An Efficient Methodologies for Data Integrity in Cloud Storage

L Jambulingam, T V Ananthan, Sumathy Eswaran

Abstract*: Majority of the organization uses cloud for storage purpose in order to reduce the cost as well as maintenance. Due to increasing threat from internal and external sources, there would be possibility of corruption in the cloud storage files. Thus the storage must to be monitored periodically for integrity checking. Since most of the Data Owners have limited resources thus the responsibility of integrity checking goes to the Third Party Auditors (TPA). In this paper, we have proposed 2 methodologies of Integrity Checking in Cloud Storage (1) Enhanced Dynamic Hash Tree – n Versions (EDHT-n), which has best performance in term of time and space complexity compared to the existing methods.(2) Hybrid Enhanced Dynamic Hash Tree (HEDHT), which is best suited for very huge number of files in a directory.

Index Terms*: Third Party Auditor (TPA), Hybrid Enhanced Dynamic Hash Tree, Enhanced Dynamic Hash Tree, Meta Data, Meta Data Server (MDS). Cloud Service Provider (CSP), Business Continuity Planning (BCP), Service Discovery.

I. INTRODUCTION

Storage data is the asset to the client, to reduce cost, original file is being stored in cloud and deleted in the end-user. Thus integrity should be foremost requirement, any corruption had happened in CSP, it should be detected & corrected either during the file request given by the users or during auditing stage and made immediate recovery in order to maintain the integrity in the form of consistency across all of its data backup copies by executing the recovery routines [1], [2], [9], [19]. Most importantly, the user may not aware about these processes are running in background, and thus BCP would be ensured always. Subsequently the trustworthiness towards the CSP would be improved.

II. METHODOLOGY

A. EDHT-n

Enhanced Dynamic Hash Tree – n Versions which has best performance in term of time and space complexity compared to the existing methods as shown in Figure 1 [10], [18], [23]; each non-leaves in this hash tree are computed using its "n" children nodes and error detection time is very much reduced which is proportional to its versioning to find the faulty files. Less number of meta-data will be used for integrity checking as well as less memory would require as we go on to higher version of EDHT. It also has provision to split-up huge sized directories into multiple small sized directories; which ease in both auditing as well as caching.

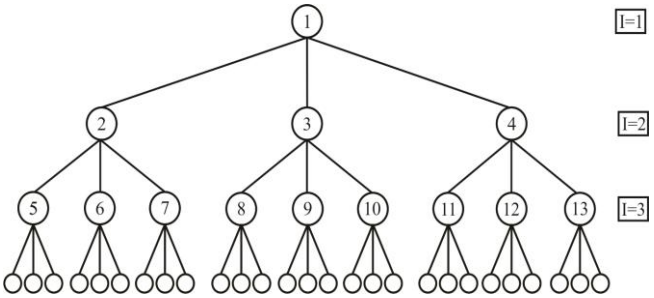


Fig.1 Enhanced Dynamic Hash Tree – Version 3

B. HEDHT

Hybrid Enhanced Dynamic Hash Tree - Best suited for very huge number of files in a directory. Hybrid EDHT, where root is arrived using 2 child hash node; these 2 children hash nodes are computed individually using 3 child hash node and so on as shown in Figure 2, hence at each level down, the nodes are computed by its hash node incrementally by one. Leaf nodes are representing the actual file hashes using SHA-256/SHA-512. More the number of consecutive nodes merged, we get height of Hybrid Enhanced Dynamic Hash Tree minimal, thus it needs minimum hash computation to calculate HEDHT compared to EDHT-n as well as it requires less recovery time and less memory usage compared to EDHT-n and the current methodology like Merkle Hash Tree (MHT) [11], [13], [14], [17] where most of the auditing would currently has, and subsequently it generates the challenge as well as verification block in a very less computation time and space which in turn improves the auditing speed. Both these proposed methodologies would also supports batch auditing, dynamic update [20], [27], privacy preserving auditing [5], [12], [21], [22], [24], [26], [28], [29], all file formats, allowed multiple proxies as well as auditors are capable of auditing number of backup copies based on the agreed SLA by CSP in an efficient way. Sequential auditing [7], [16], [25] and recovery process to maintain the integrity of data will be delayed, hence it is replaced with batch auditing using multi-threading, caching and load balancing. Meta-data is the key information for performing the integrity checking; hence it should be generated using multiple threading.

