International Journal of Digital Crime and Forensics
Volume 11 • Issue 1 • January-March 2019

Dynamic Provable Data Possession of Multiple Copies in Cloud Storage Based on Full-Node of AVL Tree

Min Long, Changsha University of Science and Technology, Changsha, China
You Li, Changsha University of Science and Technology, Changsha, China
Fei Peng, Hunan University, Changsha, China

ABSTRACT

This article describes how to protect the security of cloud storage, a provable data possession scheme based on full-nodes of an AVL tree for multiple data copies in cloud storage. In the proposed scheme, a Henon chaotic map is first implemented for the node calculation of the AVL tree, and then the location of the data in the cloud is verified by AVL tree. As an AVL tree can keep the balance even with multiple dynamic operations made on the data in the cloud, it can improve the search efficiency of the data block, and reduce the length of the authentication path. Simulation results and analysis confirm that it can achieve good security and high efficiency.

KEYWORDS

AVL Tree, Dynamic Operation, Information Security Integrity Verification, Multiple Copies, Provable Data Possession

1. INTRODUCTION

Recently, cloud storage has been paid wide attention for its mass storage capability and low cost (Li, Qiu, Qiu, Qiu & Zhao, 2016). However, the open application mode makes the security of cloud storage face severe challenges (Feng, Zhang, Zhang & Xu, 2011). How to protect the security of cloud storage has become an urgent problem to be resolved. Integrity verification is an important part of the data security. Multiple data copies are often used in the cloud storage to keep the reliability and availability. Dynamic operations are used to support data updating on the cloud platform. Thus, data integrity verification to support multiple data copies and dynamic operation is desirable. Currently, according to the implementation of fault-tolerance preprocessing or not, the existing data integrity verification mechanisms are classified into proof of retrievability (PoR) (Juels & Kaliski, 2007; Yan, 2013; Zhou, Li, Guo & Jia, 2014) and provable data possession (PDP) (Ateniese, Burns, Curtmola, Herring & Kissner, 2007; Erway, Küpçü, Papamanthou & Tamassia, 2009; Gritti, Susilo & Plantard, 2015; Curtmola, Khan, Burns & Ateniese, 2008; Barsoum & Hasan, 2010).

DOI: 10.4018/IJDCF.2019010110

The remote data integrity verification is firstly realized by using HMAC hash function (Deswarte, Quisquater & Saïdane,2004). The MAC value of data is stored in local, and all the data is needed to retrieve from the root node to compute the MAC value during the validation. This mechanism needs a large computation cost and communication overhead; thus, it is cannot be applied to the integrity verification of mass data in cloud storage. After that, the conception of PDP is proposed (Ateniese, Burns, Curtmola, Herring & Kissner, 2007). Homomorphic veritable tags are used to inspect the sampled data in the cloud. It allows verifying data possession without having access to the actual data file, and it achieves a low overhead, but it does not support dynamic operation to the data. Thereafter, they proposed an improved scheme (Ateniese, Pietro, Mancini & Tsudik, 2008), but only modification and deletion can be performed, and it cannot support insertion. Erway et al. (Erway, Küpçü, Papamanthou & Tamassia, 2009) used the rank value of the authentication jump table to support the dynamic operation. Gritti et al. (Gritti, Susilo & Plantard, 2015) proposed a highly efficient scheme that supports dynamic authentication and protects user privacy. These schemes are only designed for the verification of a single copy of data.

In order to solve the problem of data integrity authentication for multiple copies, a MR-PDP (Multiple-Replica PDP) scheme is proposed by Curtmola et al. (Curtmola, Khan, Burns & Ateniese, 2008). It can quickly generate multiple copies and restore the damaged copies. Barsoum & Hasan (Barsoum & Hasan, 2010) put forward a multiple copies PDP scheme for static file, but this scheme only applies single copy PDP scheme to different copies, and the efficiency is low. Homomorphic linear authenticator was used to identify the multiple copies data (Ateniese, Kamara & Katz, 2009). Fu et al. (Fu, Zhang, Chen & Feng, 2014) proposed a proof of data possession scheme of multiple copies by taking the advantages of distributed computing ability of the multiple servers, and it can verify whether the servers hold the correct number of copies or not. However, full dynamic operations are not always supported in these schemes.

