denoted as F
f_i	The ith block of F
n	The number of blocks data F is divided into
s	The number of edge servers
GF(2^l)	The extension field of GF(2) and the addition on it is XOR
α	The parameter of algebraic signature
ID_k	The identity of the edge server ES_k
ID	$ID = \{ID_1, ID_2, \dots, ID_s\}$
I_F	The unique identifier of data replica F
ϕ	A pseudo-random function
$Per()$	A pseudo-random permutation function
C_k	The challenged index set of ES_k
f_{C_k}	The challenged block set of F on ES_k
$f_{C_k^\lambda}$	The challenged block set of F^λ on ES_k
$chal_k$	Challenged initiated by the app vendor to ES_k
$\Vert$	The concatenation of strings
$\bigoplus_{i\in\{1,\dots,n\}} f_i$	The XOR of all elements denoted as $f_1 \bigoplus f_2 \bigoplus \cdots \bigoplus f_n$

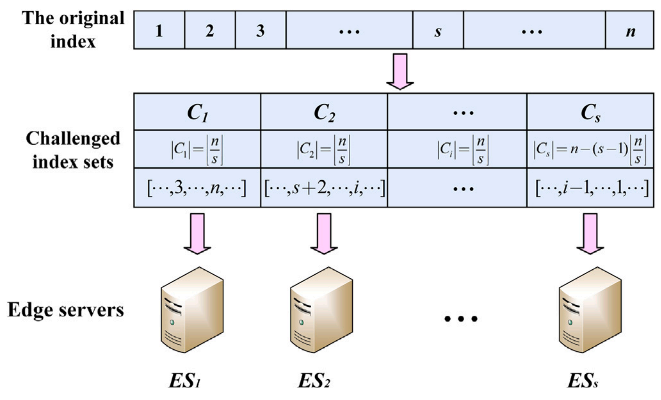
The original index
1
2
3
···
s
···
n
Challenged index sets
C_1
C_2
···
C_s
$|C_1|=\lfloor \frac{n}{s} \rfloor$
$|C_2|=\lfloor \frac{n}{s} \rfloor$
$|C_i|=\lfloor \frac{n}{s} \rfloor$
$|C_s|=n-(s-1)\lfloor \frac{n}{s} \rfloor$
$[\cdots,3,\cdots,n,\cdots]$
$[\cdots,s+2,\cdots,i,\cdots]$
···
$[\cdots,i-1,\cdots,1,\cdots]$
Edge servers
ES_1
ES_2
···
ES_s

Fig. 2. An illustration of sampling challenge blocks based on the improved sampling algorithm.

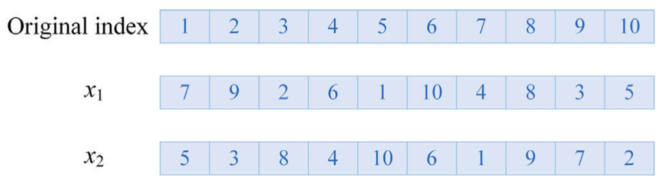

Fig. 3. An illustration of shuffling the indexes.

challenged index sets for s edge servers, respectively. Generally, as shown in Fig. 2, each subset has about $\lfloor n/s \rfloor$ elements, where $\lfloor n/s \rfloor$ represents the integer part of n/s. In particular, the last subset has $n - (s-1)\lfloor n/s \rfloor$ elements.

For example, assume there are two edge servers ES_1 and ES_2 and 10 data blocks whose indexes are denoted as $\{1, 2, \dots, 10\}$, see Fig. 3. Then, EDI-SA shuffles the indexes of these 10 data blocks by a pseudo-random permutation function $Per(x_1, 10) = [7, 9, 2, 6, 1, 10, 4, 8, 3, 5]$ (or $Per(x_2, 10) = [5, 3, 8, 4, 10, 6, 1, 9, 7, 2]$ if the random seed x_1 changes into x_2) by **Algorithm 1**. Thus ES_1 may get the challenged index set $C_1 = \{7, 9, 2, 6, 1\}$ and ES_2 get the challenged index set $C_2 = \{10, 4, 8, 3, 5\}$.

Further to say, $Per(x, y)$ first ensures that each index has the same probability of appearing in any position. Second, even if y is unchanged, the output of $Per(x, y)$ will change due to the different x. Third, since the signature parameters α of each round are also different, even if two challenged indexes are the same, the integrity proofs of different auditing rounds are different and cannot be replaced each other. The more random and uniform the index (challenged blocks) is, the more

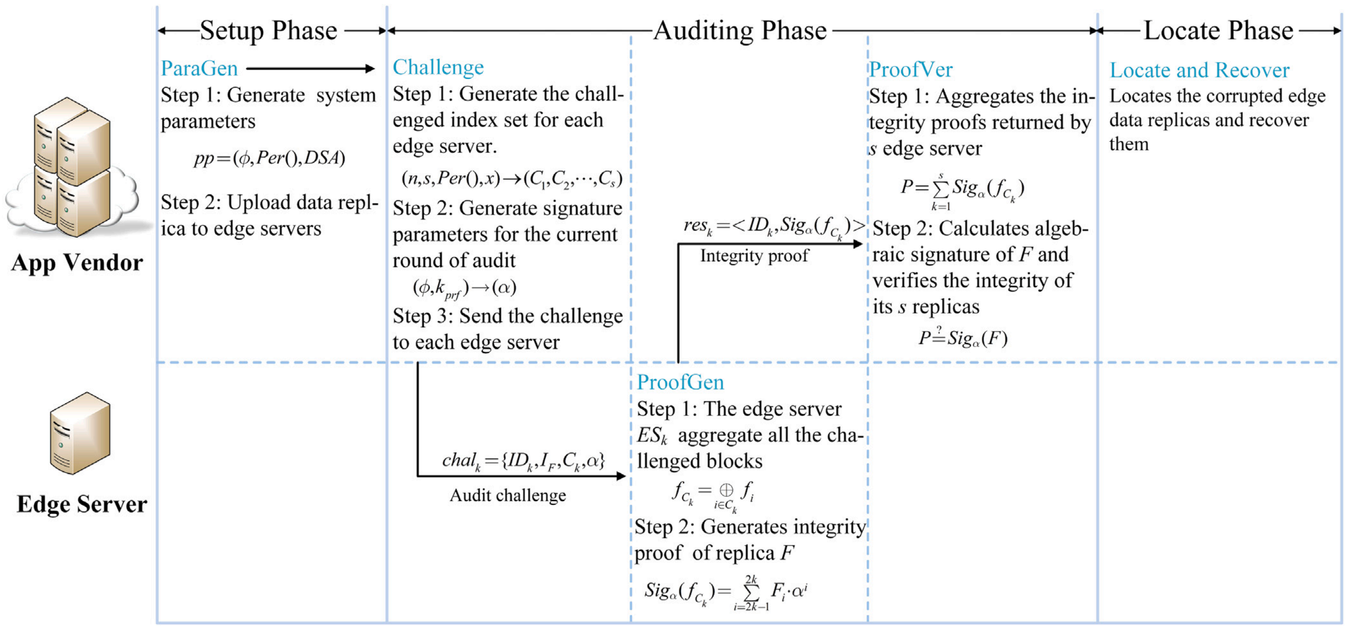
Setup Phase
Auditing Phase
Locate Phase
App Vendor
Edge Server
ParaGen
Step 1: Generate system parameters
$pp=(\phi,Per(),DSA)$
Step 2: Upload data replica to edge servers
Challenge
Step 1: Generate the challenged index set for each edge server.
$(n,s,Per(),x)\to(C_1,C_2,\cdots,C_s)$
Step 2: Generate signature parameters for the current round of audit
$(\phi,k_{prf})\to(\alpha)$
Step 3: Send the challenge to each edge server
$chal_k=\{ID_k,I_F,C_k,\alpha\}$
Audit challenge
ProofGen
Step 1: The edge server ES_k aggregate all the challenged blocks
$f_{C_k}=\bigoplus_{i\in C_k} f_i$
Step 2: Generates integrity proof of replica F
$Sig_\alpha(f_{C_k})=\sum_{i=2k-1}^{2k} F_i\cdot\alpha^i$
$res_k=<ID_k,Sig_\alpha(f_{C_k})>$
Integrity proof
ProofVer
Step 1: Aggregates the integrity proofs returned by s edge server
$P=\sum_{k=1}^{s} Sig_\alpha(f_{C_k})$
Step 2: Calculates algebraic signature of F and verifies the integrity of its s replicas
$P\overset{?}{=}Sig_\alpha(F)$
Locate and Recover
Locates the corrupted edge data replicas and recover them

