IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p-ISSN: 2278-8727
PP 56-63
www.iosrjournals.org

AFS: Privacy-Preserving Public Auditing With Data Freshness in the Cloud

P. Maheswari, B. Sindhumathi
M.E CSE Syed Ammal Engineering College Ramanathapuram , Tamil Nadu.
M.E CSE Syed Ammal Engineering College Ramanathapuram , Tamil Nadu.

***Abstract*:-** *In Cloud Storage, users can remotely store their data and enjoy the on-demand high quality applications and services. The integrity of cloud data is subject to skepticism due to the existence of hardware/software failures and human errors. Several mechanisms have been designed to allow both data owners and public verifiers to efficiently audit cloud data integrity without retrieving the entire data from the cloud server. However, public auditing on the integrity of shared data with these existing mechanisms will supports public auditing on shared data stored in the cloud that exploit ring signature to compute verification metadata needed to audit the correctness of shared data.so that a third party auditor (TPA) is able to verify the integrity of shared data for users without retrieving the entire data. Meanwhile, the identity of the signer on each block inshared data is kept private from the TPAalso able to perform multiple auditing tasks simultaneously instead of verifying them one byone.In this paper, we proved the data freshness(proved the cloud possesses the latest version of shared data)while still preserving identity privacy.Our experimental result ensures that retrieved data always reflects the most recent updates and prevents rollback attacks.*

Index terms - *AFS (Authenticated File System); data freshness; public auditing; shared data*

I. INTRODUCTION

With cloud computing and storage, users are able to access and to share resources offered by cloud service providers at a lower marginal cost. It is routine for users to leverage cloud storage services to share data with others in a group, as data sharing becomes standard feature in most cloud storage offerings, including Dropbox, iCloud and Google Drive [2]. The integrity of data in cloud storage, however, is subject to skepticism and scrutiny, as data stored in the cloud can easily be lost or corrupted due to the inevitable hardware/software failures and human errors [3], [4]. The traditional approach for checking data correctness is to retrieve the entire data from the cloud, and then verify data integrity by checking the correctness of signatures (e.g., RSA [7]) or hash values (e.g., MD5 [8])of the entire data. Certainly, this conventional approach able to successfully check the correctness of cloud data. However, the efficiency of using this traditional approach on cloud data is in doubt [9].The main reason is that the size of cloud data is largein general. Downloading the entire cloud data to verify data integrity will cost or even waste users amounts of computation and communication resources, especially when data have been corrupted in the cloud.

Recently, many mechanisms [9]–[16] have been proposed to allow not only a data owner itself but also a public verifier to efficiently perform integrity checking without downloading the entire data from the cloud, which is referred to as public auditing [5]. In these mechanisms, data is divided into many small blocks, where each block is independently signed by the owner; and a random combination of all the blocks instead of the whole data is retrieved during integrity checking [9].A public verifier could be a data user (e.g. researcher)who would like to utilize the owner's data via the cloud or a third-party auditor (TPA) who can provide expert integrity checking services. Existing public auditing mechanisms can actually be extended to verify shared data integrity and data freshness [1], [5].

However, a new significant privacy issue introduced in the case ofshared data with the use of existing mechanisms is the leakage of identity privacy to public verifiers [1].To protect theconfidential information, it is essential and critical topreserve identity privacy from public verifiers duringpublic auditing. To solve the above privacy issue onshared data, we propose Oruta1, a novel privacypreservingpublic auditing mechanism. More specifically, we utilize ring signatures to construct homomorphicauthenticators [10] in Oruta, so that a public verifier isable to verify the integrity of shared data without retrieving the entire data — while the identity of the signer oneach block in shared data is kept private from the publicverifier.