In recent years, MHT (Merkle Hash Tree) is used to construct data integrity authentication schemes (Barsoum & Hasan, 2011; Barsoum & Hasan, 2013; Barsoum & Hasan, 2015). Long, Li & Peng (Long, Li & Peng, 2017) implemented spatiotemporal chaos for node calculation of the binary tree. These schemes can achieve data dynamic operation and support multiple copies authentication by manipulating the classic Merkle-Hash-Tree (MHT). However, if insertion is performed on the same data block for many times, it will lead to one branch of the binary tree too long, therefore the efficiency of verification is reduced.

In this paper, a novel integrity verification scheme for multiple data copies in cloud storage is proposed. We try to ensure the security and efficiency by engaging the AVL tree and Henon map. AVL tree is used to construct the authentication structure for its good balance characteristics, and node computation is achieved by Henon map, thus the change of location and value of the data can be quickly discovered for its good randomness and sensitivity.

2. CONCEPTS

2.1. Notations

t : the number of copy
n : the number of blocks per copy
L : the number of character per block
F : the original file, $F = \{d_1, d_2, ..., d_n\}$
F' : encrypted file by encrypting F with cryptographic function $E_{key}(\cdot)$, $F' = \{b_1, b_2, ..., b_n\}$
$r_{i,j}$: random number
$m_{i,j}$: the j^{th} block on copy i, $m_{i,j} = b_j + r_{i,j}$
F_i : the i^{th} copy, $F_i = \{m_{i,1}, m_{i,2}, ..., m_{i,n}\}_{1 \le i \le t}$

$A_{i,j,k}$: the ASCII of k^{th} character in the j^{th} block on copy i
R_i : the root of binary tree of i^{th} copy
M : metadata of the file
$Filename$: the name of the file stored in cloud
$\sigma_{i,j}$: the tag of j^{th} block on copy i
ϕ_j : the aggregated tags of j^{th} block of all copies
$E_{key}(\cdot)$: encryption function
$D_{key}(\cdot)$: decryption function

2.2. Bilinear Pairings

The bilinear map is defined as: $e: G \times G = G_T$, where G is a group of Diffie-Hellman hypotheses and G_T is the multiplicative group of prime number *P*. This map has the following three characteristics:

1. Computability: for any $h_1, h_2 \in G$, there are valid methods for computing $e(h_1, h_2)$;
2. Bilinearity: for $h_1, h_2 \in G$, $a, b \in Z_p$, $e(h_1^a, h_2^b) = e(h_1, h_2)^{ab}$;
3. non-degeneracy: when g is a generator of the cycle group G , $e(g, g) \neq 1$.

2.3. System Model

A cloud data verification system generally includes user, cloud server and TPA (Long, Li & Peng, 2017). User is the entity that possesses data/files needing to be stored in the cloud. Cloud server has strong computing power and storage capacity, which consists of two parts: a) master server S , it is used for communication between the users and scheduling the task for the storage servers. Master Server has a storage directory including $Filename$, ω_i , ϕ_j and the serial number of the storage server where the copies of the file are stored; b) storage server $S_1, S_2, ..., S_t$, and they are mainly used for files storage. In this scheme, each copy is stored in different storage server, and it means t copies of the file, $F_1, F_2, ..., F_t$ stored on t different storage servers $S_1, S_2, ..., S_t$, respectively. TPA has the expertise and capabilities that users do not have. As the proposed scheme supports public auditing, TPA has the public key and it can act as a verifier, which is helpful to reduce the calculation burden of user. However, TPA is not necessarily to be credible.