Fig. 4. Workflow diagram of our EDI-SA approach.

accurate the integrity audit is. The improved sampling algorithm is given in **Algorithm 1**.

Algorithm 1: the improved sampling algorithm to sample challenged blocks for s edge server

```
Input: $T=\{1,2,\cdots,n\}$, $s$, $x$
Output: $\{C_k\}_{k=1}^{s}$
1 initial empty array list $C_1[]$, $C_2[]$, $\cdots$, $C_s[]$
2 $T'\leftarrow T$ s.t. $T'=\{Per(x,n)[1], Per(x,n)[2]\cdots Per(x,n)[n]\}$
3 for $i$ in $[1, s-1]$ do
4     for $j$ in $[i, i+\lfloor\frac{n}{s}\rfloor-1]$ do
5         $C_i[].add(t'_{(s+1)\cdot(i-1)+j})$
6 for $i$ in $[(s-1)\cdot\lfloor\frac{n}{s}\rfloor+1, n]$ do
7     $C_s[].add(t'_i)$
8 return $\{C_k\}_{k=1}^{s}$=$\{C_1, C_2, \cdots, C_s\}$
```

5.2. The basic auditing scheme of EDI-SA

In the basic auditing scheme, only a replica on each edge server is audited, and these replicas are the same since they all are the replicas of the original file F. Note that the original file F that stored on the app vendor and its s data replicas that cached on s edge servers are all denoted as F. The basic auditing scheme includes three phases: **Setup Phase**, **Auditing Phase**, **Locate and Recover Phase**. Specifically, the framework is shown in Fig. 4.

Setup Phase. In this phase, the system parameters are generated and data replicas are outsourced to s edge servers.

ParaGen$(\kappa)\to(pp)$: Input a security parameter κ, the system generates a keyed pseudo-random function (PRF) $\phi:\{0,1\}^\kappa\to\{0,1\}^l$, where $k_{prf}\in\{0,1\}^\kappa$ is the key of ϕ, a pseudo-random permutation function $Per(x,y)$, where the random number x is the seed and y is the total number of data blocks to be permuted, an uniform data segmentation algorithm DSA, which is defined to enable the app vendor and edge servers for the same file to obtain the same block. The public system parameters are $pp=\{\phi, Per(), DSA\}$.

Data Upload: Assume the app vendor owns the public data or the encrypted file F. Then the app vendor divides each replica of F into n blocks, i.e., $F=f_1\|f_2\|\cdots\|f_n$, where each f_i is l bit to ensure that $f_i\in$ GF(2^l), $i\in\{1,2,\ldots,n\}$. If the length of the last block is less than l bit, fill it with the identifier. Finally, the app vendor sends $F=f_1\|f_2\|\cdots\|f_n$ to each edge server.

Auditing Phase. In this phase, the integrity of s data replicas of F on s edge servers will be verified.

Challenge$(pp, ID, I_F, T, s, x)\to(chal)$: The app vendor challenge s edge servers and the challenged index set of each edge server is unique, i.e., union of these s challenged index sets is the entire index set of F and the intersection is empty. In other words, the challenged blocks of s edge servers are completely unrepeatable and their union is exactly all the data blocks of F. To be fair, each edge server has a completely different challenge block for each round.

(1) First, the app vendor selects a random number x as the random seed of $Per()$. Then the app vendor runs **Algorithm 1** to generate the challenged index set C_k for each ES_k, where $C_k=\{k_1,k_2,\ldots,k_r\}$ and satisfying $C_1\cup C_2\cup\cdots\cup C_s=\{1,2,\ldots,n\}$ and $C_i\cap C_j=\emptyset$ for $i\neq j, i,j\in\{1,2,\ldots,s\}$.
(2) Second, the app vendor randomly picks $k_{prf}\in\{0,1\}^\kappa$ and calls ϕ to generate the signature parameter α of the current auditing. Note that α is a primitive element of GF(2^l) and it never repeats.
(3) Third, the app vendor sends the challenge $chal_k=\{ID_k, I_F, C_k, \alpha\}$ to ES_k, where ID_k is the identity of ES_k, I_F is the unique identifier of F, C_k is the challenged index set of ES_k.

ProofGen$(ID_k, I_F, chal_k, F)\to(res_k)$: $ES_k, k\in\{1,2,\ldots,s\}$ responds to the challenge by providing an integrity proof of F.

(1) First, when ES_k receives the challenge $chal_k$, it verifies ID_k and I_F in $chal_k$. If they are not correct, ES_k recontacts the app vendor to provide a new challenge. Otherwise, ES_k does the steps below.
(2) Second, ES_k aggregates all the challenged blocks, i.e., $f_{C_k}=\bigoplus_{i\in C_k} f_i$, then it divides f_{C_k} into three blocks F_{3k-2}, F_{3k-1} and F_{3k} according to the uniform data segmentation algorithm DSA, satisfying $f_{C_k}=F_{3k-2}\|F_{3k-1}\|F_{3k}$. Besides, ES_k calculates the algebraic signature of f_{C_k}, i.e., $Sig_\alpha(f_{C_k})=\sum_{i=3k-2}^{3k}F_i\cdot\alpha^i, k=\{1,2,\ldots,s\}$.
(3) Third, ES_k sends res_k=<$ID_k, Sig_\alpha(f_{C_k})$> (the integrity proof) to the app vendor.

ProofVer$(res, pp) \rightarrow (True, False)$: The app vendor receives $res = \{res_1, res_2, \dots, res_s\}$ and aggregates all the proofs to verify s replicas that stored on s edge servers are intact or not.

(1) First, the app vendor aggregates all the integrity proofs returned by s edge servers, i.e., $P = \sum_{k=1}^{s} Sig_\alpha(f_{C_k})$.
(2) Second, the app vendor obtains $\{f_{C_1}, f_{C_2}, \dots, f_{C_s}\}$ according to the challenged index set C_k of $chal_k$ and data blocks $\{F_1, F_2, F_3, \dots, F_{3s-2}, F_{3s-1}, F_{3s}\}$ according to the uniform data segmentation algorithm DSA, satisfying $f_{C_k} = F_{3k-2} \| F_{3k-1} \| F_{3k}$, where $k \in \{1, 2, \dots, s\}$. Subsequently, the app vendor calculates the algebraic signature of F, i.e., $Sig_\alpha(F) = \sum_{i=1}^{3s} F_i \cdot \alpha^i$.
(3) Third, the app vendor verifies the following Eq. (4) holds or not.

$$P = Sig_\alpha(F) \tag{4}$$

If Eq. (4) holds, the app vendor returns *True*, which means that s replicas of F on s edge servers are intact. Otherwise returns *False*, which means that at least one edge data replica of F is corrupted.