Revised Manuscript Received on July 06, 2019.

L Jambulingam, Research Scholar, Department of Computer Science and Engineering, Dr.M.G.R. Educational and Research Institute University, Chennai, India.

T V Ananthan, Professor, Department of Computer Science and Engineering, Dr.M.G.R. Educational and Research Institute University, Chennai, India.

Sumathy Eswaran, Professor, Department of Computer Science and Engineering, Dr.M.G.R. Educational and Research Institute University, Chennai, India.

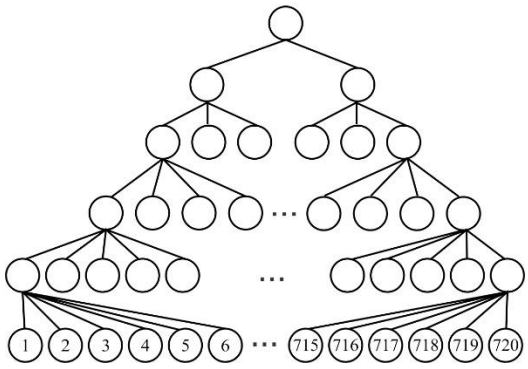


Fig.2 Hybrid Enhanced Dynamic Hash Tree

III. META DATA GENERATION

When client want to store files in cloud storage, it should to be inputted in Meta Data Server, where SHA-256/SHA-512 processing would happen to generate Meta-Data (digest) which is the minimal representation of the file. Meta Data Server would spawns multiple threads to achieve parallelism in generation of meta-data, subsequently distribute the meta-data to TPA as well as CSP along with actual file content and delete the actual file in its end-user. Besides thread spawning, load balancing would happen by means of Service Discovery which further enhances the speed by distributing the load to multiple clusters in cloud, so the processing time would speed-up to that many times of cluster available in MDS and CSP end as shown in Figure 3.

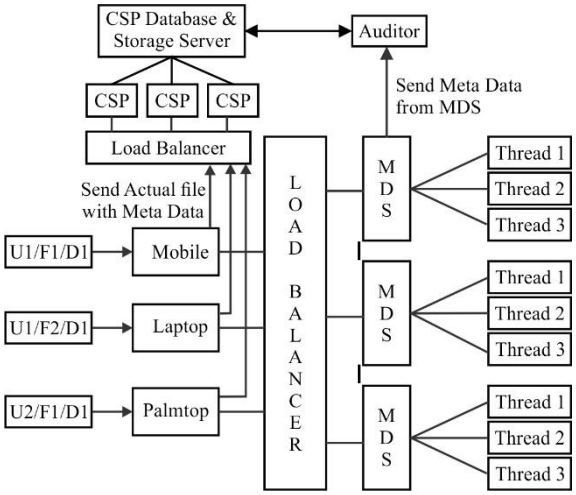


Fig.3 Meta-Data Generation; U1/U2 – User 1/User 2; F1/F2 – File 1/File 2; D1 – Directory 1

IV. FILE RETRIEVAL IN CLOUD

Suppose user want to retrieve a specific file in a particular directory in the cloud then they would send the request to auditor, which in turn get the challenges appropriate to the requested file and sent to CSP, based on these challenges the proof will be returned in the form of root by the CSP to the auditor, hence the auditor will compare the root comes from CSP with the existing root of the EDHT-n/HEDHT, if both are same then the actual file retrieval from the CSP would happen else the actual recovery routine would be invoked as shown in Figure 4. Hence proposed integrity methodology EDHT-n version should be best way for integrity checking compared to existing Merkle Hash Tree [15], [3], [8], [4], [6] since EDHT-n would generate tree of minimal height thus the verification of file intact would be faster, also detection time to find the faulty file is proportional to its version number, soon after the detection, recovery time is minimal. Memory also required minimal for auditing and recovery process compared to all other conventional integrity method including Merkle Hash Tree.