3. FULL NODE AVL TREE BASED ON HENON CHAOTIC MAP

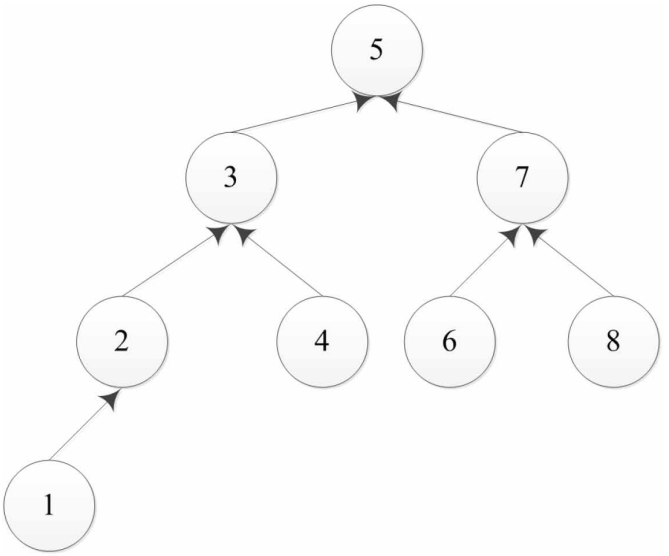
3.1. AVL Tree

AVL tree is a self-balanced binary tree (Sun, Ryozo & Sun, 2000), which has the following properties:

1. If the left subtree is not empty, the node value on the left subtree is less than or equal to the value of its root node;
2. If the right subtree is not empty, the node value on the right subtree is greater than or equal to the value of its root node;
3. The left and right subtrees are also binary sort trees.

As shown in Figure 1, the node value on the left subtree is less than root node value and the node value on the right subtree is greater than root node value. In this scheme, the node value represents

Figure 1. AVL tree



the sequence number of the data blocks. AVL tree is the self-balance tree and the node balance factor is defined as the difference between the height of the left subtree and the height of the right subtree. If the node balance factor is 1,0 or -1, this node is a balanced node (Wei, Zhang & Chun, 2010). When some updated operations on the AVL tree make it out of balance, it will rebalance by self-rotation. For its highly balanced characteristic, the maximum complexity of updating operations on AVL tree is O(log n). Compared with MHT, the structure of the AVL tree will greatly improve the search efficiency.

3.2. Henon Chaotic Map

Henon map is defined as:

$$X_{n+1} = 1 - aX_n^2 + bX_{n-1} \tag{1}$$

When $a = 1.4$ and $b = 0.3$, it is chaotic. Henon chaotic map has complex behavior and more abundant characteristics: good randomness and sensitivity to initial condition, thus will improve the security.

3.3. Construction of the Full Node AVL Tree

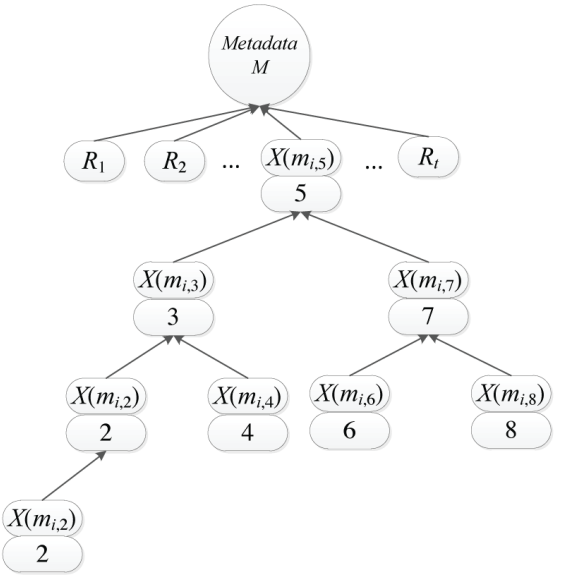
As shown in Figure 2, each replica corresponds to a full node AVL tree, and each data block in the replica corresponds to one node, which will greatly reduce the overhead for the authentication path and improve the efficiency.

In this scheme, both TPA and CSP should keep AVL tree of data, but there are some difference in the trees. Here *TTree* represents AVL tree kept by TPA, and *CTree* represents AVL tree kept by CSP.

3.3.1. TTree

In *TTree*, each node stores node value $X(m_{i,j})$ and the sequence number of the data blocks. The value of the leaf nodes and non-leaf nodes of the binary tree are calculated by (2) and (3), respectively.