Locate and Recover Phase. Assume the auditing result is *False*, the app vendor can apply the binary search to locate the corrupted data replica and recover it in this phase.

(1) The app vendor aggregates the $Sig_\alpha(f_{C_k})$ in all res_k into two parts P_{left} and P_{right} according to the binary search.
(2) Since the values of $F_i \cdot \alpha^i, i \in \{1, 2, \dots, 3s\}$ are calculated and recorded in **ProofVer**, the app vendor adds these signature values according to the binary search and matches them with P_{left} and P_{right}.
(3) The processes (1) and (2) are respectively iterated until all the corrupted replicas are located. Since the app vendor owns the original file, it can recover the corrupted replica by overwriting the original file.

5.3. Extension to batch auditing

Considering that a large number of data replicas are stored on each edge server in the real world, the app vendor needs to audit multiple data replicas on s edge servers concurrently. Auditing multiple replicas on s edge servers individually using the basic auditing scheme will be tedious and inefficient. In this section, we extend the basic auditing scheme to allow the app vendor to simultaneously audit multiple data replicas on s edge servers.

Setup Phase. In this phase, the algorithm **ParaGen** and the process of **Data Upload** are exactly the same with those in the basic auditing scheme.

Auditing Phase. In this phase, the integrity of m data replicas $\{F^\lambda\}_{\lambda=1}^m$ on each edge server will be audited at the same time.

Challenge: The app vendor challenges s edge servers. Specifically, the challenged index set of all challenged data replicas on each edge server is the same.

(1) First, the app vendor selects a random number x and generates the challenged index set C_k for each edge server, satisfying $C_1 \cup C_2 \cup \dots \cup C_s = \{1, 2, \dots, n\}$ and $C_i \cap C_j = \emptyset$ for $i \neq j, i, j \in \{1, 2, \dots, s\}$, where $C_k = \{k_1, k_2, \dots, k_r\}$ is the challenged index set of all data replicas on ES_k.
(2) Second, the app vendor randomly picks $k_{prf} \in \{0,1\}^\kappa$ and calls ϕ to generate the signature parameter $\alpha \in GF(2^l)$ of this auditing.
(3) Third, the app vendor sends the challenge $chal_k = \{ID_k, \{I_{F^\lambda}\}_{\lambda=1}^m, C_k, \alpha\}$ to ES_k, where ID_k is the identity of ES_k, I_{F^λ} is the unique identifier of F^λ.

ProofGen: Each edge server responds to the challenge by providing an integrity proof of m data replicas.

(1) First, when ES_k receives the challenge $chal_k$, it verifies ID_k and $\{I_{F^\lambda}\}_{\lambda=1}^m$ in $chal_k$. If they are not correct, ES_k recontacts the app vendor to provide a new challenge. Otherwise, ES_k does the steps below.
(2) Second, ES_k aggregates all the challenged blocks of F^λ, i.e., $f_{C_k}^\lambda = \bigoplus_{i \in C_k} f_i^\lambda$, $\lambda \in \{1, 2, \dots, m\}$, then it divides $f_{C_k}^\lambda$ into three blocks F_{3k-2}^λ, F_{3k-1}^λ and F_{3k}^λ according to DSA, which satisfies $f_{C_k}^\lambda = F_{3k-2}^\lambda \| F_{3k-1}^\lambda \| F_{3k}^\lambda$. In addition, ES_k calculates the algebraic signature of $f_{C_k}^\lambda$, i.e., $Sig_\alpha(f_{C_k}^\lambda) = \sum_{i=3k-2}^{3k} F_i^\lambda \cdot \alpha^i$ and computes $p_k = \sum_{\lambda=1}^m Sig_\alpha(f_{C_k}^\lambda)$.
(3) Third, ES_k sends the integrity proof $res_k = \langle ID_k, p_k \rangle$ to the app vendor.

ProofVer: The app vendor aggregates all the integrity proofs received from s edge servers to verify the m data replicas $\{F^\lambda\}_{\lambda=1}^m$ that stored on each edge server are intact or not.

(1) First, the app vendor aggregates all the integrity proofs returned by s edge servers, i.e., $P = \sum_{k=1}^s p_k$.
(2) Second, the app vendor gets $\{f_{C_1}^\lambda, f_{C_2}^\lambda, \dots, f_{C_s}^\lambda\}$ according to the challenged index set C_k of $chal_k$ and data blocks $\{F_1^\lambda, F_2^\lambda, F_3^\lambda, \dots, F_{3s-2}^\lambda, F_{3s-1}^\lambda, F_{3s}^\lambda\}$ according to DSA, satisfying $f_{C_k}^\lambda = F_{3k-2}^\lambda \| F_{3k-1}^\lambda \| F_{3k}^\lambda$. Subsequently, the app vendor calculates the algebraic signature of $\sum_{\lambda=1}^m F^\lambda$, that is, $Sig_\alpha\left(\sum_{\lambda=1}^m F^\lambda\right) = \sum_{\lambda=1}^m Sig_\alpha\left(F^\lambda\right) = \sum_{\lambda=1}^m \sum_{i=1}^{3s} F_i^\lambda \cdot \alpha^i$.
(3) Finally, the app vendor verifies the following Eq. (5) holds or not.

$$P = Sig_\alpha\left(\sum_{\lambda=1}^m F^\lambda\right) \tag{5}$$

5.4. Data provable dynamic update

In actual applications, the app vendor may need to update the data and its replicas stored on edge servers. Therefore, we use Divide and Conquer Adjacency Table (D&CAT) data structure [28] to realize dynamic data update. When inserting or deleting a data block, only the linked list pointer of the corresponding data block needs to be modified without moving subsequent data blocks. Furthermore, the provable update is performed to verify whether the update are correct or not.

The D&CAT consists of the header array and the linked list. Take the D&CAT data structure of F with unique identifier I_F in Fig. 5 as an example, each node of header array is composed of five parts (I_F, *LMIN*, *LMAX*, *REAR*, *FIRST*), where *LMIN* indicates the smallest index number of the data block in the corresponding linked list, *LMAX* means the largest index number of the data block in the corresponding linked list, *REAR* denotes the last node pointer of the homologous linked list and *FIRST* denotes the head pointer of the homologous linked list. The linked list is composed of three parts (*LN*, *TIME*, *NEXT*), where *LN* represents the index number of the data block, *TIME* is the latest update time of the data block, and *NEXT* denotes the pointer of the next node. According to the *LMIN* and *LMAX* of the table header node, the node can be quickly located, which can better improve the efficiency of the data update operation.

The app vendor creates the corresponding D&CAT data structure for each data replica before storing it on edge servers. For the sake of clarity, the following only shows the dynamic update of one data block of the data replica F on ES_k. All the regenerating a new signature parameter α in the following three update algorithms is mainly for the purpose of provable updating. If we do not need provable updating, the ES_k just need to locate and modify/insert/delete the target data block without generating α and computing algebraic signature of the update block.

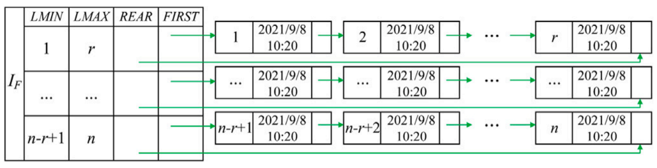

Fig. 5. The original D&CAT-I_F.