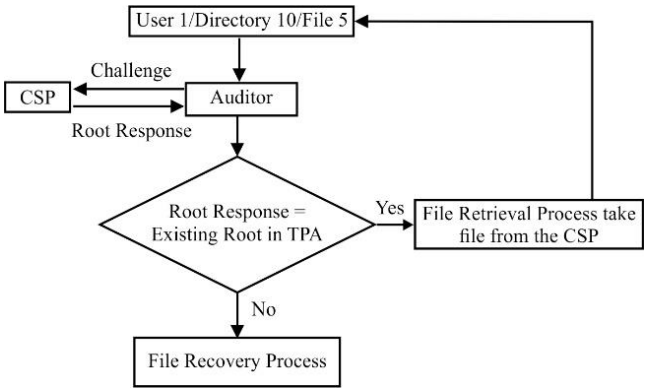


Fig.4 Proof of Ownership

V. THREE STEP RECOVERY PROCESS

A. Identify the Faulty Tree and files in the EDHT-n / Hybrid EDHT Suppose user want to retrieve a specific file but meta-data in the form of EDHT-n in CSP got corrupted by removal of file by CSP explicitly or file lost due to other varied reasons. Auditor is sole responsible for the recovery process to maintain consistency of data. Using meta-data, user data intactness can be checked by auditor in CSP.

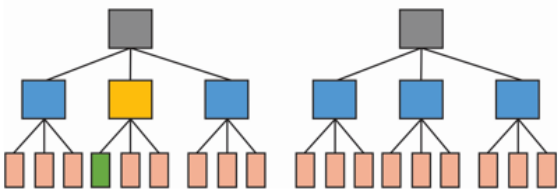


Fig.5 LHS is Corrupted EDHT-3 in CSP versus RHS is Uncorrupted EDHT-3 in TPA.

Firstly, we should detect the faulty tree by comparing the concatenated EDHT-n leaves at CSP side with the concatenated EDHT-n leaves in Auditor side, if both hashed values are same then it is said to be non-faulty tree else it would identified as faulty. If CSP data is not intact then recovery process to be taken care by auditors as follows:

Step 1: Build EDHT-n using the existing files' meta-data in the CSP side.

Step 2: Calculate level-2 nodes using level-3 nodes of CSP, then same level-2 nodes are copied from auditors and compared, assume node2 of "level-2" is not matching then we can say, files under Node 2 got corrupted.

Step 3: Similarly calculate level-3 nodes using level-4 nodes of CSP and compared with auditors copy, assume node7 of level-3 is not matching then we can say files under Node7 got corrupted. Same process is repeated till leaf node to find out the exact file got corrupted. In this

optimize way, all the faulty files would be detected.
B. Recovery routine would recover the faulty files from the backup copy
C. Reconstruct the EDHT-n/Hybrid EDHT using recovered files from faulty
Step 1: Recovered files will be sorted by file name numeric identifier in ascending and hashed using SHA-256 by its contents and pushed inside the STACK-ONE.
Step 2: POP from the STACK-ONE one by one and identify its partner hash either within the STACK-ONE or from the tree representation of EDHT-n / HEDHT and concatenate all those and assign to the idle threads and this process continues till the STACK-ONE becomes emptied
Step 3: Output of these threads are hashed, which are collected individually and get sorted in the order it were sent to the thread and again all these hashes are pushed inside the STACK-ONE and the step 2 will be continued till the root hash is constructed.