Figure 2. *Full node AVL tree——* $TTree$



$$X(m_{i,j}) = \frac{\sum_{k=1}^{L} A_{i,j,k} \times \frac{1}{256}}{L} \tag{2}$$

$$\begin{cases} K(m_{i,j}) = 1 - aX_{LS}^2 + bX_{RS} \\ T(m_{i,j}) = \dfrac{\sum_{k=1}^{L} A_{i,j,k} \times \frac{1}{256}}{L} \\ X(m_{i,j}) = 1 - aK(m_{i,j})^2 + bT(m_{i,j}) \end{cases} \tag{3}$$

where X_{LS} and X_{RS} are the left and right child node of X_F, respectively. Henon map can spread and enlarge the change of node value. As we can see from (2) and (3), any change in the AVL tree will bring great change in the root node value, therefore we can judge the integrity of the data.

To check multiple copies, we aggregate the root nodes of all copies of the file into metadata M as done in (Long, Li & Peng, 2017). Metadata is calculated by

$$M = h(R_1 + R_2 + ... + R_t) \tag{4}$$

In this way, we can only check the metadata to determine the position of the data block of all copies of the file that is changed.

3.3.2. CTree

In *CTree*, each node should store $T(m_{i,j})$, $m_{i,j}$ and the sequence number of the data blocks *n* as shown in Figure 3.

4. THE PROPOSED VERIFICATION SCHEME

4.1. Initialization

The initialization consists of the following three algorithms:

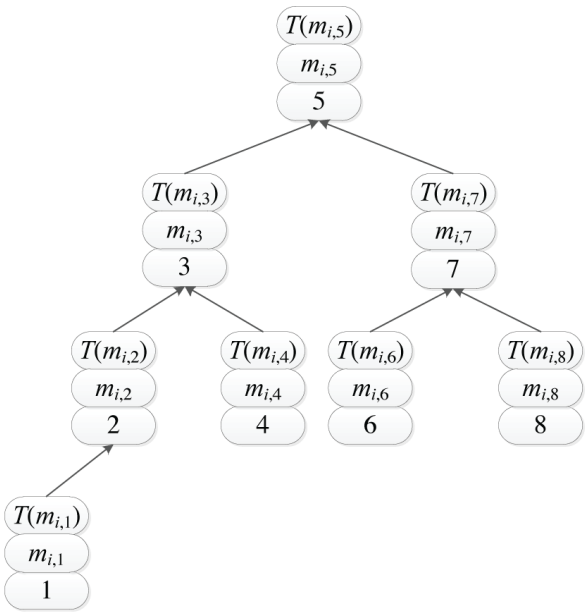
4.1.1. Key Generation $KeyGen(\cdot)$

$KeyGen(1^k) \to (sk, pk)$ is used to generate public and secret key pairs that is run by the user. Firstly, user chooses a random $x \in Z_p$ and computes $\gamma = g^x$, then apply for a public-private key pair (ssk, spk) from the key management center as the keys of encryption function $E_{key}(\cdot)$ and decryption function $D_{key}(\cdot)$, respectively. The secret key is $sk = (x, K_1, ssk)$ and the public key is $pk = (\gamma, g, spk)$.

4.1.2. Replica Generation $ReplicaGen(\cdot)$

$ReplicaGen(t, F) \to F_i$ is run by the user to produce multiple copies. Encryption function $E_{ssk}(\cdot)$ is firstly performed to F and get the encrypted file $F' = \{b_1, b_2, ..., b_n\}$. After that, computes $m_{i,j} = b_j + r_{i,j}$, $r_{i,j}$ is a random number, and get t different file copies $F_i = \{m_{i,j}\}_{1 \le i \le t, 1 \le j \le n}$.

Figure 3. *Full node AVL tree——* $CTree$



4.1.3. Tag Generation $TagGen(\cdot)$

$TagGen(sk, F_i) \to (\phi_j, M)$ is executed by user. Firstly, the user generates a set of random numbers $u_i \leftarrow G(i = 1, 2, 3, ..., t)$ and computes the tag $\sigma_{i,j} = [u_i^{m_{i,j}}]^x \in G$ for each block $m_{i,j}$, then integrates these tags into aggregated tags $\phi_j = \prod_{i=1}^{t} \sigma_{i,j} \in G$. After that, the metadata M can be acquired according to (4).