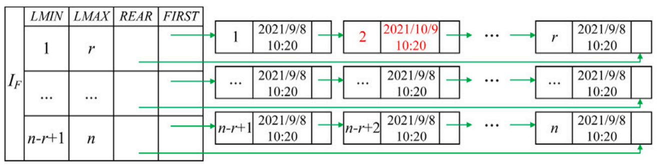

Fig. 6. D&CAT-I_F after modifying the 2nd block.

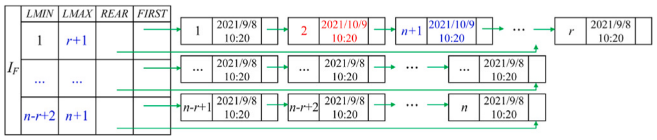

Fig. 7. D&CAT-I_F after inserting the 3rd block.

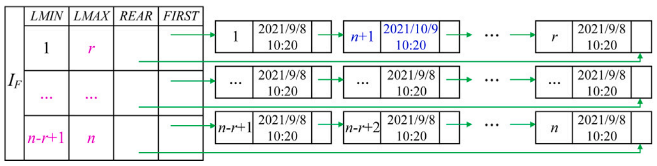

Fig. 8. D&CAT-I_F and the changes after updates.

5.4.1. *Modify*$(ID_k, I_F, i) \rightarrow (f'_i, Node'_i)$

(1) Suppose the app vendor needs to modify the ith data block f_i of F to f'_i, it first calls the PRF ϕ to generate a new signature parameter α and sends the update request Mod_i=$\{ID_k, I_F, i, f'_i, \alpha\}$ to ES_k.

(2) After receiving Mod_i, ES_k first finds the corresponding D&CAT-I_F according to the unique identifier I_F and looks up the table header node by the index number i. If $LMIN \leqslant i \leqslant LMAX$, ES_k traverses the relevant linked list with the header pointer *FIRST* to find the corresponding $Node_i$. Then, ES_k replaces f_i with f'_i and modifies the time $TIME_i := TIME'_i$ in $Node_i$, where $TIME'_i$ is the current time stamp. After that, ES_k computes $p'_k = f'_i \cdot \alpha^i$ to ensure the modification is correct and sends $res'_k = \langle ID_k, p'_k \rangle$ to the app vendor.

(3) When receiving res'_k, the app vendor also computes $Sig_\alpha(f'_i) = f'_i \cdot \alpha^i$ and verifies whether the following Eq. (6) holds or not.

$$p'_k = Sig_{\alpha'}(f'_i) \tag{6}$$

If Eq. (6) holds, it means that ES_k has correctly modified the block f_i to f'_i. Otherwise the process of modifying f_i needs to be redone. Fig. 6 shows the change of D&CAT-I_F data structure after the app vendor modifies the 2nd data block.

5.4.2. *Insert* $(ID_k, I_F, i, f^*_{i+1}) \rightarrow Node^*_{i+1}$

(1) Suppose the app vendor needs to insert a new block f^*_{i+1} after the ith data block, it first calls the PRF ϕ to generate a new signature parameter α^* and sends the update request Ins_i=$\{ID_k, I_F, i, f^*_{i+1}, \alpha^*\}$ to ES_k.

(2) After receiving Ins_i, ES_k first finds the D&CAT-I_F and looks up the table header node by the index number i. If $LMIN \leqslant i \leqslant LMAX$, ES_k traverses the relevant linked list with the header pointer *FIRST* to find the corresponding $Node_i$. Then, ES_k creates a new linked list $Node^*_{i+1}$ after $Node_i$ and inserts the new block f^*_{i+1}. For simplicity, *MAX* is used to indicate the maximum logical index number and in the new $Node^*_{i+1}$, $LN := MAX+1$, $TIME_{i+1} := TIME^*_{i+1}$, where $TIME^*_{i+1}$ is the current time stamp. In addition, ES_k sets $Next_{i+1} := Next_i$, and $Next_i$ points to the new insertion $Node^*_{i+1}$. Moreover, ES_k adds 1 to the *LMAX* of the current table header node and the *LMIN* and *LMAX* of the subsequent table header nodes. After that, ES_k calculates $p^*_k = f^*_{i+1} \cdot (\alpha^*)^{i+1}$ and sends $res^*_k = \langle ID_k, p^*_k \rangle$ to the app vendor.

(3) When receiving res^*_k, the app vendor also computes $Sig_{\alpha^*}(f^*_{i+1}) = f^*_{i+1} \cdot (\alpha^*)^{i+1}$ and verifies whether the following Eq. (7) holds or not.

$$p^*_k = Sig_{\alpha^*}(f^*_{i+1}) \tag{7}$$

If Eq. (7) holds, it means that ES_k has correctly inserted the block f^*_{i+1}. Otherwise the process of inserting f^*_{i+1} needs to be redone. Fig. 7 shows the change of D&CAT-I_F data structure after the app vendor inserts the 3rd data block.

5.4.3. *Delete* $(ID_k, I_F, i) \rightarrow (1, 0)$

(1) Suppose the app vendor needs to delete the ith data block, it sends the update request Del_i=$\langle ID_k, I_F, i \rangle$ to ES_k.

(2) After receiving Del_i, ES_k first searches the related D&CAT-I_F and looks up the table header node by the index number i. If $LMIN \leqslant i \leqslant LMAX$, ES_k traverses the relevant linked list with the header pointer *FIRST* to find the $Node_i$. Then, ES_k deletes block f_i. Specifically, if $Node_i$ is the first node of the linked list, set the table header node field $FIRST := Next_i$. If it is the last node of the linked list, then directly point the table header field *REAR* to $(i-1)$-th node. Generally, set $Next_{i-1} := Next_i$. And minus 1 from the *LMAX* of the current table header node and the *LMIN* and *LMAX* of the subsequent table header nodes. When the deletion is completed, the app vendor only needs to conduct an integrity auditing of the current data F to realize the provable deletion updating. Fig. 8 shows the change of D&CAT-I_F data structure after the app vendor deletes the 2nd data block.

6. Security analyses

This section first presents the correctness of EDI-SA. Then the security analyses of EDI-SA are given, including the efficiency and robustness of algebraic signature, as well as resistance to reply attacks, replace attacks and forge attacks.

6.1. Correctness

The correctness of EDI-SA is equivalent to the correctness and validity of verification equations Eqs. (4) and (5).

Theorem 1 (*Valid Verification*). *If edge servers store the intact and correct data replicas, they can generate the integrity proofs which can pass Eqs.* (4) and (5) *successfully.*

Proof. The correctness of Eq. (4) is elaborated below.

$$\begin{aligned} P &= \sum_{k=1}^{s} Sig_\alpha(f_{C_k}) \\ &= \sum_{k=1}^{s} \sum_{i=3k-2}^{3k} F_i \cdot \alpha^i \\ &= \sum_{i=1}^{3s} F_i \cdot \alpha^i \\ &= Sig_\alpha(F) \end{aligned}$$

Proof. The correctness of Eq. (5) is elaborated below.

$$\begin{aligned} P &= \sum_{k=1}^{s} p_k \\ &= \sum_{k=1}^{s}\sum_{\lambda=1}^{m} Sig_\alpha(f_{C_k}^{\lambda}) \\ &= \sum_{\lambda=1}^{m}\sum_{k=1}^{s} Sig_\alpha(f_{C_k}^{\lambda}) \\ &= \sum_{\lambda=1}^{m}\sum_{k=1}^{s}\sum_{i=3k-2}^{3k} F_i^{\lambda}\cdot\alpha^i \\ &= \sum_{\lambda=1}^{m}\sum_{i=1}^{3s} F_i^{\lambda}\cdot\alpha^i \\ &= \sum_{\lambda=1}^{m} Sig_\alpha(F^{\lambda}) \\ &= Sig_\alpha(\sum_{\lambda=1}^{m} F^{\lambda}) \quad \square \end{aligned}$$