In addition, extend this mechanismto support batch auditing, which can perform multipleauditing tasks simultaneously and improve the efficiencyof verification for multiple auditing tasks. Meanwhile,Oruta is compatible with random masking [5]; **Oruta stands for "One Ring to Rule Them All".**

In this paper, to prove the data freshness (prove the cloud possesses the latest version of shared data)while still preserving identity privacy.Ensures that retrieved data always reflects the most recent updates and prevents rollback attacks.
Achieving data freshness is essential to protect against mis-configuration errors or rollbacks caused intentionally. We can develop an authenticated file system, that supports the migration of an enterprise-class distributes file system into the cloud efficiently, transparently and in a scalable manner. It's authenticated in the sense that enables an enterprise tenant to verify the freshness of retrieved data while performing the file system operations.

II. PROBLEM STATEMENTS

2.1 SYSTEM MODEL

As illustrated in Fig. 1, the system model in thispaper involves three parties: the cloud server, a groupof users and a public verifier. There are two types ofusers in a group: the original user and a number ofgroup users. The original user initially creates shareddata in the cloud, and shares it with group users. Boththe original user and group users are members of thegroup. Every member of the group is allowed to accessand modify shared data. Shared data and its verificationmetadata (i.e. signatures) are both stored in the cloudserver. A public verifier, such as a third-party auditor(TPA) providing expert data auditing services or a datauser outside the group intending to utilize shared data, isable to publicly verify the integrity of shared data storedin the cloud server.

When a public verifier wishes to check the integrityof shared data, it first sends an auditing challenge tothe cloud server. After receiving the auditing challenge,the cloud server responds to the public verifier with anauditing proof of the possession of shared data. Then,this public verifier checks the correctness of the entiredata by verifying the correctness of the auditing proof.Essentially, the process of public auditing is a challengeand-response protocol between a public verifier and thecloud server [9].

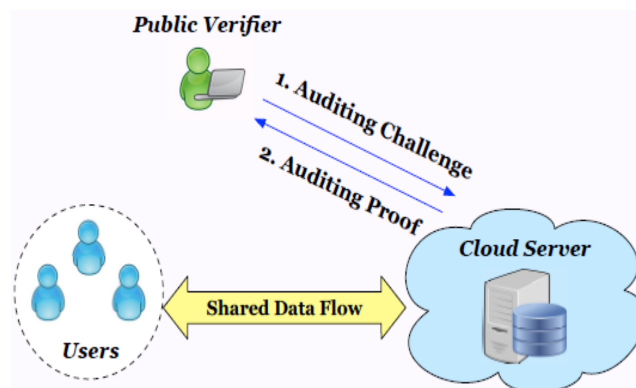


Fig.1 System model of three parties.

2.2 THREAT MODEL

2.2.1 Integrity Threats Two kinds of threats related to theintegrity of shared data are possible. First, an adversary may try to corrupt the integrity of shared data. Second,the cloud service provider may inadvertently corrupt(or even remove) data in its storage due to hardwarefailures and human errors. Making matters worse, thecloud service provider is economically motivated, whichmeans it may be reluctant to inform users about suchcorruption of data in order to save its reputation andavoid losing profits of its services.

2.2.2 Privacy Threats The identity of the signer on eachblock in shared data is private and confidential to thegroup. During the process of auditing, a public verifier,who is only allowed to verify the correctness of shareddata integrity, may try to reveal the identity of thesigner on each block in shared data based on verificationmetadata. Once the public verifier reveals the identity ofthe signer on each block, it can easily distinguish a highvaluetarget (a particular user in the group or a specialblock in shared data) from others.

2.3 DESIGN OBJECTIVES

Orutashould be designed to achievefollowing properties:

(1) **Public Auditing**: A public verifieris able to publicly verify the integrity of shared datawithout retrieving the entire data from the cloud.
(2)**Correctness**: A public verifier is able to correctly verifyshared data integrity.
(3) **Unforgeability**: Only a user inthe group can generate valid verification metadata (i.e., signatures) on shared data.
(4)**Identity Privacy**: A publicverifier cannot distinguish the identity of the signer on each block in shared data during the process of auditing**.**
(5)Data freshness: data freshness is essential to protect against mis-configuration errors or rollbacks caused intentionally. We can develop an authenticated file system that supports the migration of an enterprise-class distributed file system into the cloud efficiently, transparently and in a scalable manner. It's authenticated in the sense that enables an enterprise tenant to verify the freshness of retrieved data while performing the file system operations.