Number of hash computation for MHT for recoverability

$$(h-1) * 2 + [1 + \sum_{s=1}^{s=h-1} 2^s \,] \quad (1)$$

Number of hash computation for Enhanced Dynamic Hash Tree –Version 3 for recoverability

$$(h-1) * 3 + [1 + \sum_{s=1}^{s=h-1} 3^s \,] \quad (2)$$

Number of hash computation for Hybrid Enhanced Dynamic Hash Tree for recoverability

$$[\sum_{s=2}^{s=h} s + \sum_{s=1}^{s=h} \prod_{i=1 \text{ to } s} i \,] \quad (3)$$

Number of hash transfer and hash comparison for MHT for recoverability

$$(h-1) * 2 \quad (4)$$

Number of hash concatenation for MHT for recoverability

$$(h-2) * 2 \quad (5)$$

Number of hash transfer and hash comparison for Enhanced Dynamic Hash Tree –Version 3 for recoverability

$$(h-1) * 3 \quad (6)$$

Number of hash concatenation for Enhanced Dynamic Hash Tree – Version 3 for recoverability

$$(h-2) * 3 \quad (7)$$

Number of hash transfer and hash comparison for Hybrid Enhanced Dynamic Hash Tree for recoverability

$$\sum_{s=2}^{s=h-1} s \quad (8)$$

Number of hash concatenation for Hybrid Enhanced Dynamic Hash Tree for recoverability

$$\sum_{s=2}^{s=h-1} s \quad (9)$$

VI. BATCH AUDIT USING EDHT-N AND HEDHT

Batch Audit using EDHT-n or HEDHT would be many times faster than the earlier integrity checking methods like MHT and so on. As we increase the version of EDHT, we could find an huge difference in speed between the EDHT-n and EDHT-(n-1) as well as between HEDHT and EDHT-n. HEDHT would be more benefited if more number of files in a given directory in cloud storage, similar way, batch recovery process can be possible.

3 Step Batch Audit Process

1. TPA picks up huge number of directories details to be audited from EDHT-n/HEDHT stored database based on the audit frequency policy.

2. Partition process would put the picked directories details into number of buckets, each bucket contains minimum of 100 to 200 directories to perform batch audit.

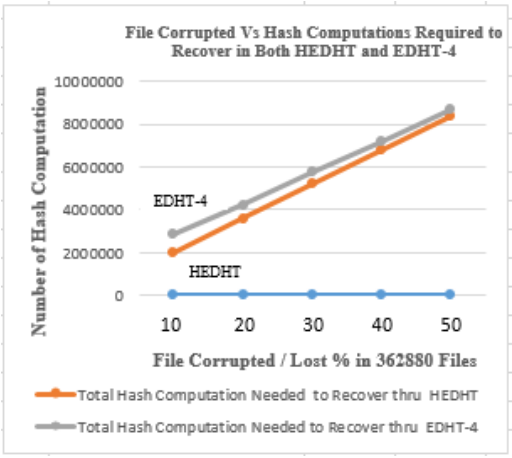
3. Each bucket filled with multiple directory details (at TPA side) would be processed by available individual thread to perform the following process:

3.1 Reader process would read the actual directories details from the bucket and update the audit status for each directories as work in progress (WIP), and would process by invoking the application load balancer by passing the unprocessed bucket, based on service discovery it would identify the clusters in cloud to do the following:
3.1.1 Partition process would create multiple chunks as that of the number of directories available in the given bucket, assume each bucket have 200 directories, then we have 200 chunks.
3.1.2 Assume we have 25 threads as per the configuration, hence each chunk would go to the separate thread at CSP side to perform the actual audit:
3.1.2.1 Reader process would read the directory details in the form of challenge message
3.1.2.2 Processor would process the challenge message to arrive the root based on the Hash Tree methodology defined in the directory details, so it can be either EDHT-n/HEDHT.
3.1.2.3 Writer would write the root of the HEDHT/EDHT-n as a verification message
3.2 Processor would process by getting the verification message arrived at the CSP side (i.e., root of the HEDHT/EDHT-n) and compare this with the actual root stored in the TPA side; if both are same then we can say the integrity is maintained hence the integrityFlag would set as “true” else “false”.
3.3 Writer process would update the audit status for each directory as COMPLETED and based on the above process step the integrity Flag can also be set in the HEDHT/EDHT-n database; In case of integrity Flag as “false” then the separate procedure would be invoked to perform the recovery process by TPA and thus appropriate correction of the file happen in the respective CSP.

