IOSR Journal of Computer Engineering (IOSRJCE)
ISSN: 2278-0661 Volume 1, Issue 6 (July-Aug. 2012), PP 43-49
www.iosrjournals.org

Securing Cloud Data Storage

S. P. Jaikar[1], M. V. Nimbalkar[2]
[1,2]*(Department of Information Technology, Sinhgad College of Engineering/ University of Pune, India)*

Abstract : *Innovations are necessary to ride the inevitable tide of change. Most of enterprises are striving to reduce their computing cost through the means of virtualization. This demand of reducing the computing cost has led to the innovation of Cloud Computing. One fundamental aspect of this new computing is that data is being centralized or outsourced into the cloud. From the data owners perspective, including both individuals and IT enterprises, storing data remotely in a cloud in a flexible on-demand manner brings appealing benefits: relief of the burden of storage management, universal data access with independent geographical locations, and avoidance of capital expenditure on hardware, software, personnel maintenance, and so on although the infrastructures under the cloud are much more powerful and reliable than personal computing devices, they still face a broad range of both internal and external threats to data integrity. Outsourcing data into the cloud is economically attractive for the cost and complexity of long-term large scale data storage, it does not offer any guarantee on data integrity and availability. We propose a distributed scheme to ensure users that their data are indeed stored appropriately and kept intact all the time in the cloud. We are using erasure correcting code in the file distribution preparation to provide redundancies. We are relaying on challenge response protocol along with pre-computed tokens to verify the storage correctness of user's data & to effectively locate the malfunctioning server when data corruption has been detected. Our scheme maintains the same level of storage correctness assurance even if users modify, delete or append their data files in the cloud.*

Keywords *- Cloud computing, Distributed data storage, Data security, Pervasive Computing, Virtualization.*

I. INTRODUCTION

Cloud Computing moves the data & application software's to the large data centers, where the management of the data and services may not be fully trustworthy. This poses many new security challenges which have not been well understood. Cloud computing inevitably poses new challenging security threats for number of reasons.

1. Due to user's loss of control over the data under cloud, we cannot directly adopt the traditional cryptographic primitives for the purpose of data security protection. Therefore, verification of correct data storage in the cloud must be conducted without explicit knowledge of the whole data. Considering various kinds of data for each user stored in the cloud and the demand of long term continuous assurance of their data safety, the problem of verifying correctness of data storage in the cloud becomes even more challenging.
2. Cloud Computing is not just a third party data warehouse. User's may frequently update stored data by performing operations like insertion, deletion, modification, appending, reordering, etc. To ensure storage correctness under dynamic data update is hence of paramount importance. However, this dynamic feature also makes traditional integrity insurance techniques futile.
3. The deployment of Cloud Computing is powered by data centers running in a simultaneous, cooperated and distributed manner. Individual user's data is redundantly stored in multiple physical locations to further reduce the data integrity threats. Therefore, distributed protocols for storage correctness assurance will be of most importance in achieving a robust and secure cloud data storage system in the real world.

Our goal is to focus on cloud data storage security & to ensure the correctness of user's data in the cloud. We aim to localize the errors & to perform successful recovery of data, as well as to provide support for dynamic operations on data.

Recently, the importance of ensuring the remote data integrity has been highlighted by many research works [1-4, 14]. These techniques, while can be useful to ensure the storage correctness without having users possessing data, cannot address all the security threats in cloud data storage, since they are all focusing on single server scenario and most of them do not consider dynamic data operations. As a complementary approach, researchers [5, 10, 15] have also proposed distributed protocols for ensuring storage correctness across multiple servers or peers. Again, none of these distributed schemes is aware of dynamic data operations. As a result, their applicability in cloud data storage can be drastically limited.

Prior work [16] has addressed this storage security problem using either public key cryptography or requiring the client to outsource its data in encrypted form which is not a feasible solution. Also some provable data possession schemes [2, 3, 8, 9] are proposed but they have limitations like they need to migrate whole data block in case of updating data block. Previous schemes [1-5, 8, 10] are not able to support dynamic data operations which are very important in cloud computing. Some schemes [7, 17] need third parties to conduct the audits which are again not the better option. The Challenge Response protocol is almost used in every scheme with little modifications but they are having drawbacks of having large token size which puts burden on clients.