III. NEW RING SIGNATURE SCHEME

3.1 Overview

The design of new homomorphicauthenticable ring signature (HARS) scheme, which is extendedfrom a classic ring signature scheme [15]. The ringsignatures generated by HARS are not only able to preserveidentity privacy but also able to support block lessverifiability. We will show how to build the privacypreservingpublic auditing mechanism for shared datain the cloud based on this new ring signature scheme inthe next section.

3.2 Construction of HARS

HARS contains three algorithms: **KeyGen**, **RingSign**and **RingVerify**. In **KeyGen**, each user in the group generateshis/her public key and private key. In **RingSign**,a user in the group is able to generate a signature on ablock and its block identifier with his/her private keyand all the group members' public keys. A block identifieris a string that can distinguish the correspondingblock from others. A verifier is able to check whether agiven block is signed by a group member in **RingVerify**.

IV. PUBLIC AUDITING MECHANISM

4.1 Overview

Using HARS and its properties, we now construct Oruta, a privacypreservingpublic auditing mechanism for shared datain the cloud. With Oruta, the public verifier can verifythe integrity of shared data without retrieving the entiredata. Meanwhile, the identity of the signer on each blockin shared data is kept private from the public verifierduring the auditing.

4.2 Construction of Oruta

Now, we present the details of our public auditingmechanism. It includes five algorithms: **KeyGen**,**SigGen**, **Modify**, **ProofGen**and **ProofVerify**. In **Key-Gen**, users generate their own public/private key pairs.In **SigGen**, a user (either the original user or a groupuser) is able to compute ring signatures on blocks inshared data by using its own private key and all thegroup members' public keys. Each user in the group isable to perform an insert, delete or update operationon a block, and compute the new ring signature onthis new block in **Modify**. **ProofGen**is operated by apublic verifier and the cloud server together to interactivelygenerate a proof of possession of shared data.
In **ProofVerify**, the public verifier audits the integrity ofshared data by verifying the proof.Note that for the ease of understanding, we first assume the group is static, which means the group is predefinedbefore shared data is created in the cloud andthe membership of the group is not changed during datasharing. Specifically, before the original user outsources shared data to the cloud, he/she decides all the groupmembers.
Dynamic Groups: We now discuss the scenario ofdynamic groups under our proposed mechanism. If anew user can be added in the group or an existinguser can be revoked from the group, then this groupis denoted as a dynamic group. To support dynamicgroups while still allowing the public verifier to performpublic auditing, all the ring signatures on shared dataneed to be re-computed with the signer's private key andall the current users' public keys when the membershipof the group is changed.

4.3 Security Analysis of Oruta

Now, we discuss security properties of Oruta, includingits correctness, unforgeability, identity privacy anddata privacy.

Theorem: A public verifier is able to correctly audit theintegrity of shared data under Oruta.
Proof: According to the description of **ProofVerify, a** public verifier believes the integrity of shared data iscorrect if Equation 6 holds. So, the correctness of ourscheme can be proved by verifying the correctness ofEquation 6. Based on properties of bilinear maps andTheorem 1, the right-hand side (RHS) of Equation 6 canbe expanded as follows:

$$
\begin{aligned}
\text{RHS} \models & \left(\prod_{i=1}^{d} e(\prod_{j\in\mathcal{J}} \sigma_{j,i}^{y_j}, w_i)\right) \cdot e(\prod_{l=1}^{k} \lambda_l^{h(\lambda_l)}, g_2) \\
= & \left(\prod_{j\in\mathcal{J}} (\prod_{i=1}^{d} e(\sigma_{j,i}, w_i)^{y_j})\right) \cdot e(\prod_{l=1}^{k} \eta_l^{\tau_l h(\lambda_l)}, g_2) \\
= & \left(\prod_{j\in\mathcal{J}} e(\beta_j, g_2)^{y_j}\right) \cdot e(\prod_{l=1}^{k} \eta_l^{\tau_l h(\lambda_l)}, g_2) \\
= & e(\prod_{j\in\mathcal{J}} (H_1(id_j) \prod_{l=1}^{k} \eta_l^{m_{j,l}})^{y_j}, g_2) \cdot e(\prod_{l=1}^{k} \eta_l^{\tau_l h(\lambda_l)}, g_2) \\
= & e(\prod_{j\in\mathcal{J}} H_1(id_j)^{y_j} \cdot \prod_{l=1}^{k} \eta_l^{\sum_{j\in\mathcal{J}} m_{j,l} y_j} \cdot \prod_{l=1}^{k} \eta_l^{\tau_l h(\lambda_l)}, g_2) \\
= & e(\prod_{j\in\mathcal{J}} H_1(id_j)^{y_j} \cdot \prod_{l=1}^{k} \eta_l^{\mu_l}, g_2).
\end{aligned}
$$

4.4 Batch Auditing

Sometimes, a public verifier may need to verify thecorrectness of multiple auditing tasks in a very shorttime. Directly verifying these multiple auditing tasksseparately would be inefficient. By leveraging the propertiesof bilinear maps, we can further extend Orutatosupport batch auditing, which can verify the correctnessof multiple auditing tasks simultaneouslyand improvethe efficiency of public auditing.

Assume the integrity of B auditing tasks need to be verified, where the B corresponding shared data are denoted as $M_1, ..., M_B$, the number of users sharing data M_b is described as d_b, where $1 \le b \le B$.

BatchProofGen. A public verifier first generates an auditing challenge $\{(j, y_j)\}_{j\in\mathcal{J}}$ as in **ProofGen**. After receiving the auditing challenge, the cloud server generates and returns an auditing proof $\{\boldsymbol{\lambda}_b, \boldsymbol{\mu}_b, \boldsymbol{\phi}_b, \{id_{b,j}\}_{j\in\mathcal{J}}\}$ for each shared data M_b as in **ProofGen**, where $1 \le b \le B$, $1 \le l \le k$, $1 \le i \le d_b$ and

$$
\begin{cases}
\lambda_{b,l} = \eta_{b,l}^{\tau_{b,l}} \\
\mu_{b,l} = \sum_{j\in\mathcal{J}} y_j m_{b,j,l} + \tau_{b,l} h(\lambda_{b,l}) \\
\phi_{b,i} = \sum_{j\in\mathcal{J}} \sigma_{b,j,i}^{y_j}
\end{cases}
$$

Here $id_{b,j}$ is described as $id_{b,j} = \{f_b, v_{b,j}, r_{b,j}\}$, where f_b is the data identifier (e.g., file name) of shared data M_b.

BatchProofVerify. After receiving all the B auditing proofs, the public verifier checks the correctness of these B proofs simultaneously by checking the following equation with all the $\sum_{b=1}^{B} d_b$ users' public keys:

$$
\begin{aligned}
& e(\prod_{b=1}^{B} \left(\prod_{j\in\mathcal{J}} H(id_{b,j})^{y_j} \cdot \prod_{l=1}^{k} \eta_{b,l}^{\mu_{b,l}}\right), g_2) \\
\stackrel{?}{=} & \left(\prod_{b=1}^{B} \prod_{i=1}^{d_b} e(\phi_{b,i}, w_{b,i})\right) \cdot e(\prod_{b=1}^{B} \prod_{l=1}^{k} \lambda_{b,l}^{h(\lambda_{b,l})}, g_2), \quad (7)
\end{aligned}
$$

where $\mathbf{pk}_{b,i} = w_{b,i} = g^{x_{b,i}}$ and $\mathbf{sk}_{b,i} = x_{b,i}$. If the above verification equation holds, then the public verifier believes that the integrity of all the B shared data is correct. Otherwise, there is at least one shared data is corrupted.