Therefore Theorem 1 holds. □

Theorem 2 (*Valid Localization and Recovery*). *If the data replica F on ES_k has n data blocks, the app vendor can successfully detect this corrupted data replica at least with a probability of $Pr = 1 - \left(\frac{n-d_k}{n}\right)^{t_k}$ and recover it, where d_k is the number of corrupted data blocks of F, and t_k is the number of challenged blocks.*

Proof. Based on **Algorithm 1**, the challenged blocks on s edge servers are completely unrepeatable and their union is exactly all the data blocks of F. Consequently, the app vendor can uniquely detect the corrupted replica on ES_k according to the steps in **Locate and Recover Phase**. Since the probability of finding F corrupted is equal to the probability of at least challenging one corrupted data block. In other words, we can calculate the probability that at least one corrupted block is challenged on ES_k during the auditing as follows.

$$Pr = 1 - \left(\frac{n-d_k}{n}\right)\left(\frac{n-d_k-1}{n-1}\right)\left(\frac{n-d_k-2}{n-2}\right) \cdots \left(\frac{n-d_k-t_k+1}{n-t_k+1}\right)$$

Given an arbitrary integer $i \leqslant n$, there is $\frac{n-d_k-i}{n-i} \geqslant \frac{n-d_k-i-1}{n-i-1}$. Hence, the following inequality holds.

$$Pr > 1 - \left(\frac{n-d_k}{n}\right)^{t_k}$$

Furthermore, in the auditing process of this paper, the app vendor does not need to verify all the data blocks of each replica, but only randomly sample some data blocks for integrity verification, which is more efficient and economical. We will take an specific example to illustrate how corrupted edge data replica can be detected with a few challenged blocks. Firstly, for each edge data replica, when 1% of all data blocks are corrupted, i.e., $\frac{d_k}{n} = 1\%$, and sampling only $t_k = 450$ out of $n = 450,00$ data blocks, i.e., a sampling rate of $\frac{t_k}{n} = 1\%$, the app vendor achieves a 99.3% accuracy [24]. Furthermore, when 10% of all data blocks are corrupted, i.e., $\frac{d_k}{n} = 10\%$, challenging only $t_k = 110$ out of the $n = 1250,00$ blocks of data, i.e., a sampling rate of $\frac{t_k}{n} = 0.088\%$, the app vendor can detect the replica is corrupted with a high probability of 99.99% [28]. Secondly, although the number of challenged blocks sampled by app vendor on each edge server is relatively small, they are highly randomly sampled based on **Algorithm 1**. Therefore, in real life, this has a strong deterrent to edge servers so that they do not take the risk of cheating the app vendor. In addition, due to the strong randomness, each data block has the same probability of being challenged, so the corrupted block is easier to be detected. Thirdly, since the app vendor and edge servers are all lightweight in our EDI-SA, the app vendor can increase the frequency of audits, which also enables the app vendor to effectively detect the corrupted edge data replicas. Therefore, the app vendor can find all corrupted edge data replicas with a high probability and recover them by overwriting the original data in our EDI-SA. □

6.2. Security guarantee

Next, we will show the efficiency and robustness of algebraic signature and how EDI-SA can effectively resist replay attacks, replace attacks and forgery attacks.

Theorem 3 (*Resistance to Replay Attacks*). *ES_k implements replay attacks to pass the integrity verification with a negligible probability.*

Proof. Define the game of replay attacks as follows. Assume ES_k is an adversary. The app vendor sends a challenge $chal_k = \{ID_k, I_F, C_k, \alpha\}$ to ES_k, but ES_k returns an expired proof $res'_k = \langle ID_k, Sig_{\alpha'}(f_{C'_k})\rangle$ to the app vendor, where α' and C'_k are both expired, so algebraic signature $Sig_{\alpha'}(f_{C'_k})$ is also outdated. In particular, the expired proof is previously generated by itself or intercepted from other edge servers. If res'_k can pass the integrity verification, then ES_k wins the game. Otherwise, it demonstrates that our EDI-SA can effectively resist replay attacks.

We use α', C'_k and $Sig_{\alpha'}(f_{C'_k})$ to denote the expired signature parameter, challenged index set and algebraic signature of challenged blocks, respectively. And we denote the new signature parameter, challenged index set and algebraic signature of challenged blocks as α, C_k and $Sig_\alpha(f_{C_k})$, respectively. Assume res'_k can pass the verification. Based on **Algorithm 1**, the challenged index set on each edge server in distinct rounds is different due to the different seeds x of pseudorandom function Per, i.e., $C'_k \neq C_k$ and $f_{C_k} \neq f_{C'_k}$ with a high probability. Since α' is different from α with an absolute high probability, $Sig_{\alpha'}(f_{C'_k}) \neq Sig_\alpha(f_{C_k})$. Therefore, res'_k cannot pass the verification with a significant advantage.

Therefore, ES_k can win the game with a negligible probability, that is, EDI-SA can effectively resist replay attacks. □

Theorem 4 (*Resistance to Replace Attacks*). *ES_k implements replace attacks to pass the integrity verification with a negligible probability.*

Proof. Define the game of replace attacks as follows. Assume ES_k is an adversary. The app vendor sends a challenge $chal_k = \{ID_k, I_F, C_k, \alpha\}$ to ES_k, but ES_k returns a replaced proof $res'_k = \langle ID_k, Sig_\alpha(f'_{C_k})\rangle$ to the app vendor. The replaced proof means two cases: (1) ES_k replaces the challenged block f_i with its other unchallenged valid block f_j $(j \neq i)$ to generate the integrity proof res'_k. (2) ES_i uses the integrity proof res_j of other edge server ES_j $(j \neq k)$ as its proof res'_k. If res'_k can pass the integrity verification, ES_k wins the game. Otherwise, it proves that our EDI-SA can effectively resist replace attacks.

Case (1) In this case, ES_k uses its other unchallenged valid block f_j instead of f_i $(j \neq i)$ to generate f'_{C_k} and $Sig_\alpha(f'_{C_k})$, where $i \in C_k$ and $j \notin C_k$. Assume the correct integrity proof for challenged index set C_k is f_{C_k}. Because f_j is equal to f_i with a negligible probability, $f'_{C_k} \neq f_{C_k}$ with an overwhelming probability. Therefore, $Sig_\alpha(f'_{C_k}) \neq Sig_\alpha(f_{C_k})$ and Eq. (1) cannot hold. As a result, res'_k cannot pass the integrity verification.

Case (2) For each edge server, the challenged index set C_k of each round is different, and ES_k cannot pass the integrity verification by replacing its own signature with the signature of ES_j, because the union of all C_k is not equal to $\{1, 2, \ldots, n\}$. Further, even if ES_k and ES_j obtain the same challenged index set C_k with low probability in two different rounds of auditing, their signatures cannot be replaced because the app vendor uses different signature parameters α each time. In addition, we assume all edge servers do not collude with each other, so the ordinary malicious behavior of two edge servers exchanging signatures in the same round is not considered.

Table 2
Comparison of functions.