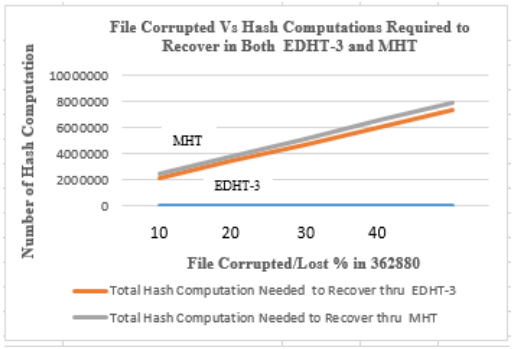
VII. RESULTS AND DISCUSSION

In Graph 1, it is very clear the hash computation required is more in EDHT-4 compared to HEDHT during recovery process in various file corrupted/lost percentage. Moreover there is a steep improvement in recovery speed in HEDHT, if huge number of files in a directory to be audited. In Graph 2, it is very clear the hash computation required is more in MHT compared to EDHT-3 during recovery

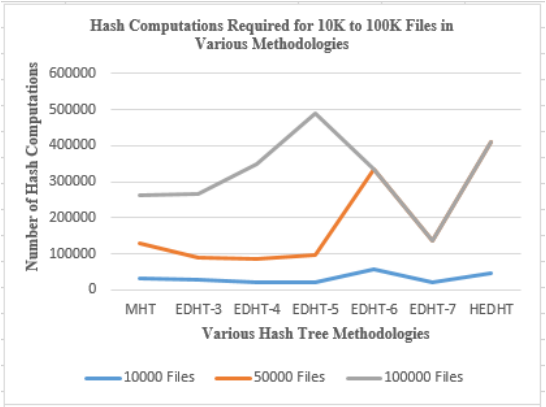
process in various file corrupted/lost percentage. In graph 3, all hash tree methodologies like MHT, EDHT-n, HEDHT would require number of hash computation which is proportional to number of files Also, each methodology for same number of files would decreases the hash computation for constructing the tree in order of MHT, EDHT-n, HEDHT, hence from both graph 3 and graph 4, it is very clear that HEDHT would be more efficient compared to other Tree construction as we increase the number of files. Thus the reason to say it is ideal for large number of files. From Graph 5, it is obvious, as we increases the Enhanced Dynamic Hash Tree versions, the cache memory required would be precipitously reduced.



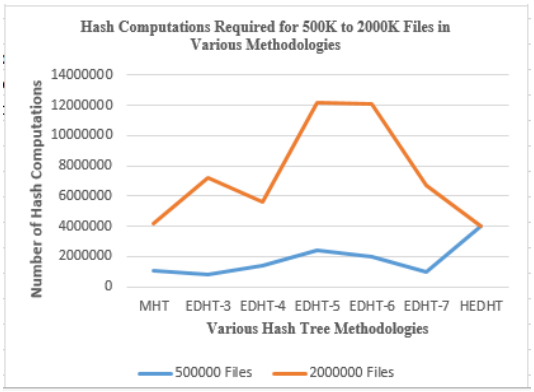
Graph 1: File Corrupted versus Hash Computations Required to Recover in Both HEDHT and EDHT-4



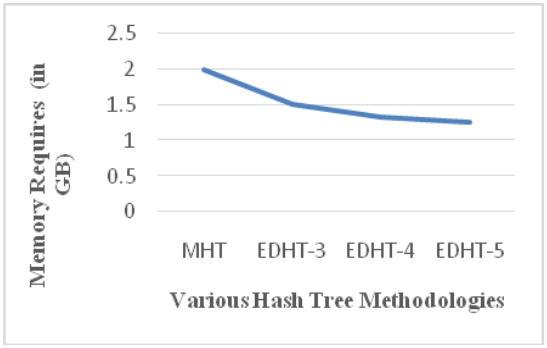
Graph 2: File Corrupted versus Hash Computations Required to Recover in Both HEDHT and EDHT-3



Graph 3: Hash Computations Required for 10K Files to 100K Files in MHT, EDHT-3, EDHT-4, EDHT-5, EDHT-6, EDHT-7, HEDHT