II. SYSTEM ARCHITECTURE

The general architecture of cloud storage system is illustrated in Fig.1. Generally two different network entities can be identified. We have assumed that user's have direct peer to peer connection between them & cloud. Users will upload their data to cloud and only they can access it & not any other cloud users. Different network entities are mentioned below:

- User: users, who have data to be stored in the cloud and rely on the cloud for data computation, consist of both individual consumers and organizations.
- Cloud Service Provider (CSP): CSP is who has the capabilities to host data & applications of users. They have huge resources that they can provide dynamically for satisfying various user needs. CSP having expertise in building & managing cloud servers, having their own data centers for hosting user's data.

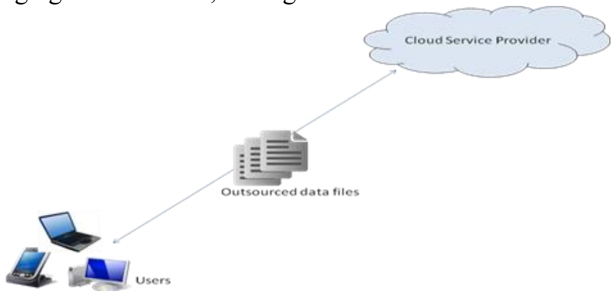

Fig.1: Storage Architecture for Cloud

Fig.1 shows how the data is outsourced in cloud and users have no control over it. This also gives perception of the problem with the storage and to ensure the integrity of the data in the cloud. In cloud data storage, a user stores his data through a CSP into a set of cloud servers, which are running in a simultaneous, cooperated and distributed manner. Data redundancy can be employed with technique of erasure-correcting code to further tolerate faults or server crash as user's data grows in size and importance. Thereafter, for application purposes, the user interacts with the cloud servers via CSP to access or retrieve his data. In some cases, the user may need to perform operations on his data.

The most general forms of these operations we are considering are update, delete, insert and append. As users no longer possess their data locally, it is of critical importance to assure users, that their data are being correctly stored and maintained. That is, users should be equipped with security means so that they can make continuous correctness assurance of their stored data even without the existence of local copies.

In case those users do not necessarily have the time, feasibility or resources to monitor their data, user's can delegate the tasks to an optional trusted TPA of their respective choices. But users need to pay to the Third Party Auditors for that. This is not our aim, what we want is to give freedom to users to ensure intactness of their data in cloud. In our scheme, we assume that the point-to-point communication channels between each cloud server and the user is authenticated and reliable. Security threats faced by cloud data storage can come from two different sources. On the one hand, a CSP can be self-interested, untrusted and possibly malicious. It may also attempt to hide a data loss incident due to management errors, Byzantine failures and so on. On the other hand, there may also exist an economically motivated adversary, who has the capability to compromise a number of cloud data storage servers in different time intervals and subsequently is able to modify or delete user's data while remaining undetected by CSPs for a certain period. So we have attackers with different purposes in different context & we need to classify them as per the severity of damage they can do to storage.

Depending upon various motivations of attackers we have classified them into categories. Specifically, we consider two types of adversary with different levels of capability:

- *Weak Adversary*: The adversary is interested in corrupting the user's data files stored on individual servers. Once a server is comprised, an adversary can pollute the original data files by modifying or introducing its own fraudulent data to prevent the original data from being retrieved by the user.
- *Strong Adversary*: This is the worst case scenario, in which we assume that the adversary can compromise all the storage servers so that he can intentionally modify the data files as long as they are internally

consistent. In fact, this is equivalent to the case where all servers are colluding together to hide a data loss or corruption incident.

To ensure the security for cloud data storage under the aforementioned adversary model, we aim to design efficient mechanisms for dynamic data verification and operation.

2.1 Notation & Preliminaries

- F – The data file to be stored. We assume that F can be denoted as a matrix of m equal-sized data vectors, each consisting of l blocks. Data blocks are all well represented as elements in Galois Field $GF(2^w)$ for $w = 4, 8, 16$.
- R – The dispersal matrix used for Reed-Solomon coding.
- D – Data matrix constructed over data vectors.
- C – The encoded file matrix, which includes a set of $n = m + k$ vectors, each consisting of l blocks.
- PRF – Pseudorandom function.
- PRP – Pseudorandom permutation.