Scheme	Localization	Aggregation	Batch auditing	Data dynamic	provable updating	Security
EDI-V [6]	√	×	×	×	×	√
ICL-EDI [24]	√	√	×	×	×	×
EDI-S [25]	√	√	√	×	×	√
our EDI-SA	√	√	√	√	√	√

∗ √ : The scheme possesses the corresponding function.
∗ × : The scheme does not possesses the corresponding function.

Therefore, ES_k can win the game with a negligible probability, that is, EDI-SA can effectively resist replace attacks. □

Theorem 5 (*Resistance to Forgery Attacks*). *ES_k implements forgery attacks to pass the integrity verification with a negligible probability.*

Proof. Define the game of forgery attacks as follows. Assume ES_k is an adversary. The app vendor sends a challenge $chal_k = \{ID_k, I_F, C_k, \alpha\}$ to ES_k, but ES_k returns a forged integrity proof $res'_k = \langle ID_k, Sig_\alpha(f'_{C_k})\rangle$ to the app vendor, where at least a challenged block f'_i or an algebraic signature $Sig_\alpha(f'_{C_k})$ is forged. If res'_k can pass the verification, then ES_k wins the game. Otherwise, it demonstrates that our EDI-SA can effectively resist forgery attacks.

Let f'_i, $Sig_\alpha(f'_{C_k})$ denote the forged data block and integrity proof, f_i and $Sig_\alpha(f_{C_k})$ represent the correct data block and integrity proof, respectively. Assume $f'_i \neq f_i$, $f'_{C_k} \neq f_{C_k}$. Based on above fact, we discuss the probability of $Sig_\alpha(f_{C_k}) = Sig_\alpha(f'_{C_k})$, that is equal to the probability of $\sum_{i=3k-2}^{3k} F_i \cdot \alpha^i = \sum_{i=3k-2}^{3k} F'_i \cdot \alpha^i$ according to the data segmentation algorithm *DSA*, where $f_{C_k} = F_{3k-2}\|F_{3k-1}\|F_{3k}$ and $f'_{C_k} = F'_{3k-2}\|F'_{3k-1}\|F'_{3k}$. According to polynomial theory, $\sum_{i=3k-2}^{3k} F_i \cdot \alpha^i = \sum_{i=3k-2}^{3k} F'_i \cdot \alpha^i$ if and only if $F_i = F'_i, i \in \{3k-2, 3k-1, 3k\}$, which can derive that $f'_{C_k} = f_{C_k}$. This contradicts our assumption. Furthermore, when the length of algebraic signature is long enough, the collision probability is negligible. For example, when the length of algebraic signature is 128, 256, 512 bits respectively, a dishonest edge server forges a proof $Sig_\alpha(f'_{C_k})$ to pass the verification with the negligible probability of $\frac{1}{2^{128}}$, $\frac{1}{2^{256}}$, $\frac{1}{2^{512}}$, respectively.

Therefore, ES_k can win the game with a negligible probability, i.e., EDI-SA can effectively resist forgery attacks. □

7. Performance evaluation

In order to evaluate the performance of our EDI-SA, we first compare it with related schemes EDI-V [6], ICL-EDI [24], EDI-S [25] in terms of functions and security because the application scenarios of these three schemes are exactly the same as that of EDI-SA. Next, we numerically analyze our EDI-SA with EDI-V and ICL-EDI. Due to the data in EDI-S is not divided into blocks, we will not compare with it for the sake of fairness. Furthermore, we experimentally evaluate the computation overhead of **ParaGen**, **ProofGen** and **ProofVer** algorithms in above three schemes, respectively.

7.1. Function comparison

We compare the functions of the four schemes, as shown in Table 2. EDI-V can only verify the integrity proofs returned by edge servers one by one and, which greatly increases the computation overhead of the app vendor. EDI-S and ICL-EDI can achieve aggregation verification and EDI-S also supports batch auditing, but ICL-EDI cannot guarantee security, i.e., resistance to replay, replace and forgery attacks. More unfortunately, none of them support data dynamic and provable updating. However, our EDI-SA realizes all the functions and guarantees security and it is more suitable for practical application.

Table 3
Comparison of computation overhead.

Scheme	Setup	Challenge	ProofGen	ProofVer
EDI-V [6]	$O(n)$	$O(n+s)$	$O(n)$	$O(2^c+s)$
ICL-EDI [24]	$O(n)$	$O(s)$	$O(c)$	$O(s+c)$
our EDI-SA	$O(1)$	$O(s)$	$O(1)$	$O(s)$

Table 4
Comparison of communication overhead.

Scheme	Challenge	ProofGen
EDI-V [6]	$5s\|Z_p\|$	$\|Z_p\|$
ICL-EDI [24]	$2s\|Z_p\|$	$\|Z_p\|$
our EDI-SA	$2s\|Z_p\|$	$\|Z_p\|$

Table 5
Comparison of storage overhead.

Scheme	App vendor	Edge server
EDI-V [6]	VMHT	–
ICL-EDI [24]	$n\|Z_p\|$	–
our EDI-SA	D&CAT	D&CAT

Note: "–" means that there is no additional storage overhead.

7.2. Numeric analyses

For the sake of comparison, we give some notations as follows. n denotes the number of data blocks that each data replica is divided into. s is the number of edge servers. c represents the number of challenged blocks on each edge server. $|Z_p|$ denotes the element size in Z_p.

We first compare the time complexity of EDI-V, ICL-EDI, EDI-SA, the results are shown in Table 3. It can be seen that the time complexity of EDI-V is significantly higher than that of ICL-EDI and our EDI-SA. The reason is that in EDI-V, the app vendor builds a VMHT with n data blocks for verification, and each edge server generates a subtree of VMHT as the integrity proof in each round of auditing, which greatly increases the computation overhead of the app vendor and edge servers. When n is large, the depth of VMHT is quite large, the search is more time-consuming and the space occupancy rate of the app vendor is extremely high. Edge servers also face the same computation burden like the app vendor. Moreover, EDI-V has no aggregation function, it can only verify the proofs returned by edge servers one by one by comparing them with nodes on VMHT, this takes a lot of time searching on VMHT. Therefore, the time complexity of EDI-V is very high. ICL-EDI and our EDI-SA can aggregate integrity proofs, which can effectively reduce computation overhead. However, ICL-EDI cannot resist replay, replacing and forgery attacks. Further, our EDI-SA makes the cost of algebraic signature independent of the number of blocks n and linearly related to the number of edge servers s. Hence, both the computation overheads of the app vendor and edge servers are lightweight in our EDI-SA.

Then, we compare the communication overhead of EDI-V, ICL-EDI, and EDI-SA. As shown in Table 4, we can see that the communication overheads of the three schemes are relatively small, which is in line with the lightweight goal of the edge storage scenarios.

Finally, the comparison of storage overhead is given in Table 5. None of the three comparison schemes considers the storage overhead

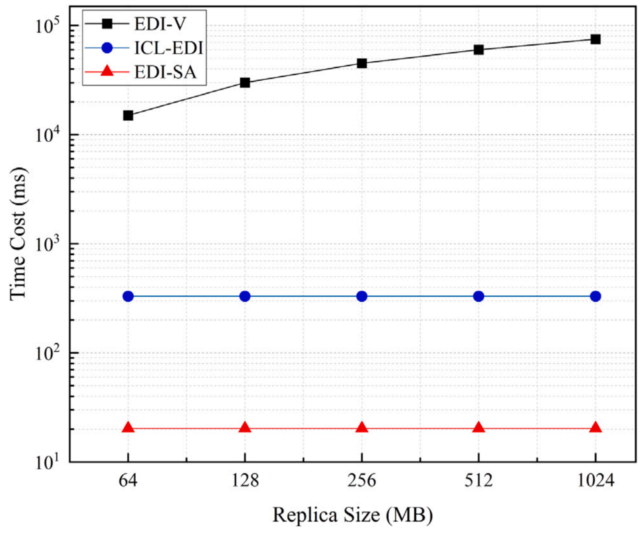

Fig. 9. Comparison of computation overhead in **ParaGen**.

