Postprint of a Multi-Replica Data Integrity Verification Scheme Supporting Dynamic Operations

Authors: Liu Hongyu, Ding Yiwen, Leiting Chen

Date: 2018-06-19T00:00:00+00:00

Abstract

To address the security threats faced by outsourced data in cloud storage environments and to overcome the limitations of existing cloud data integrity verification schemes, we propose a multi-replica data integrity verification scheme that supports dynamic operations. The scheme considers multi-replica application scenarios and achieves multi-replica verification for files with relatively low overhead based on existing cloud data integrity verification schemes. By introducing an authenticated data structure—the rank-based Merkle Hash Tree—it enables verifiable dynamic updates of files. By associating multiple replicas, synchronous updates of multiple replicas can be achieved. Security analysis and experiments demonstrate the security and effectiveness of the scheme, which achieves secure storage and updating of data and effectively guarantees the privacy and security of data replicas.

Full Text

Preamble

Title: Multiple-Replica Data Integrity Checking Protocol with Efficient Dynamic Update in Cloud Storage

Authors: Liu Hongyu[1], Ding Yiwen[2], Chen Leiting[1] [1]School of Computer Science & Engineering, [2]School of Economics & Management, University of Electronic Science & Technology of China, Chengdu 611731, China

Abstract: This paper proposes a multiple-replica remote data integrity checking protocol with efficient data dynamic update in cloud storage, addressing the security risks of outsourced data in cloud environments and the limitations of existing cloud data integrity verification schemes. The scheme is suitable for multiple-replica scenarios and achieves multiple-replica provable data possession at a small cost on the basis of existing cloud data integrity checking schemes. By introducing an authenticated data structure called rank-based Merkle Hash

Tree (rMHT), the protocol supports full data dynamic operations. Through replica correlation, multiple replicas can be updated synchronously. Security analysis and experimental results demonstrate the security and effectiveness of the proposed protocol, which can guarantee the privacy of multiple replicas while ensuring secure data storage and updates.

Keywords: multiple-replica; data integrity checking; dynamic update; cloud storage

0 Introduction

Cloud computing as a new computing paradigm enables users to obtain computing and storage resources on demand. Through virtualization technology, users can conveniently access shared resource pools composed of distributed computing nodes, saving hardware procurement costs and data maintenance overhead. As the most fundamental and widely applied service in cloud computing, cloud storage has become a focal point for researchers. However, the separation of data usage rights and management rights introduces security risks that pose major obstacles to cloud computing adoption. Highly centralized storage resources expose cloud servers to serious security challenges—external attacks or internal misoperations by cloud service providers can compromise data integrity.

Data integrity verification mechanisms enable users to verify data stored in the cloud and promptly detect corruption. Current cloud data integrity verification protocols can be categorized into Provable Data Possession (PDP) and Proof of Retrievability (PoR) based on whether they can recover original data. PDP protocols can check cloud data integrity without downloading entire files, while PoR protocols allow data recovery in addition to integrity verification. The primary difference is that PoR mechanisms use error-correcting codes for original data recovery.

Ateniese et al. [1] first proposed the PDP model in 2007, which utilizes sampling technology to achieve data integrity verification while significantly reducing computational and communication overhead for users. They also presented two efficient and provably secure PDP schemes based on RSA signatures. Subsequently, PDP protocols based on short signature technology emerged [2], offering shorter signatures under the same security conditions. These signatures possess homomorphic properties, enabling aggregation of multiple signatures into one. Short-signature PDP mechanisms support public verification, allowing users to delegate audit tasks to a Third-Party Auditor (TPA), thereby reducing storage overhead and communication costs.

Considering that users may need to modify data stored in the cloud, PDP schemes must support dynamic operations. To address this, Wang et al. and Erway et al. [3,4] utilized dynamic data structures to ensure data block positions. Wang et al. improved the PoR model using the classic Merkle Hash Tree (MHT), while Erway et al. employed a modified authenticated skip list

data structure to enable data dynamic updates, though this approach incurs substantial computational and communication overhead during authentication.