Thereafter, the user sends $\{pk, M, TTree\}$ to TPA and $\{F_i, \phi_j, CTree_i\}_{1 \le i \le t, 1 \le j \le n}$ to the master server, then deletes the copies, the trees, and tags at its local storage, and only keeps (pk, sk).

4.2. Verification

The validation includes two algorithms as follows:

4.2.1. Proof Generation $GenProof(\cdot)$

$Gen\,\mathrm{Pr}oof(F_i, \phi_j, chal) \to P$ is performed by cloud server. TPA periodically challenges the cloud server to determine if cloud server possesses all copies of the file. Firstly, TPA randomly selects c elements from [1, n] to compose an integer set $I = \{l_1, l_2, ..., l_c\}$ ($l_1 \le l_2 \le ... \le l_c$), and picks a random element $v_j \in Z_p$ for each $j \in I$, then gets the challenged information $chal = \{j, v_j\}_{l_1 \le j \le l_c}$, finally sends $\{chal\}$ to the master server.

After receiving $\{chal\}$, the master server computes $\{T(m_{i,j})\}_{1 \le i \le t, l_1 \le j \le l_c}$, and selects the random numbers $k_i \in Z_p (i = 1, 2, ..., t)$, then get $\mu_i^{'}$ and $\phi^{'}$ given by (5) and (6).

$$\mu_i^{'} = \sum_{j=l_1}^{l_c} v_j \cdot m_{i,j} + k_i \in Z_p \tag{5}$$

$$\phi^{'} = \prod_{j=l_1}^{l_c} \phi_j^{v_j} \in G \tag{6}$$

Set $u_i^{k_i} = K_i$ and get the verification proof $P = \{K_i, \mu_i^{'}, \phi^{'}, \{T(m_{i,j}), \lambda_{i,j}\}\}_{1 \le i \le t, l_1 \le j \le l_c}$, then send it to TPA. As shown in Figure 3, where the challenged block is $m_{i,2}$, and $\lambda_{i,2} = \{\tau_{i,2}, \tau_{i,4}, \tau_{i,3}, \tau_{i,7}, \tau_{i,5}\}$, where $\tau_{i,j}$ is the transformation value of the node value. For example, $\tau_{i,2} = \{T(m_{i,2}), T(m_{i,1})\}$ and $\tau_{i,7} = \{T(m_{i,7}), T(m_{i,6}), T(m_{i,8})\}$.

4.2.2. Proof Verification $Verify\,\mathrm{Pr}oof(\cdot)$

$Verify\,\mathrm{Pr}oof(P, M) \to (\text{"}False\text{"}, \text{"}True\text{"})$ is executed by TPA. After TPA receives the proof *P*, the metadata M' can be computed according to $\{X(m_{i,j}), \lambda_{i,j}\}_{1 \le i \le t, l_1 \le j \le l_c}$, and then checks if $M' = M$ holds. If it is not true, output "*False*" to the user, otherwise TPA continues to calculate:

$$e\left(\prod_{i=1}^{t} K_i \cdot \phi', g\right) \stackrel{?}{=} e\left(\prod_{i=1}^{t} u_i^{\mu_i'}, \gamma\right) \tag{7}$$

If Equation (7) holds, TPA returns "*True*" to the user, otherwise returns "*False*". The correctness of the Equation (7) is verified as follows:

$$e\left(\prod_{i=1}^{t} K_i \cdot \phi', g\right) = e\left(\prod_{j=l_1}^{l_c}\left(\prod_{i=1}^{t} u_i^{k_i} \cdot \phi_j^{v_j}, g\right)\right) = e\left(\prod_{j=l_1}^{l_c}\left(\prod_{i=t}^{t} u_i^{k_i} \cdot (u_i)^{m_{i,j}}\right)^{v_j}, \gamma\right)$$

$$= \left(e\left(\prod_{i=1}^{t} u_i^{\sum_{j=l_1}^{l_c} v_j \cdot m_{i,j} + k_i}\right), \gamma\right) = e\left(\prod_{i=1}^{t} u_i^{\mu_i'}, \gamma\right)$$

5. PERFORMANCE ANALYSIS

Theorem 1. The cloud server should meet the following two points if it can pass the integrity verification by TPA.