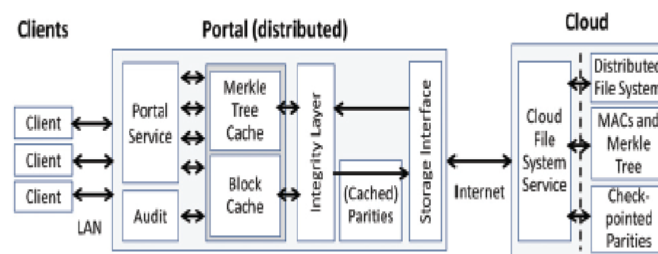
4.5 Data freshness verification

We extend the construction of oruta with data freshness in the authenticated file system that verify the freshness of any data retrieved from the file system while performing typical file system operations. Freshness ensures that the latest version of the data is always retrieved (and thus prevents rollback attacks reverting the file system state to a previous version. Another challenge is efficient management and caching of the authenticating information.

Freshness verification should be extremely efficient for existing file system operations and induce minimal latency. To ensure freshness, it is necessary to authenticate not just data blocks, but also their *versions*. Each block has an associated version counter that is incremented every time the block is modified. This version number is bound to the file-block's MAC: To protect against cloud replay of stale file-blocks (rollback attacks), the counters themselves must be authenticated.

4.5.1System architecture

The cloud maintains the distributed file system, consisting of all files and directories belonging to enterprise users. Iris is designed to use any existing cloud storage system transparently in the back end without modification. In addition, the cloud also stores the MACs and Merkle tree necessary for authenticating data, as well as the checkpointed parity information needed to recover from potential corruptions at the portal. As an additional resilience measure the parity information could be stored on a different cloud or replicated internally within the enterprise.



4.5.2 Solution overview and challenges

It consists of two major components:

Authenticated file system: As already described, the first challenge we address in building an authenticated enterprise-class file system is the high cost of network latency and bandwidth between the enterprise and cloud. Another challenge is efficient management and caching of the authenticating information. Integrity and freshness verification should be extremely efficient for existing file system operations and induce minimal latency. To guarantee data freshness for the entire file system, an authentication scheme consisting of two layers. At the lowest layer, it stores a MAC for each file block (file blocks are fixed size file segments typical size 4KB). This enables random access to file blocks and a verification of individual file block without accessing full files.Forfreshness,MACs are not sufficient.Instead, that associates a counter or version number with each file block that is incremented on every block update and included in the block MAC[12].Different versions of a block can be distinguished through different version numbers. But for freshness, block version numbers need to be authenticated too! The upper layer of the authentication scheme is a Merkle tree tailored to the file system directory tree. The leaves of the Merkle tree store block version numbers in a compacted form. The authentication of data is separated from the authentication of block version numbers to enable various optimizations in the data structure. Internal nodes of the tree contain hashes of children as in a standard Merkle tree. The root of the Merkle tree needs to be maintained at all times within the enterprise trust boundary at the gateway.

The tenant can efficiently verify the freshness of a file data block by checking the block MAC and the freshness of the block version number. The tenant verifies the later by accessing the sibling nodes on the path from the leaf storing the version number up to the root of the tree, re-computing all hashes on the path to the root and checking that the root matches the value stored locally. With the similar mechanism the tenant can additionally verify the correctness of file paths in the file system and more generally of any other file system meta data (file names, number of files in a directory, file creation system,etc.).

This Merkle tree based structure has two distinctive features compared to other authenticated file systems: (1) Support for existing file system operations: that maintains a balanced binary tree over the file system directory structure to efficiently support existing file system calls; and (2) Support for concurrent operation: the Merkle tree supports efficient updates from multiple clients operating on a file system in parallel. It also optimizes for sequential file block accesses: sequences of identical version counters are compacted into a single leaf. The authenticated mechanism is practical and scalable: in a prototype system, the use of a Merkle tree catch of only 10 MB increases the system throughput by a factor of 3(compared to no catching employed),the throughput is fairly constant for about 100 clients executing operations on the file system in parallel and the operation latency overhead introduced by processing at the gateway is at most 15%.These numbers are reported from a user level implementation of sequential file reads and writes, and archiving of an entire directory structure