Storing multiple data replicas in the cloud is a widely adopted method to improve data reliability and availability. However, cloud service providers might not store all replicas to save storage space and maximize profits, significantly impairing data availability. Curtmola et al. [5] addressed integrity verification for multiple replicas, enabling verification of multiple copies with small computational overhead but without supporting dynamic data operations. Liu et al. [6] proposed a cloud data integrity verification protocol supporting dynamic operations based on a multi-replica Merkle hash tree, where Merkle tree levels are generated top-down and replica blocks are constructed within identical replica subtrees. However, this approach is not designed for distributed storage environments; for high data availability, replicas should be stored across multiple geographically distributed servers, as disk failures could destroy all replicas otherwise. Additionally, it processes each replica separately, with each replica block having its own authentication tag, increasing user computational overhead.

Recent extensive research on cloud data integrity protocols has yielded numerous PDP variants [7–13] that implement more features than original PDP protocols [14–16]. In 2015, He et al. [17] considered data integrity verification in wireless body area networks and proposed a certificateless public auditing scheme. In 2016, Zhang et al. [18] proposed an efficient identity-based public auditing scheme for cloud storage. Yu et al. [19] presented an identity-based data integrity verification scheme with perfect privacy preservation. In 2017, Yan et al. [20] proposed a remote data auditing protocol supporting dynamic operations based on homomorphic hash functions, though it does not support multiple-replica storage. Li et al. [21] proposed a data integrity verification method based on relative index hash trees.

This paper proposes a multiple-replica data integrity verification scheme supporting dynamic operations for multi-replica cloud storage scenarios. By symmetrically encrypting the original file to generate ciphertext and securely generating multiple replicas from this ciphertext, the scheme achieves multi-replica storage and verification with low computational overhead. By introducing the rank-based MHT authenticated data structure, it enables verifiable dynamic updates including data block modification, insertion, and deletion. Correlating replicas stored across different cloud servers enables synchronous updates of multiple replicas. Finally, security analysis and performance testing demonstrate the scheme' s effectiveness and practicality.

1.1 Bilinear Pairings

Let G_1 and G_2 be two multiplicative cyclic groups of prime order p, where G is a cyclic group formed by discrete points on an elliptic curve. A bilinear pairing $\hat{e} : G_1 \times G_2 \to G_T$ is a mapping with the following properties:

a) **Computational Efficiency**: For all $h_1, h_2 \in G$, there exists an efficient algorithm to compute $\hat{e}(h_1, h_2)$.

b) **Bilinearity**: For all $h_1, h_2 \in G$ and $a, b \in Z_p$, we have $\hat{e}(h_1^a, h_2^b) = \hat{e}(h_1, h_2)^{ab}$.

c) **Non-degeneracy**: $\hat{e}(g, g) \neq 1$, where g is a generator of G.

1.2 Merkle Hash Tree

The Merkle Hash Tree (MHT) is a widely used authenticated structure that can effectively verify whether a series of elements are correctly stored, making it suitable for data integrity checking. An MHT is a binary tree where leaf nodes contain hashes of stored data, and each internal node stores the hash of its two children. Each internal node and the root node are generated from their child nodes. For each element a_i, its Auxiliary Authentication Information (AAI) is the set of sibling nodes on the path from a_i to the root. Thus, given element a_i and its AAI, the root node can be computed. Assuming the verifier knows the MHT root, to check the integrity of the i-th data block m_i, the verifier needs both the data itself and its AAI. The one-way hash function used to build the MHT ensures authentication correctness and reliability, allowing the MHT to authenticate data block values but not their positions.