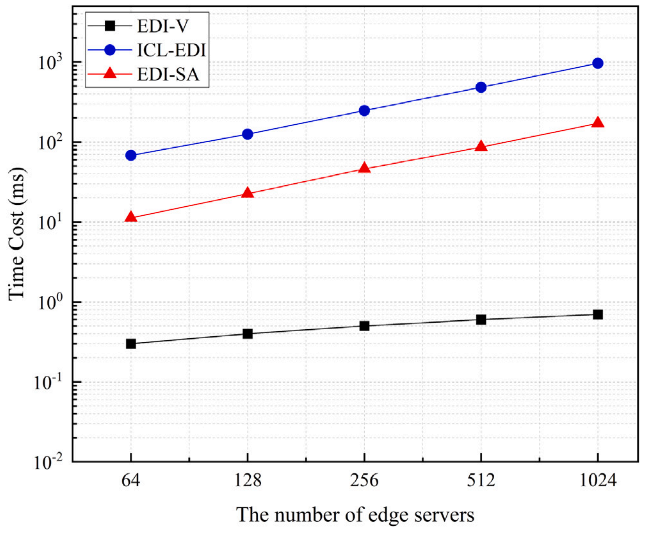

Fig. 11. Comparison of computation overhead in **ProofVer**.

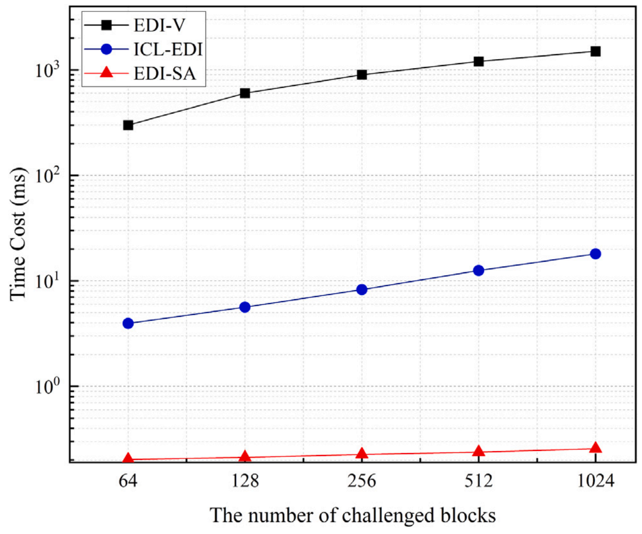

Fig. 10. Comparison of computation overhead in **ProofGen**.

of data F and its replicas stored on the app vendor and edge servers. In EDI-V, the app vendor needs to build and store a tree VMHT. When the number of the data blocks is very larger, the depth of VMHT is quite large, the search is more time-consuming and the space occupancy rate is extremely high for the app vendor. In ICL-EDI, the app vendor needs to store n homomorphic tags. What is more, neither EDI-V nor ICL-EDI supports the dynamic update of edge data replicas. Opposite to above two schemes, our EDI-SA adopts a D&CAT data structure to support data dynamics, which requires very small memory for the app vendor and edge servers.

7.3. Experiment results

All the experiments are simulated by the configuration of Intel(R) Core TM i7-6700HQ CPU @2.60 GHz, 16 GB RAM and Windows 10 Enterprise LTSC operating system. We experimentally implement the comparison of computation overhead of EDI-V [6], ICL-EDI [24] and our EDI-SA. These experimental results are the average values of 100 trials to assess the effectiveness of our EDI-SA.

Fig. 9 gives the computation overhead of the three schemes in **ParaGen** algorithm, which is performed by the app vendor. When replica size increases, the number of blocks of the replica increases. Since the app vendor in EDI-V needs to spend more time to construct a deeper VMHT according to the size of data replica F, the computation overhead increases with the data size (the replica size ranges from 64M to 1G in Fig. 9). On the contrary, the app vendors in ICL-EDI and our EDI-SA only need to generate some parameters necessary for the auditing process. According to the current security requirement of RSA, ICL-EDI requires the public key at least to be 4096 bits. While our EDI-SA needs a smaller finite field to generate a pseudorandom permutation ϕ, a pseudorandom function $Per()$ and a uniform data segmentation algorithm DSA, this requires relatively less computation overhead than ICL-EDI.

The comparison of computation overhead of the three schemes in **ProofGen** algorithm are given in Fig. 10, which is performed by edge servers to generate integrity proofs. Since the computation overhead is linearly related to the number of challenged blocks in EDI-V and ICL-EDI, we set the number of challenged blocks ranges from 64 to 1024. In EDI-V, each edge server generates the subtree of VMHT as an integrity proof, and in ICL-EDI each edge server uses homomorphic signature technology to generate the integrity proof. However, each edge server in our EDI-SA first aggregates all challenged blocks, and then generates an algebraic signature by setting $\alpha = 5$ in order to facilitate experimental simulation, which makes the computation overhead of the edge server independent of the number of challenged blocks. Therefore, the computation overhead of our EDI-SA is basically constant, and a slight increase is the computation overhead caused by aggregating.

The computation overhead of the three schemes in **ProofVer** algorithm is compared in Fig. 11, which is performed by the app vendor to verify the integrity proofs. Since the computation overhead of the app vendor in **ProofVer** algorithm is related to the number of edge servers, we set the number of edge servers ranges from 64 to 1024. In EDI-V, the computation overhead of the app vendor is relatively low because they only needs to match the integrity proofs returned by all edge servers without additional computations. In ICL-EDI and our EDI-SA, the app vendor computes some values that help him determine whether the integrity proofs returned by edge servers are correct or not, and the computation overhead of two schemes is linearly related to the number of edge servers in this process.

In summary, the experimental results show that our EDI-SA comprehensively has less computation overhead compared with other related work.

8. Conclusion

In this paper, we propose an edge data integrity auditing scheme (EDI-SA) to help app vendors check the integrity of a large number of edge data replicas that cached on the distributed edge servers. First, based on algebraic signature and the improved sampling algorithm, EDI-SA can aggregate the integrity proofs returned by edge servers and efficiently verify whether the edge data replicas are intact or not. To make the EDI-SA more lightweight, the cost of algebraic signature is independent with the number of data blocks. Besides, EDI-SA also supports batch auditing, dynamic update of the edge data, and provable updating. The security analyses and performance analyses demonstrate the security and effectiveness of our proposed scheme. In the future, we will consider decentralization and collaborative self-auditing scheme for multiple edge server scenarios.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] M. Satyanarayanan, The emergence of edge computing, Computer 50 (2017) 30–39, http://dx.doi.org/10.1109/MC.2017.9.

[2] T. Wang, L. Qiu, A.K. Sangaiah, A. Liu, Y. Ma, Edge computing based trustworthy data collection model in the internet of things, IEEE Internet Things J. 7 (2020) 4218–4227, http://dx.doi.org/10.1109/JIOT.2020.2966870.

[3] T. Zhang, Y. Li, C. Chen, Edge computing and its role in industrial internet: Methodologies, applications, and future directions, Inform. Sci. 557 (2021) 34–65, http://dx.doi.org/10.1016/j.ins.2020.12.021.