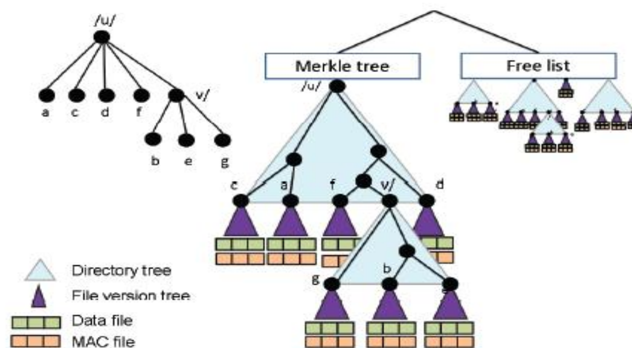


Fig. 2 Authenticated data structure.

4.5.3Authentication process in AFS:

We describe in this section with data freshness, for both file system data and meta-data. The authentication scheme is based on Merkle trees, and designed to support existing file system operations. In addition, random access to files for both read and writes operations are a desirable feature that we also choose to implement. The tenant needs to maintain at all times the root of the Merkle trees for checking the freshness of data retrieved from the cloud. For reducing operation latency, recently accessed nodes in the tree are also cached at the portal.

(a) Block-levelMACs:

For providing freshness, we need to bind a unique version number to each file block every time it's updated and include the version number in the block MAC. To protect against rollback attacks (in which clients are presented with an old state of the file system), version numbers will have to be authenticated as well.

(b) File version trees:

We construct a *file version tree* per file that authenticates version numbers for all file blocks in a compressed form. Briefly, the file version tree compresses the versions of a consecutive range of blocks into a single node, storing the index range of the blocks and their common version number. File version trees are optimized for sequential access to files. For instance, if a file is always written sequentially then its file version tree consists of only one root node. The compacted version tree essentially behaves as a range tree data structure.

(c) Directory trees:

To authenticate file system meta-data (or the directory structure of the file system), the file system directory tree is transformed into a Merkle tree in which every directory is mapped to a *directory subtree*. We have chosen to map our authenticated data structure onto the existing file system tree in order to efficiently support file system operations like delete or move of entire directories.

To support directories with large number of files efficiently, we create a balanced binary tree for each directory that contains file and subdirectory nodes in the leaves, and includes intermediate, empty internal nodes for balancing. Nodes in a directory tree have unique identifiers assigned to them, chosen as random strings of fixed length. A leaf for each file and subdirectory is inserted into the directory tree in a position given by a keyed hash applied to its name and its parent's identifier (to ensure tree balancing). At the leaves of the directory tree, we insert the file version trees in compacted form, as described above. Internal nodes in the

Merkle tree contain hash values computed over their children, as well as some additional information, e.g., node identifiers, their rank (defined as the size of the subtree rooted at the node), file and directory names.
Our Merkle tree supports the following operations. Clients can insert or delete file system object nodes (files or directories) at certain positions in the tree. Those operations trigger updates of the hashes stored on the path from the inserted/deleted nodes up to the root of the tree. Deleted subtrees are added to the free list, as explained below. Clients can verify a file block version number, by retrieving all siblings on the path from the leaf corresponding to that file block up to the root of the tree. Searches of files or directories in the tree can also be performed, given absolute path names. We also implement an operation randompath-dir-tree for directory trees. This feature is needed to execute the challenge-response protocols of the auditing component in Iris. A (pseudo)-random path in the tree is returned by traversing the tree from the root, and selecting at each node a child at random, weighted by rank.

In addition, the authentication information for the random path is returned, so the tenant can verify that the path has been chosen pseudo-randomly. With this Merkle tree construction, we authenticate both file system meta-data, as well as file block version numbers. Together with the file block MACs, this mechanism ensures data integrity and freshness, assuming that the portal always stores the root of the Merkle tree.