Graph 4: Hash Computations Required for 500K Files to 2000K Files in MHT, EDHT-3, EDHT-4, EDHT-5, EDHT-6, EDHT-7, HEDHT



Graph 5: RAM requires in Giga Bytes in MHT, EDHT-3, EDHT-4 and EDHT-5

VIII. CONCLUSION

Cloud Storage is an Infrastructure as a Service (IaaS) which is a cost effective to an organizations with an integrity challenges, thus storage intactness to be checked periodically. In this paper we have done the research work to find the efficient Audit process for the Cloud Storage as well as Speedy Recovery process for the lost or corrupted file in that storage maintained in the CSP in order to maintain integrity thru the proposed EDHT-n and HEDHT proposed methods. Batch processing can be implemented to speed-up both the processes.

REFERENCES

1. 1. A. Juels and J. B. S. Kaliski, “PORs: Proofs of retrievability for large fildoes,” in Proc. 14th ACM Conf. Comput. Commun. Security, Alexandria, USA, 2007, pp. 584–597.
2. 2. G. Ateniese, R. Burns, R. Curtmola, J. Herring, L.Kissner, Z. Peterson, and D. Song, “Provable data possession at untrusted storages,” in Proc. 14th ACM Conf. Comput. Commun. Secur., 2007, pp. 598–609.
3. G. Ateniese, R. Burns, R. Curtmola, J. Herring, O. Khan, L. Kissner, Z. Peterson, and D. Song, “Remote data checking using provable data possession,” ACM Trans. Inform. Syst. Secur., vol. 14, no. 1, pp. 1–34, 2011.
4. G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, “Scalable and efficient provable data possession,” in Proc. 4th Int. Conf. Secur. Privacy Commun. Netow., 2008, pp. 1–10.
5. C. Erway, A. Kupcu, C. Papamanthou, and R. Tamassia, “Privacy-Preserving Public Auditing for Data,” in Proc. 16th ACM Conf. Comput. Commun. Secur., 2009, pp. 213–222.
6. F. Sebe, J. Domingo-Ferrer, A. Martinez-Balleste, Y. Deswarte, and J.-J. Quisquater,

"Efficient remote data possession checking in critical information infrastructures," IEEE Trans. Knowl. Data Eng., vol. 20, no. 8, pp. 1034–1038, Aug. 2008.