[4] X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek, H. Jin, Online collaborative data caching in edge computing, IEEE Trans. Parallel Distrib. Syst. 32 (2021) 281–294, http://dx.doi.org/10.1109/TPDS.2020.3016344.

[5] J. Wu, Y. Li, F. Ren, B. Yang, Robust and auditable distributed data storage with scalability in edge computing, Ad Hoc Netw. 117 (1) (2021) 102494, http://dx.doi.org/10.1016/j.adhoc.2021.102494.

[6] B. Li, F. Chen, H. Jin, Y. Xiang, Q. He, Auditing cache data integrity in the edge computing environment, IEEE Trans. Parallel Distrib. Syst. 32 (2021) 1210–1223, http://dx.doi.org/10.1109/TPDS.2020.3043755.

[7] J.C. Nobre, A.M. de Souza, D. Rosário, C. Both, L.A. Villas, E. Cerqueira, T. Braun, M. Gerla, Vehicular software-defined networking and fog computing: Integration and design principles, Ad Hoc Netw. 82 (2019) 172–181, http://dx.doi.org/10.1016/j.adhoc.2018.07.016.

[8] J. Shen, D. Liu, D. He, X. Huang, Y. Xiang, Algebraic signatures-based data integrity auditing for efficient data dynamics in cloud computing, IEEE Trans. Sustain. Comput. 5 (2020) 161–173, http://dx.doi.org/10.1109/TSUSC.2017.2781232.

[9] T. Hui, Y. Chen, C.C. Chang, J. Hong, Y. Huang, Y. Chen, L. Jin, Dynamic-hash-table based public auditing for secure cloud storage, IEEE Trans. Serv. Comput. 10 (2017) 701–714, http://dx.doi.org/10.1109/TSC.2015.2512589.

[10] J. Li, Z. Lei, J.K. Liu, H. Qian, Z. Dong, Privacy-preserving public auditing protocol for low-performance end devices in cloud, IEEE Trans. Inf. Forensics Secur. 11 (1) (2016) 2572–2583, http://dx.doi.org/10.1109/TIFS.2016.2587242.

[11] J. Han, Y. Li, W. Chen, A lightweight and privacy-preserving public cloud auditing scheme without bilinear pairings in smart cities, Comput. Stand. Interfaces 62 (2) (2019) 84–97, http://dx.doi.org/10.1016/j.csi.2018.08.004.

[12] X. Zhang, H. Wang, C. Xu, Identity-based key-exposure resilient cloud storage public auditing scheme from lattices, Inform. Sci. 472 (2019) 223–234, http://dx.doi.org/10.1016/j.ins.2018.09.013.

[13] Y. Shu, M. Dong, K. Ota, J. Wu, S. Liao, Binary reed-solomon coding based distributed storage scheme in information-centric fog networks, in: 2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks, Vol. 23, CAMAD, 2018, pp. 1–5, http://dx.doi.org/10.1109/CAMAD.2018.8514998.

[14] A. Aral, T. Ovatman, A decentralized replica placement algorithm for edge computing, IEEE Trans. Netw. Serv. Manag. 15 (2) (2018) 516–529, http://dx.doi.org/10.1109/TNSM.2017.2788945.

[15] S. Yi, Z. Qin, Q. Li, Security and Privacy Issues of Fog Computing: A Survey, Vol. 9024, Springer, Cham, 2015, pp. 685–695, http://dx.doi.org/10.1007/978-3-319-21837-3_67.

[16] J. Chang, B. Shao, Y. Ji, G. Bian, Efficient identity-based provable multi-copy data possession in multi-cloud storage, revisited, IEEE Commun. Lett. 24 (2020) 2723–2727, http://dx.doi.org/10.1109/LCOMM.2020.3013280.

[17] Y. Su, Y. Li, B. Yang, Y. Ding, Decentralized self-auditing scheme with errors localization for multi-cloud storage, IEEE Trans. Dependable Secure Comput. (2021) 1, http://dx.doi.org/10.1109/TDSC.2021.3075984.

[18] R. Curtmola, O. Khan, R. Burns, G. Ateniese, MR-PDP: Multiple-replica provable data possession, in: Distributed Computing Systems, 2008. ICDCS '08. the 28th International Conference on, 2008, pp. 411–420, http://dx.doi.org/10.1109/ICDCS.2008.68.

[19] R. Masood, N. Pandey, Q.P. Rana, DHT-PDP: A distributed hash table based provable data possession mechanism in cloud storage, in: 2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions), ICRITO, 2020, pp. 275–279, http://dx.doi.org/10.1109/ICRITO48877.2020.9198019.

[20] X. Yang, X. Pei, M. Wang, T. Li, C. Wang, Multi-replica and multi-cloud data public audit scheme based on blockchain, IEEE Access 8 (2020) 144809–144822, http://dx.doi.org/10.1109/ACCESS.2020.3014510.

[21] Z. Liu, Y. Liu, X. Yang, X. Li, Integrity auditing for multi-copy in cloud storage based on red-black tree, IEEE Access 9 (2021) 75117–75131, http://dx.doi.org/10.1109/ACCESS.2021.3079143.

[22] H. Tian, F. Nan, C. Chang, Y. Huang, J. Lu, Y. Du, Privacy-preserving public auditing for secure data storage in fog-to-cloud computing, J. Netw. Comput. Appl. 127 (2019) 59–69, http://dx.doi.org/10.1016/j.jnca.2018.12.004.

[23] W. Tong, B. Jiang, F. Xu, Q. Li, S. Zhong, Privacy-preserving data integrity verification in mobile edge computing, in: 2019 IEEE 39th International Conference on Distributed Computing Systems, ICDCS, 2019, pp. 1007–1018, http://dx.doi.org/10.1109/ICDCS.2019.00104.

[24] G. Cui, Q. He, B. Li, X. Xia, Y. Yang, Efficient verification of edge data integrity in edge computing environment, IEEE Trans. Serv. Comput. (2021) 1, http://dx.doi.org/10.1109/TSC.2021.3090173.

[25] B. Li, Q. He, F. Chen, H. Jin, Y. Xiang, Y. Yang, Inspecting edge data integrity with aggregated signature in distributed edge computing environment, IEEE Trans. Cloud Comput. (2021) 1, http://dx.doi.org/10.1109/TCC.2021.3059448.

[26] B. Li, Q. He, F. Chen, H. Dai, H. Jin, Y. Xiang, Y. Yang, Cooperative assurance of cache data integrity for mobile edge computing, IEEE Trans. Inf. Forensics Secur. 16 (2021) 4648–4662, http://dx.doi.org/10.1109/TIFS.2021.3111747.

[27] W. Litwin, T. Schwarz, Algebraic signatures for scalable distributed data structures, in: The 20th International Conference on Data Engineering, 2004, pp. 412–423, http://dx.doi.org/10.1109/ICDE.2004.1320015.

[28] Q. Fu, L. Chen, J. Li, Z. Yao, Efficient divide and conquer adjacency tables based dynamic integrity auditing, J. Cryptol. Res. 8 (2021) 601–615, http://dx.doi.org/10.13868/j.cnki.jcr.000462.

Liping Qiao received the B.S. degree from Tiangong University in 2020. She is currently pursuing the M.S. degree from Shaanxi Normal University, Xi'an, China. Her research interests include cryptography and its applications.

Yanping Li received the M.S. degree from Shaanxi Normal University in 2004 and the Ph.D. degree from Xidian University in 2009. She is currently an associate professor of Shaanxi Normal University, Xi'an, China. Her research interests include cryptography and its applications.