1. The value and the position of the node in binary trees are correct.
2. The cloud servers have the correct blocks.

Proof:

1. During the data integrity verification, TPA first calculates the new metadata M' and check if $M' \neq M$ holds, where M is derived from the root node $\{R_i\}_{1 \le i \le t}$ of all copies, and $\{R_i\}_{1 \le i \le t}$ are calculated by $K(m_{i,j}) = 1 - aX_{LS}^2 + bX_{RS}$ and $X(m_{i,j}) = 1 - aK(m_{i,j})^2 + bT(m_{i,j})$. According to the sensitivity of the chaotic system, any little change in AVL tree will lead to great change in the root node R_i and the metadata M. Therefore, that $M' = M$ can guarantee the value and position of the node in AVL tree are correct.
2. This proof is based on Discrete Logarithm (DL) problem. DL problem: given $g, h \in G$ for some group G, it is hard to find x such that $h = g^x$.

Cloud servers return the integrity evidence $P = \{K_1, \mu_i', \phi'\}$ to the TPA when TPA challenges the cloud servers. However, malicious cloud servers may remove or tamper a part of blocks and forge the mendacious evidence $P = \{K_1, \mu_i^{*'}, \phi'\}$. Supposing $\mu_i^* \neq \mu_i$, both P and P' can pass the integrity verification by TPA, then check

$$e\left(\prod_{i=1}^{t} K_i \cdot \phi', g\right) \stackrel{?}{=} e\left(\prod_{i=1}^{t} u_i^{\mu_i'}, \gamma\right) \tag{8}$$

$$e\left(\prod_{i=1}^{t} K_i \cdot \phi', g\right) \stackrel{?}{=} e\left(\prod_{i=1}^{t} u_i^{\mu_i^*}, \gamma\right) \tag{9}$$

Let $\Delta\mu_i = \mu_i - \mu_i^*$ and $u_i = g^{\delta_i} h^{\beta_i}$, where $\delta_i, \beta_i \in Z_p$. Dividing (8) by (9), then get $e\left(\prod_{i=1}^{t} u_i^{\mu_i - \mu_i^*}, \gamma\right) = 1$, $\prod_{i=1}^{t}\left(g^{\delta_i} \cdot h^{\beta_i}\right)^{\Delta\mu_i} = 1$, $g^{\prod_{i=1}^{t}\delta_i \cdot \Delta\mu_i} \cdot h^{\prod_{i=1}^{t}\beta_i \cdot \Delta\mu_i} = 1$ and $h = g^{(-\prod_{i=1}^{t}\delta_i \cdot \Delta\mu_i)/(\prod_{i=1}^{t}\beta_i \cdot \Delta\mu_i)}$. The DL problem is resolved unless the denominator of $x = \left(\prod_{i=1}^{t} \delta_i \cdot \Delta\mu_i\right) \Big/ \left(\prod_{i=1}^{t} \beta_i \cdot \Delta\mu_i\right)$ is zero. However, $\beta_i \in Z_p$, the probability of $\beta_i = 0$ is only $1/p$, which is negligible. It means there is no x can be found such that $h = g^x$ when $\Delta\mu_i = 0$ holds, that is $\mu_i^* = \mu_i$. Therefore, the cloud server can pass the verification by TPA only when it holds the correct data blocks.

6. EFFICIENCY ANALYSIS

In this section, we compare the proposed scheme to some integrity verification schemes. As shown in Table 1, the proposed scheme can achieve data dynmic operation and support multiple copies authentication, and keep the binary tree balanced therefore get low complexity of updating operations.