1.3 Rank-Based Merkle Hash Tree

To adapt MHT for authenticated data structures supporting dynamic operations, a novel data structure called rank-based MHT (rMHT) has been developed. In rMHT, each node v stores, in addition to its hash value h_v, the number of leaf nodes it can reach. This value is called the rank of node v. Specifically, the rank of a non-leaf node equals the sum of the leaf node ranks of its children. Unlike MHT, in rMHT, the value h_v is computed by concatenating the values of its two child nodes with its rank and then hashing. Another value stored in the node is its height, denoted by h_v^h, and the last value is edge information, indicating whether it is the left or right child of its parent, represented by 0 and 1 respectively. The edge information is denoted by v_s. Therefore, in rMHT, node v is represented as $v = (v_v, v_h, v_r, v_s)$.

More precisely, assuming node v has child nodes v_l (left child) and v_r (right child), the value of node v can be calculated as follows:

$$h_v = H(h_{v_l} || h_{v_r} || v_r)$$

The root node R does not contain edge information, so v_s is empty. In rMHT, the AAI for the i-th authenticated data block a_i is $\Omega_i = \{(v_v, v_h, v_r, v_s) | (v_v, v_h, v_r, v_s) = v_j, 1 \leq j \leq k\}$. Since nodes in the AAI are at different heights, they must be arranged from bottom to top by height, which facilitates efficient root computation from the bottom up.

[Figure 1: see original paper] provides an example of rMHT construction. For leaf nodes, $h_i = H(m_i||1)$, where 1 is the rank of the leaf node. For internal node c, $h_c = H(h_1||h_2||2)$, where 2 is the rank of node c because it has two leaf children. Similarly, nodes d, e, and f each have rank 2. Nodes a and b have rank 4 because they each have four leaf children. The root node r has rank 8.

2.2 Security Threats

After data owners store data in cloud servers and delete local files, data security completely depends on the cloud server. However, data stored in the cloud faces numerous security risks. First, data may be lost or corrupted due to improper operations by cloud provider insiders or attacks by external adversaries, and to protect their reputation, cloud providers may conceal data loss incidents. Second, cloud providers may delete infrequently accessed data to save storage space, and when users store multiple file replicas, cloud servers might store only one copy while colluding to make users believe all replicas are stored.

2.3 Design Goals

To ensure secure data storage and dynamic updates, the scheme should achieve two design objectives:

a) **File Integrity Verification**: Data owners can verify file integrity with minimal overhead. They can perform either single-replica verification or multi-replica verification to ensure the integrity and availability of multiple replicas.

b) **Verifiable Multi-Replica Updates**: When users modify, insert, or delete data stored in the cloud, they can verify that the cloud server has performed the corresponding operations correctly.

3.1 Scheme Overview

A multi-replica PDP scheme supporting dynamic operations consists of six algorithms: KeyGen, Store, Challenge, ProofGen, ProofVerify, and Verifiable Dynamic Update.

KeyGen: A probabilistic algorithm run by the data owner to generate public and private keys.

Store: A probabilistic algorithm run by the data owner to generate t replicas of file F. The data owner first encrypts the original file to produce ciphertext, then masks it with random numbers to generate multiple replicas. The replicas are stored on servers. The data owner then generates authentication tags for the file by dividing it into blocks and sectors, creating tags for each block, building an rMHT for the file, and signing the root. For all replicas of the file, the data owner generates only one set of authentication tags. Finally, the data owner

uploads the replicas and their authentication tags to cloud servers and deletes local copies.

Challenge: To check data integrity on cloud servers, the data owner initiates a challenge to the cloud, which can be either single-replica or multi-replica. For single-replica challenges, the data owner interacts with a specific cloud server to verify whether it correctly stores replica F_u. For multi-replica challenges, the data owner can challenge multiple servers storing replicas to verify the availability of all replicas. By challenging only a random subset of data blocks, the data owner can detect data loss or corruption with high probability.

ProofGen: A deterministic algorithm run by the cloud server. Upon receiving a challenge, the cloud server generates a proof based on the challenge request and sends it to the data owner.