2.2 File Distribution Preparation

It is well known that erasure-correcting code may be used to tolerate multiple failures in distributed storage systems [18]. In cloud data storage, we rely on this technique to disperse the data file F redundantly across a set of $n = m + k$ distributed servers. $R(m + k, k)$ Reed-Solomon erasure-correcting code is used to create k redundancy parity vectors from m data vectors in such a way that the original m data vectors can be reconstructed from any m out of the $m + k$ data and parity vectors. By placing each of the $m + k$ vectors on a different server, the original data file can survive the failure of any k of the $m + k$ servers without any data loss, with a space overhead of k/m. For support of efficient sequential I/O to the original file, our file layout is systematic, i.e., the unmodified m data file vectors are distributed across $m + k$ different servers. We are using Reed Solomon Algorithm to disperse the file redundantly over m storage devices.

2.2.1 RS-RAID ALGORITHM

Let there be n storage devices $(D_1, D_2, D_3, \ldots\ldots, D_n)$ each of which holds k bytes. These are called the Data Devices. Let there be m more storage devices $(C_1, C_2, C_3, \ldots\ldots, C_m)$ each of which also holds k bytes. These are called the Checksum Devices. The contents of each checksum device will be calculated from the contents of the data devices. The goal is to define the calculation of each c_i such that if any m of $(D_1, D_2, D_3, \ldots\ldots, D_n, C_1, C_2, C_3, \ldots\ldots, C_m)$ fail, then the contents of the failed devices can be reconstructed from the non-failed devices. In other words we have n data words $d_1, d_2, d_3, \ldots\ldots, d_n$ and m checksum words $c_1, c_2, c_3, \ldots\ldots, c_m$ which are computed from the data words in such a way that the loss of any m words can be tolerated.

To compute a checksum word c_i for the checksum device C_i, we apply function P_i to the data words:

$$c_i = P_i\,(d_1, d_2, \ldots.., d_n)$$

If a data word on device D_j is updated from d_j to d_j' , then each checksum word c_i is recomputed by applying a function $G_{i,j}$ such that

$$c_j' = G_{i,j}\left(d_j, d_j', c_i\right)$$

When up to m devices fail, we reconstruct the system as follows. First, for each failed data device D_j, we construct a function to restore the words in D_j from the words in the non-failed devices. When that is completed, we recomputed any failed checksum devices C_i with P_i.

For example, suppose $m = 1$. We can describe $n + 1$ parity in the above terms. There is one checksum device C_1. To compute each checksum word c_1, we take the parity (XOR) of the data words:

$$c_1 = P_1\,(d_1, d_2, \ldots.., d_n) = d_1 \oplus d_2 \oplus \ldots\ldots.\oplus d_n$$

If a word on data device D_j changes from d_j to d_j' , then c_1 is recalculated from the parity of its old value and the two data words:

$$c_1' = G_{1,j}\left(d_j, d_j', c_1\right) = c_1 \oplus d_j \oplus d_j'$$

If a device D_j fails, then each word may be restored as the parity of the corresponding words on the remaining devices:

$$d_j = d_1 \oplus d_2 \oplus \ldots\ldots . \oplus d_n \oplus c_1$$

In such a way, the system is resilient to any single device failure. We are given n data words $d_1, d_2, \ldots\ldots d_n$. We define functions P and G which we use to calculate and maintain the checksum words $c_1, c_2, \ldots\ldots c_m$. We then describe how to reconstruct the words of any lost data device when up to m devices fail. Once the data words are reconstructed, the checksum words can be recomputed from the data words and P. Thus, the entire system is reconstructed.

2.2.2 Calculating & Maintaining Checksums

We define each function P_i to be a linear combination of the data words:

$$c_i = P_i\left(d_1, d_2, \ldots\ldots, d_n\right) = \sum_{j=1}^{n} d_j P_{i,j}$$

In other words, if we represent the data and checksum words as the vectors D and C, the functions P_i as rows of the matrix P, then the state of the system adheres to the following equation:

$PD = C$

We define to be the $m \times n$ matrix $p_{i,j} = j^{i-1}$, and thus the above equation becomes:

$$\begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,n} \\ \vdots & \vdots & & \vdots \\ p_{m,1} & p_{m,2} & \cdots & p_{m,n} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 2 & \cdots & n \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 2^{m-1} & \cdots & n^{m-1} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}$$

When one of the data words d_j changes to d_j', then each of the checksum words must be changed as well. This can be affected by subtracting out the portion of the checksum word that corresponds to d_j, and adding the required amount for d_j' .