Table 2 shows the storage cost of three integrity verification schemes. Here assume that the number of the copies is 1, and n represents the number of blocks per copy. The size of file copies

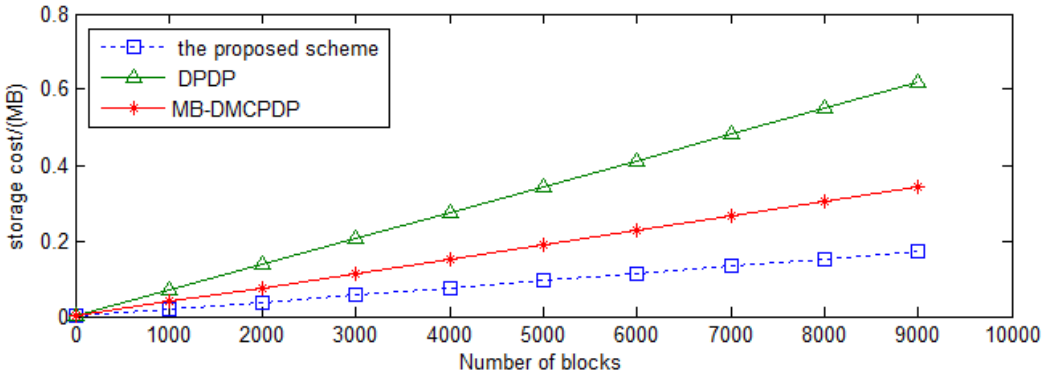
Table 1. Complexity of updating operations

		Zhang, Ni, Tao, Wang & Yu, 2015	Barsoum & Hasan, 2010	Li & Wang, 2016	Zhang & Blanton, 2013	The proposed scheme
Multiple copies		Yes	Yes	No	No	Yes
Balance		No	No	No	Yes	Yes
Complexity of updating operations	Insertion	O(n)	O(n)	O(n)	O(log n+1)	O(log n)
	Deletion	O(n)	O(n)	O(n)	O(log n+1)	O(log n)
	Search	O(n)	O(n)	O(n)	O(log n+1)	O(log n)

Table 2. Storage cost

	DPDP	MB-DMCPDP	The proposed scheme
Copies number	$\|F\|$	$\|F\|$	$\|F\|$
Storage in CSP	$160(2n-1)+257n$	$257n$	$64n+16n$
Storage in TPA	160	$64n$	$64n+16n+160$

Figure 4. Comparison of storage cost



is $|F|$. Assume the hash function h_{SHA} is $SHA-1$, where the length of the hash value is 160 bits. In the proposed scheme, the node in *CTree* stores $X(m_{i,j})$ and n, whose size are 64 bits and 16 bits respectively. While the node in *TTree* stores M with size 160 bits besides $X(m_{i,j})$ and n. In DPDP (Li & Wang, 2016), CSP needs to store the binary tree, where every node in the tree store the hash value and data tag, and the storage in CSP is $160(2n-1)+257n$, while TPA needs only store M and the storage in TPA is 160 bits. In MB-DMCPDP (Barsoum & Hasan, 2011), for the updating operations are achieved by map-version, the storage in CSP is $257n$ and the storage in TPA is $64n$. Overall, the proposed scheme outperforms the other two schemes in terms of storage cost as shown in Figure 4.

7. CONCLUSION

A data integrity verification scheme based on AVL tree is proposed in this paper. It supports multiple copies with dynamic operations, and can improve efficiency. Theoretical analysis confirms the security, and the Experimental results show that the computational overhead and storage cost of the proposed scheme outperform those of the methods based on Merkle-Hash-Tree. It has great potential in the application of data protection in cloud storage.

REFERENCES

Ateniese, G., Burns, R., Curtmola, R., Herring, J., & Kissner, L. (2007). Provable data possession at untrusted stores. In *ACM Conference on Computer and Communications Security* (pp. 598-609). New York, NY: ACM Press.

Ateniese, G., Kamara, S., & Katz, J. (2009). Proofs of Storage from Homomorphic Identification Protocols. In *International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology* (pp. 319-333). Berlin Heidelberg, Germany: Springer-Verlag. doi:10.1007/978-3-642-10366-7_19

Ateniese, G., Pietro, R. D., Mancini, L. V., & Tsudik, G. (2008). Scalable and efficient provable data possession. In *Proceedings of the 4th international conference on Security and privacy in communication networks*. New York, NY: ACM Press.