ProofVerify: A deterministic algorithm run by the data owner. Upon receiving the proof from the cloud server, the data owner verifies it to determine the integrity of data stored in the cloud.

Verifiable Dynamic Update: A deterministic algorithm run by both data owner and cloud server. When the data owner modifies, inserts, or deletes data in the cloud, the cloud server performs operations on all replicas according to the update request and returns update evidence, enabling the data owner to verify the update.

3.2 Concrete Construction

Let $\hat{e} : G_1 \times G_2 \to G_T$ be a bilinear map, where G_1 and G_2 are multiplicative cyclic groups, $ord(G_1) = ord(G_2) = ord(G_T) = p$, and p is a large prime. Let g be a generator of G_1. Let $H : \{0,1\}^* \to G_1$ be a collision-resistant hash function for building rMHT. Let $\varphi : \{0,1\}^* \to Z_p$ be a secure pseudo-random function.

KeyGen: The data owner first randomly selects $\alpha \in Z_p$ and computes $v = g^\alpha$. Then it generates a random signature key pair (spk, ssk), produces an AES encryption key K, and a pseudo-random function key z. The data owner' s secret key is $sk = (\alpha, ssk, K, z)$, and the public key is $pk = (v, spk)$.

Store: The data owner first divides the original file F into n blocks: $F = \{f_1, f_2, ..., f_n\}$. It encrypts the original file with key K to obtain $F' = \{b_1, b_2, ..., b_n\}$, where $b_i = E_K(f_i)$ for $1 \le i \le n$. Then the data owner generates t different file replicas as follows: For replica F_u $(1 \le u \le t)$, it uses the pseudo-random function φ with key z to generate random $r_u = \varphi_z(u)$. The data blocks in F_u are generated by operating each data block of F' with random value r_u: $F_u = \{m_{u,1}, m_{u,2}, ..., m_{u,n}\}$, where $m_{u,i} = b_i + r_u$ for $1 \le i \le n$.

The data owner divides each file block into s sectors: $b_i = \{b_{i,1}, b_{i,2}, ..., b_{i,s}\}$. It randomly selects a file name $fname$ from $\{0,1\}^*$ and s random elements $u_1, u_2, ..., u_s \in G_1$. Then it generates authentication tags for each block i of file F: $T_i = \{T_{i,1}, T_{i,2}, ..., T_{i,s}\}$, where $T_{i,j} = (H(fname||u) \cdot \prod_{j=1}^{s} u_j^{b_{i,j}})^\alpha$. The tag

set is $T = \{T_i\}_{1 \le i \le n}$. The file tag is $t = fname||SSig_{ssk}(fname||u_1||...||u_s)$, where $SSig_{ssk}$ is the signature with secret key ssk.

For all replicas F_u, the data owner generates only one authentication tag set T. It then builds an rMHT for file F, signs the root R as $sig_{ssk}(h_R)$, and uploads replicas F_u, tag set T, and file tag t to cloud server S_u. Finally, the data owner deletes the original file, all replicas, and tag sets locally. The cloud server establishes a replica catalog for uploaded data containing all replicas' Logical File Names (LFN) and Physical File Names (PFN). LFN is the file name $fname$, which is identical for all replicas, while PFN consists of the physical storage path and cloud server ID. The replica catalog is sent to the data owner.

Challenge: The data owner randomly selects l indices to form challenge set $Q = \{(i, v_i)\}$, where $i \in [1, n]$ and $v_i \in Z_p$. The data owner sends challenge Q for replica F_u to the corresponding cloud server.

ProofGen: Upon receiving challenge Q, the cloud server computes for each $i \in Q$:

$$\mu_{i,j} = \sum_{(i,v_i) \in Q} v_i \cdot m_{i,j}$$

$$\sigma = \prod_{(i,v_i) \in Q} T_i^{v_i}$$

Then the cloud server sends $(\mu, \sigma, \{T_i\}_{i \in Q}, \Omega)$ to the data owner, where $\mu = \{\mu_j\}_{1 \le j \le s}$ and Ω is the AAI of challenged nodes.