7. Boyang Wang, Student Member, IEEE, Baochun Li, Senior Member, IEEE, and Hui Li, Member, IEEE, "Panda: Public Auditing for Shared Data with Efficient User Revocation in the Cloud", IEEE Trans. on services computing, vol. 8, no. 1, Jan./Feb. 2015.
8. Y. Zhu, H. Hu, G.-J. Ahn, and M. Yu, "Cooperative provable data possession for integrity verification in multicloud storage," IEEE Trans. Parallel Distrib. Syst., vol. 23, no. 12, pp. 2231–2244, Dec. 2012.
9. H. Shacham and B. Waters, "Compact proofs of retrievability," in Proc. 14th Int. Conf. Theory Appl. Cryptol. Inform. Secur.: Adv. Cryptol., 2008, pp. 90–107.
10. Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in Proc. Comput. Secur., 2009, pp. 355–370.
11. J. Xu and E.-C. Chang, "Towards efficient proofs of retrievability," in Proc. 7th ACM Symp. Inform., Comput. Commun. Secur., 2012, pp. 79–80.
12. E. Stefanov, M. van Dijk, A. Juels, and A. Oprea, "Iris: A scalable cloud file system with efficient integrity checks," in Proc. 28th Annu. Comput. Secur. Appl. Conf., 2012, pp. 229–238.
13. M. Azraoui, K. Elkhiyaoui, R. Molva, and M. €Onen, "Stealthguard: Proofs of retrievability with hidden watchdogs," in Proc. Comput. Secur., 2014, pp. 239–256.
14. J. Li, X. Tan, X. Chen, and D. Wong, "An efficient proof of retrievability with public auditing in cloud computing," in Proc. 5th Int. Conf. Intell. Netw. Collaborative Syst., 2013, pp. 93–98.
15. H. Li, B. Wang, and B. Li, "Oruta: Privacy-preserving public auditing for shared data in the cloud," IEEE Trans. Cloud Comput., vol. 2, no. 1, pp. 43–56, Jan.–Mar. 2014.
16. Shiuan-Tzuo Shen,StudentMember,IEEE, Hsiao-Ying Lin,and Wen-Guey Tzeng,Member,IEEE, "An Effective Integrity Check Scheme for Secure Erasure Code-Based Storage Systems", IEEE Trans. on reliability, vol. 64, no. 3, Sep. 2015.
17. Chang Liu, Rajiv Ranjan, Chi Yang, Xuyun Zhang, Lizhe Wang, Senior Member, IEEE, and Jinjun Chen, Senior Member, IEEE, "MuR-DPA: Top-Down Levelled Multi-Replica Merkle Hash Tree Based Secure Public Auditing for Dynamic Big Data Storage on Cloud", IEEE Trans. Computers, vol. 64, no. 9, Sep. 2015.
18. Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 5, pp. 847–859, 2011.
19. Jin Li, Xiao Tan, Xiaofeng Chen, Duncan S. Wong, and Fatos Xhafa, "OPoR: Enabling Proof of Retrievability in Cloud Computing with Resource-Constrained Devices", IEEE Trans. on cloud computing, vol. 3, no. 2, April/June 2015.
20. K. Yang and X. Jia, ''An efficient and secure dynamic auditing protocol for data storage in cloud computing,'' IEEE Trans. Parallel Distrib. Syst., vol. 24, no. 9, pp. 1717–1726, Sep. 2013.
21. C. Wang, S. S. M. Chow, Q. Wang, K. Ren, and W. Lou, ''Privacy preserving public auditing for secure cloud storage,''IEEETrans.Comput., vol. 62, no. 2, pp. 362–375, Feb. 2013.
22. C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing," Proc. IEEE INFOCOM, pp. 525-533, 2010.
23. Y. Zhu, H. Wang, Z. Hu, G.-J. Ahn, H. Hu, and S.S Yau, "Dynamic Audit Services for Integrity Verification of Outsourced Storages in Clouds," Proc. ACM Symp. Applied Computing (SAC'11), pp. 1550-1557, 2011.
24. C. Wang, S.S. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-Preserving Public Auditing for Secure Cloud Storage", IEEE Trans. Computers, vol. 62, no. 2, pp. 362-375, Feb. 2013.
25. Jingwei Li, Jin Li, Dongqing Xie, and Zhang Cai, "Secure Auditing and Deduplicating Data in Cloud", IEEE Trans. Computers, vol. 65, no. 8, Aug. 2016.
26. Jiangtao Li, Lei Zhang, Member, IEEE, Joseph K. Liu, Haifeng Qian, and Zheming Dong, "Privacy-Preserving Public Auditing Protocol for Low-Performance End Devices in Cloud", IEEE Trans. on information forensics and security, vol. 11, no. 11, Nov. 2016.
27. Chang Liu, Jinjun Chen, Senior Member, IEEE, Laurence T. Yang, Member, IEEE, Xuyun Zhang, Chi Yang, Rajiv Ranjan, and Ramamohanarao Kotagiri, "Authorized Public Auditing of Dynamic Big Data Storage on Cloud with Efficient Verifiable Fine-Grained Updates", IEEE Trans. on parallel and distributed systems, vol. 25, no. 9, Sep. 2014.
28. Boyang Wang, Student Member, IEEE, Baochun Li, Senior Member, IEEE, and Hui Li, Member, IEEE, "Oruta: Privacy-Preserving Public Auditing for Shared Data in the Cloud", IEEE Trans. on cloud computing, vol. 2, no. 1, Jan-Mar. 2014.
29. Yong Yu, Man Ho Au, Member, IEEE, Giuseppe Ateniese, Xinyi Huang, Willy Susilo, Yuanshun Dai, and Geyong Min, "Identity-Based Remote Data Integrity Checking With Perfect Data Privacy Preserving for Cloud Storage", IEEE Trans. on information forensics and security, vol. 12, no. 4, Apr. 2017.