Barsoum, A., & Hasan, A. (2013). Enabling Dynamic Data and Indirect Mutual Trust for Cloud Computing Storage Systems. *IEEE Transactions on Parallel and Distributed Systems*, *24*(12), 2375–2385. doi:10.1109/TPDS.2012.337

Barsoum A. F., Hasan M. A. (2010). Provable Possession and Replication of Data over Cloud Servers. Centre For Applied Cryptographic Research (CACR), University of Waterloo.

Barsoum, A. F., & Hasan, M. A. (2011). On Verifying Dynamic Multiple Data Copies over Cloud Servers. Retrieved from http://www.cacr.math.uwaterloo.ca/techreports/2010/ cacr2010-32.pdf

Barsoum, A. F., & Hasan, M. A. (2015). Provable Multicopy Dynamic Data Possession in Cloud Computing Systems. *IEEE Transactions on Information Forensics and Security*, *10*(3), 485–497. doi:10.1109/TIFS.2014.2384391

Curtmola, R., Khan, O., Burns, R., & Ateniese, G. (2008). MR-PDP: Multiple-Replica Provable Data Possession. In *The International Conference on Distributed Computing Systems* (pp. 411-420). Piscataway, NJ: IEEE Press.

Deswarte, Y., Quisquater, J. J., & Saïdane, A. (2004). Remote Integrity Checking. (2014). *Integrity and Internal Control in Information Systems*, *VI*, 1–11.

Erway, C., Küpçü, A., Papamanthou, C., & Tamassia, R. (2009) Dynamic provable data possession. *ACM Conference on Computer and Communications Security* pp. 213-222). New York, NY: ACM Press.

Feng, D. G., Zhang, M., Zhang, Y., & Xu, Z. (2011). Study on Cloud Computing Security. *Journal of Software*, *22*(01), 71–83. doi:10.3724/SP.J.1001.2011.03958

Fu, Y., Zhang, M., Chen, K., & Feng, D. (2014). Proofs of data possession of multiple copies [In Chinese]. *Journal of Computer Research & Development*, *51*(7), 1410–1416.

Gritti, C., Susilo, W., & Plantard, T. (2015). *Efficient Dynamic Provable Data Possession with Public Verifiability and Data Privacy. Information Security and Privacy* (pp. 395–412). Berlin Heidelberg, Germany: Springer International Publishing.

Juels, A., & Kaliski, B. S. (2007). Pors: proofs of retrievability for large files. In *ACM Conference on Computer and Communications Security* (pp. 584-597). New York, NY: ACM Press. doi:10.1145/1315245.1315317

Li, Y., Qiu, L., Qiu, L., Qiu, M., & Zhao, H. (2016). Intelligent cryptography approach for secure distributed big data storage in cloud computing. *Information Sciences*, *387*, 103–115. doi:10.1016/j.ins.2016.09.005

Long, M., Li, Y., & Peng, F. (2017). Integrity Verification for Multiple Data Copies in Cloud Storage Based on Spatiotemporal Chaos. *International Journal of Bifurcation and Chaos in Applied Sciences and Engineering*, *27*(4). doi:10.1142/S0218127417500547

Sun, N., Ryozo, N., & Sun, W. (2000). The average behavior of the AVL tree insertion algorithm. *Journal of the CNN [Natural Sciences Edition]*, *01*, 31–39. (in Chinese)

Wei-Wei, D. U., Zhang, Y. Y., & Chun-Liu, Q. U. (2010). AVL Tree Design and Implementation Based on Balancing Factor. *Computer Technology & Development*, *20*(3), 24–27. (in Chinese)

Yan, X. T., & Li, Y. (2013). Data Integrity Checking Approach Based on Message Authentication Function for Cloud. *Dianzi Yu Xinxi Xuebao*, *35*(2), 310–313. doi:10.3724/SP.J.1146.2012.00629

Zhang, Y., Ni, J., Tao, X., Wang, Y., & Yu, Y. (2015). Provable multiple replication data possession with full dynamics for secure cloud storage. *Concurrency and Computation*, *28*(4), 1161–1173. doi:10.1002/cpe.3573

Zhou, E., & Li, Z., GUO, H., Jia Y. L. (2014). An improved data integrity verification scheme in cloud storage system. *Tien Tzu Hsueh Pao*, *42*(1), 150–154. (in Chinese)