ProofVerify: The data owner first verifies the signature in file tag t. If invalid, it aborts. Otherwise, it extracts $fname$ from t and verifies:

$$\hat{e}(\sigma, g) \stackrel{?}{=} \hat{e}\left(\prod_{(i,v_i) \in Q} H(fname||i)^{v_i} \cdot \prod_{j=1}^{s} u_j^{\mu_j}, v\right)$$

Simultaneously, the data owner verifies the positions of challenged blocks via rMHT. By verifying that the sum of rank values of nodes with edge information "left" in a node' s AAI equals the correct number of leaf nodes to its left, the data owner can determine correct node positions. Let CoR denote the number of leaf nodes to the left of (including) the node being verified. For example, in [Figure 1: see original paper], to verify data block m_4' s value and position, the data owner receives AAI $\Omega_4 = \{(h_3, 1, 1, 0), (h_c, 2, 2, 0), (h_b, 3, 4, 1)\}$. It sets CoR = 1 and performs:

a) Verify $h_4 = H(m_4||1)$ and update CoR = CoR + 1 = 2.

b) Verify $h_d = H(h_3||h_4||2)$ and update CoR = CoR + 2 = 4.

c) Verify $h_b = H(h_e||h_f||4)$.

d) Verify $h_R = H(h_a||h_b||8)$ and check that CoR = 4.

When verification equation (9) holds and all challenged node positions are correct, the data stored on the cloud server is confirmed intact.

Verifiable Dynamic Update: We consider three common dynamic operations: modification, insertion, and deletion.

Modification replaces a specific data block with a new one. To modify block i to m_i^*, the data owner sends (i, m_i^*, T_i^*) to the cloud server. The server first updates the hash of leaf node i, then updates all nodes on the path from this leaf to the root, and returns the new root value h'_R and updated path nodes Ω'_i.

Insertion adds a new data block to the original block set. To insert block m^* at position i (before m_i), the data owner sends (i, m^*, T^*). The cloud server calculates the new node' s hash and rank, computes parents of m_i and m^*, recalculates values for sibling nodes in m_i' s AAI, and outputs the new root h'_R and path nodes Ω'_i.

Deletion removes a specific data block. To delete block i, the data owner sends (i, Del). The cloud server updates all path nodes, increments the level of the deleted node' s sibling by 1, updates its edge information, updates sibling nodes in m_i' s AAI, and outputs the new root h'_R and new path nodes Ω'_{i-1}.

To verify that the server performed updates correctly, the data owner retrieves necessary inputs (e.g., node AAI) from the cloud server, verifies their correctness, runs the update algorithm to compute the new root h''_R, and compares it with the updated root h'_R from the server. If they match, the server correctly updated the data.

When dynamically updating all replicas, the data owner sends update commands for all replicas to one server storing a replica. That server forwards the commands to all physical addresses storing the file according to the replica catalog, and each server updates its copy accordingly. The data owner can randomly interact with any cloud server to verify the update operation.

4.1 Functional Analysis

The proposed scheme is suitable for cloud users who store multiple replicas of files on cloud servers. Users generate only one authentication tag per file, enabling integrity verification of different replicas stored across various servers without generating separate tags for each replica, while ensuring cloud servers cannot deceive users by storing only one copy. Users can interact with multiple servers to verify each replica' s integrity.

When users dynamically update files, they send update requests to cloud servers storing replicas. After updating, the cloud server returns generated evidence, which users can verify to achieve verifiable dynamic updates. Since cloud servers store replica catalogs, they can send data update requests to other cloud servers storing file replicas, enabling synchronous updates across all replicas. Users can verify dynamic updates by interacting with any replica-storing server.