Thus, $G_{i,j}$ is defined as follows:

$c_i' = G_{i,j}\left(d_j, d_j', c_i\right) = c_i + p_{i,j}(d_j' - d_j)$

Therefore, the calculation and maintenance of checksum words can be done by simple arithmetic.

2.2.3 RECOVERING FROM FAILURES

To explain recovery from errors, we define the matrix A and the vector E as follows:

$A = \begin{bmatrix} I \\ P \end{bmatrix}$ *and* $E = \begin{bmatrix} D \\ C \end{bmatrix}$. Then we have the following equation $AD = E$

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 3 & \cdots & n \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & 2^{m-1} & 3^{m-1} & \cdots & n^{m-1} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}$$

We can view each device in the system as having a corresponding row of the matrix A and the vector E. When a device fails, we reflect the failure by deleting the device's row from A and from E. What results a new matrix A' and a new vector E' that adhere to the equation:

$A'D = E'$

Suppose exactly m devices fail. A' is $n \times n$ matrix. Because matrix P is defined to be a Vandermonde matrix, every subset of n rows of matrix A is guaranteed to be linearly independent. Thus, the matrix A' is non-singular, and the values of D may be calculated from $A'D = E'$ using Gaussian Elimination. Hence all data devices can be recovered.

Once the values of D are obtained, the values of any failed C_1 may be recomputed from D. It should be obvious that if fewer than m devices fail; the system may be recovered in the same manner, choosing any n rows of A'. Thus system can tolerate any number of device failures up to m.

So, as per RS Raid algorithm, we divide the input file to the n data vectors, where n is number of storage devices present in the system. The data vectors that are generated are of equal size, so the load will be distributed equally to all the storage devices. We create $m \times n$ matrix D & store all the data vectors in matrix D. In next step we create a Reed Solomon matrix R which is generated over Galois field, i. e. $GF(2^w)$. In our case we have assumed word size $w = 4$. After this stage, we perform matrix multiplication to generate checksum matrix C. We multiply data matrix D with Reed Solomon matrix R. The resultant matrix is the redundant matrix which contains original data from data matrix D & parity vectors added by Reed Solomon matrix. It means matrix D will be stored redundantly across the different storage devices & it will be used for token computation as well as data recovery purpose.

2.3 CHALLENGE TOKEN PRECOMPUTATION

To verify the correctness of user's data & to locate the errors, we entirely rely on the pre-computed verification tokens. These tokens are calculated before file distribution & they are very short. We are computing the tokens by pseudorandom function PRF & pseudorandom permutation function PRP. We pre-computes short verification tokens on individual vector, each token covering a random subset of data blocks. We have assumed block size as 256 bits & r as 8 number of verification per indices. We have three data devices and three checksum devices. Then $\mathrm{n} = 3$ and $\mathrm{m} = 3$. We choose $\mathrm{w} = 4$, since $2^{\mathrm{w}} > \mathrm{n} + \mathrm{m}$. Next, we set gflog and gfilog table's. gflog and gfilog tables are shown in Table 1.

i																
gfl																
gfi																

Table 1: gflog and gfilog tables for $\mathrm{GF}(2^4)$.

We construct P to be a 3×3 matrix, defined over $GF(2^4)$.

$$P = \begin{bmatrix} 1^0 & 2^0 & 3^0 \\ 1^1 & 2^1 & 3^1 \\ 1^2 & 2^2 & 3^3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 4 & 5 \end{bmatrix}$$

Now, we can calculate each word of each checksum device using $PD = C$

Later, when the user wants to make sure the storage correctness for the data in the cloud, he challenges the cloud servers. Upon receiving challenge, cloud server computes the new value of tokens, which is compared with previously calculated tokens. It gives clear idea about integrity of user's data. Again it helps to locate the error which has not been done in previous research work [1, 2, 3, 4, 8]. In previous work, we were just able to detect whether the data is intact or not. So it just provides us with binary results & not the exact location of errors.