(d) Free list:

As an optimization, we also maintain in the data structure a *free list* containing pointers of nodes deleted from the data structure, i.e., sub trees removed as part of delete or truncate operations. The aim of the free list is to defer garbage collection of deleted nodes and support remove and truncate file system operations efficiently. We omit further details due to space limitations.

V. CONCLUSION AND FUTURE WORK

In this paper, we propose a privacy-preserving public auditing with data freshness verificationmechanism for shared data in the cloud. Freshness verification should be extremely efficient for existing file system operations and induce minimal latency.

To ensure freshness, it is necessary to authenticate not just data blocks, but also their *versions*. Each block has an associated version counter that is incremented every time the block is modified. This version number is bound to the file-block’s MAC: To protect against cloud replay of stale file-blocks (rollback attacks), the counters themselves must be authenticated.

The interesting problems we will continue to study for our future work. One of them is traceability, which means the ability for the group manager (i.e., the original user) to reveal the identity of the signer based on verification metadata in some special situations.Since the current design of ours does not support traceability.

REFERENCES

[1] B. Wang, B. Li, and H. Li, “Oruta: Privacy-Preserving Public Auditing for Shared Data in the Cloud,” in Proceedings of IEEE Cloud 2012, 2012, pp. 295–302.
[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “A View of Cloud Computing,” Communications of the ACM, vol. 53, no. 4, pp. 50–58, April 2010.
[3] K. Ren, C. Wang, and Q. Wang, “Security challenges for the Public Cloud,” IEEE Internet Computing, vol. 16, no. 1, pp. 69–73, 2012.
[4] D. Song, E. Shi, I. Fischer, and U. Shankar, “Cloud Data Protection for the Masses,” IEEE Computer, vol. 45, no. 1, pp. 39–45, 2012.
[5] C. Wang, Q. Wang, K. Ren, and W. Lou, “Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing,” in Proceedings of IEEE INFOCOM 2010, 2010, pp. 525–533.
[6] B. Wang, M. Li, S. S. Chow, and H. Li, “Computing Encrypted Cloud Data Efficiently under Multiple Keys,” in Proc. of CNSSPCC’ 13, 2013, pp. pp.90–99.
[7] R. Rivest, A. Shamir, and L. Adleman, “A Method for Obtaining Digital Signatures and Public Key Cryptosystems,” vol. 21, no. 2, pp. 120–126, 1978.
[8] The MD5 Message-Digest Algorithm (RFC1321). [Online].Available: https://tools.ietf.org/html/rfc1321
[9] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, “Provable Data Possession at Untrusted Stores,” in Proceedings of ACM CCS’07, 2007, pp. 598–610.
[10] H. Shacham and B. Waters, “Compact Proofs of Retrievability,” in Proceedings of ASIACRYPT’08. Springer-Verlag,2008,pp.90–107.
[11] C. Erway, A. Kupcu, C. Papamanthou, and R.Tamassia, “Dynamic Provable Data Possession,” in Proceedings of ACM CCS’09, 2009,pp. 213–222.

[12] Q. Wang, C. Wang, J. Li, K. Ren"Enabling Public Verifiability and Data Dynamic for Storage Security in Cloud Computing," in Proceedings of ESORICS 2009. Springer-Verlag, 2009, pp. 355–370.
[13] C. Wang, Q. Wang, K. Ren, and W. Lou, "Ensuring Data Storage Security in Cloud Computing," in Proceedings of ACM/IEEE IWQoS'09, 2009, pp. 1–9.
[14] B. Chen, R. Curtmola"Remote Data Checking for Network Coding-based Distributed Stroage Systems," in Proceedings of ACM CCSW 2010, 2010, pp. 31–42.
[15] Y. Zhu, H.Wang "Dynamic Audit Services for Integrity Verification of Outsourced Storage in Clouds," in Proceedings of ACM SAC 2011, 2011, pp. 1550–1557.
[16] N. Cao, S. Yu, Z. Yang, W. Lou, and Y. T. Hou, "LT Codes-basedSecure and Reliable Cloud Storage Service," in Proceedings of IEEEINFOCOM 2012, 2012.