4.2 Security Analysis

Under our security assumptions, data owners will not actively leak key information, and the pseudo-random function used for replica generation is secure. The symmetric encryption algorithm (AES) used for original files has attack complexity dependent on block and key length, offering both high efficiency and security. Replica security relies on AES encryption and the pseudo-random function.

For the integrity verification process, Ateniese et al. [1] proved that if the digital signature scheme used for file tags is existentially unforgeable and the Computational Diffie-Hellman problem is hard in bilinear groups, then no polynomial-time adversary can break the scheme' s reliability in the random oracle model —i.e., adversaries cannot forge proofs that pass verification. Data owners can verify $sig_{ssk}(h_R)$ and AAI Ω_i correctness. Using h_R and challenged blocks, data owners can compute CoR to ensure correct data positions, resisting replacement attacks. By guaranteeing correct data values and positions, data owners can be confident that data stored on cloud servers is intact, and adversaries cannot forge verifiable proofs in polynomial time, ensuring scheme security.

5 Implementation Results and Analysis

This section evaluates the proposed scheme' s performance experimentally. The test platform uses an Intel Core(TM) i7-7500 @2.70 GHz processor, 8 GB RAM, and Win10 64-bit OS. Algorithms were implemented in Visual Studio 2012 with the Miracl library, a gold standard for elliptic curve cryptography. The security parameter was set to $\lambda = 80$, meeting security requirements. Performance is demonstrated from two aspects:

a) **Tag Generation Overhead**: We tested tag generation time as file size increased from 1 KB to 10 KB. Each sector is 160 bits, with each test increasing file size by 1 KB. According to the tag generation algorithm, when file size is fixed, more sectors per block means fewer blocks, so total tag generation time remains essentially constant. As file size increases, the total number of sectors increases, and tag generation time grows accordingly. [Figure 2: see original paper] shows tag generation time versus file size. As shown, tag generation time increases linearly with file size, consistent with analytical results; for a 10 KB file, tag generation takes approximately 0.95 seconds.

b) **Integrity Verification Performance**: We measured performance overhead for both data owners and cloud servers during integrity verification. File size was fixed at 1 MB while the number of challenged blocks increased from 50 to 500. [Figure 3: see original paper] shows proof generation and verification time versus number of challenged blocks. As challenged blocks increase, cloud server computational overhead grows gradually from 16 ms to 165 ms, while data owner verification overhead remains stable at approximately 193 ms. This is because server proof generation time increases

with challenged blocks, while data owner verification time increases only marginally. Ateniese et al. proved that if 1% of data is corrupted on the cloud server, challenging 300 blocks detects corruption with 95% probability, and challenging 460 blocks achieves 99% detection probability. As shown, challenging 300 blocks requires 99 ms for proof generation, and challenging 460 blocks requires 152 ms, with data owner verification time at 194 ms in both cases.

6 Conclusion

This paper designed a multiple-replica data integrity verification scheme supporting dynamic operations in cloud storage environments, enabling data owners to achieve multi-replica storage and integrity verification with low computational overhead while supporting dynamic data operations. The authenticated rMHT data structure ensures correct data block positions and enables verifiable dynamic updates. Security analysis demonstrates that the scheme meets security requirements, and experimental results confirm its efficiency and practicality. Future work will focus on designing a publicly verifiable multi-replica data integrity verification scheme for cloud environments that supports verifiable dynamic operations, with improved computational efficiency, reduced communication overhead, and greater flexibility.

References

[1] Ateniese G, Burns R C, Curtmola R, et al. Provable data possession at untrusted stores [C]// Proc of ACM CCS' 07. 2007: 598-609.

[2] Dan B, Lynn B, Shacham H. Short signatures from the weil pairing [J]. Journal of Cryptology, 2004, 17 (4): 297-319.

[3] Wang Qian, Wang Cong, Ren Kui, et al. Enabling public auditability and data dynamics for storage security in cloud computing [C]// Proc of ESORICS' 09. 2009: 355-370.

[4] Erway C, Alptekin K, Papamanthouc C. Dynamic provable data possession [C]// Proc of ACM Conference on Computer and Communications Security. 2009: 213-222.

[5] Curtmola R, Khan O, Burns R, et al. MR-PDP: multiple-replica provable data possession [C]// Proc of IEEE International Conference on Distributed Computing Systems. 2008: 411-420.

[6] Liu Chang, Ranjan R, Yang Chi, et al. MuR-DPA: top-down levelled multi-replica merkle hash tree based secure public auditing for dynamic big data storage on cloud [J]. IEEE Trans on Computers, 2015, 64 (9): 2609-2622.

[7] Wang Qian, Wang Cong, Ren Kui, et al. Enabling public auditability and data dynamics for storage security in cloud computing [J]. IEEE Trans on Parallel and Distributed Systems, 2011, 22 (5): 847-859.

[8] Wang Cong, Ren Kui, Lou Wenjing, et al. Toward public auditable secure cloud data storage services [J]. Network IEEE, 2010, 24 (4): 19-24.

[9] Zhu Yan, Hu Hongxin, Ahn G J, et al. Efficient audit service outsourcing for data integrity in clouds [J]. Journal of System and Software, 2012, 85 (5): 1083-1095.

[10] Zhu Yan, Hu Hongxin, Ahn G J, et al. Cooperative provable data possession for integrity verification in multicloud storage [J]. IEEE Trans on Parallel and Distributed Systems, 2012, 23 (12): 2231-2244.

[11] Yang Kan, Jia Xiaohua. An efficient and secure dynamic auditing protocol for data storage in cloud computing [J]. IEEE Trans on Parallel and Distributed Systems, 2013, 24 (9): 1717-1726.

[12] Zhu Yan, Wang Shangbiao, Hu Hongxin, et al. Secure collaborative integrity verification for hybrid cloud environments [J]. International Journal of Cooperative Information Systems, 2012, 21 (3): 165-198.

[13] Wang Cong, Chow S S M, Wang Qian, et al. Privacy-preserving public auditing for secure cloud storage [J]. IEEE Trans on Computers, 2013, 62 (2): 362-375.

[14] Juels A, Kaliski B S. PORs: proofs of retrievability for large files [C]// Proc of ACM CCS' 07. 2007: 584-597.

[15] Shacham H, Waters B. Compact proofs of retrievability [C]// Proc of Asiacrypt' 08. 2008: 90-107.

[16] Shacham H, Waters B. Compact proofs of retrievability [J]. Journal of Cryptology, 2013, 26 (3): 442-483.

[17] He Debiao, Zeadally S, Wu Libing. Certificateless public auditing scheme for cloud-assisted wireless body area networks [J]. IEEE Systems Journal, 2015, PP (99): 1-10.

[18] Zhang Jianhong, Dong Qiaocui. Efficient ID-based public auditing for the outsourced data in cloud storage [J]. Journal of Clinical Ultrasound, 2016, 6 (6): 1-14.

[19] Yu Yong, Man H A A, Ateniese G, et al. Identity-based remote data integrity checking with perfect data privacy preserving for cloud storage [J]. IEEE Trans on Information Forensics & Security, 2017, PP (99): 1-1.

[20] Yan Hao, Li Jiguo, Han Jinguang, et al. A novel efficient remote data possession checking protocol in cloud storage [J]. IEEE Trans on Information Forensics & Security, 2017, 12 (1): 78-88.

[21] 李萌庭, 周安宁. 云环境中基于相对索引散列树的数据审核方法 [J/OL]. 计算机应用研究, 2019, 36 (4). [2018-02-07]. http://www.arocmag.com/article/02-2019-04-039.html.

[22] 田荣阳. 数据网格中的副本定位及选择服务 [D]. 重庆: 重庆大学, 2007.