Algorithm: TOKEN PRE-COMPUTATION

```
1. Begin
2. Choose file $F$ to upload & encrypt the file using $AES$.
3. Generate $n \times m$ Vector Matrix $D$ on file $F$.
4. Create Reed Solomon Matrix $P$ over Galois Field $GF(2^w)$. where $w = 4$.
5. Generate Matrix $C = D \times P$. It is Checksum Matrix created for fault tolerance.
6. Compute Token over Matrix $C$ i.e. $ComputeToken\ (C, l, t, r)$, where $l -$ block size, $t -$ no. of tokens, $r -$
   indices per verification. Compute the tokens by pseudorandom function $PRF$ & pseudorandom permutation
   function $PRP$.
7. Store these precomputed tokens on the main cloud server.
8. Disperse the file over the Cloud. i.e. $DisperseFile\,(Matrix\ \ D)$
9. End.
```

2.4 CORRECTNESS VERIFICATION & ERROR LOCALIZATION

To eliminate the errors in storage systems key prerequisite is to locate the errors. However, many previous schemes do not explicitly consider the problem of data error localization, thus only provide binary results for the storage verification. In our scheme we integrate the correctness verification and error localization in our challenge-response protocol. The newly computed tokens from servers for each challenge are compared with pre-computed tokens to determine the correctness of the distributed storage. This also gives information to locate potential data errors.

Algorithm: CORRECTNESS VERIFICATION

```
1. Begin Challenge $i$, for $i = (i = 1\ \ to\ \ n)$, where $n$ – total number of cloud servers.
2. Get $TokenArr()$ // Getting precomputed tokens from main cloud server.
3. $HandleChallenge()$ // Reading file blocks from all cloud servers for calculating new tokens.
4. Generate Vector Matrix $D$ on all file blocks that are read in step 3.
5. Create Reed Solomon Matrix $P$
6. Generate Matrix $C = D \times P$. On this matrix, new tokens will be computed.
7. Compute token on Matrix $C$. $ComputeToken\ (C, l, t, r)$
8. If (($Precomputed\ token == newly\ computed\ token$) then,
                Data is intact Else
        Data is Corrupt. For that $i$, initiate the recovery.
9. End
```

2.5 ERROR RECOVERY & FILE RETRIEVAL

Once the data corruption is detected, next important step is to recover the corrupted data and bring data storage back to consistent state. The comparison of pre-computed tokens and received response values can guarantee the identification of misbehaving server. Therefore user can recover the corrupted data. Our system recovers data from backup server & distributes all data vectors to corresponding servers. This will results in successful recovery of corrupted data. But due to file splitting we made at the time of file distribution, user's need to recover file from all the servers. Error localization is limited to misbehaving servers only, i.e. servers giving false assurance of posing user's data.

Algorithm: Error Recovery

```
1. Begin (Assume that the data corruptions have been detected & $s \leq k$ servers have been identified
   misbehaving.)
2. Download consistent data blocks from backup server.
3. Create the data vectors as per number of cloud storage servers.
4. Distribute the consistent data blocks to corresponding servers & recover the data.
5. End.
```

2.6 DYNAMIC OPERATIONS

In cloud data storage, there are many potential scenarios where data stored in the cloud is dynamic, like electronic documents, photos, or log files etc. Therefore, it is crucial to consider the dynamic case, where a user may wish to perform various operations of update, delete and append to modify the data file while maintaining the storage correctness assurance. The straightforward and trivial way to support these operations is for user to download all the data from the cloud servers and re-compute the whole parity blocks as well as verification tokens. This would clearly be highly inefficient. In cloud data storage, sometimes the user may need to modify some data stored in the cloud, from its current value to a new one. We refer this operation as data update. To perform update operation on particular data block client need to recalculate the verification token on updated data. Also client need to update this value of newly calculated token to all the replicas of file in storage cloud. When user want to perform update operation, the splitted file from all storage servers is merged and given to the user to perform data updates. Once user has finished with the updating the data, new tokens are calculated on whole file and they are stored on main cloud server. After this, updated file is splitted back and dispersed onto corresponding cloud storage servers. Update operations include modifying file, inserting data, as well as deleting data from file.

Sometimes, after being stored in the cloud, certain data may need to be deleted. The delete operation we are considering is a general one. When user wants to delete some file, he can simply delete it. In delete operation, file blocks that are distributed among cloud storage servers are all deleted. Once file is deleted, we cannot perform any recovery of deleted files as there won't be any backup available in main cloud server. In some cases, the user may want to increase the size of his stored data in file by adding data at the end of the data file, which we refer as data append operation. So in case of append operation whenever user append data to his file, new verification tokens are calculated & stored on main cloud server & file is splitted as before and dispersed among the cloud storage servers. Previous schemes [5, 6, 7] don't support insert operation on data. In our scheme user can insert data at any location wherever he desires. In case of insert operation, we are treating as a part of update operation and we are relaying on update operation for insert operation.

III. CONCLUSION

We have analyzed the data security concerns in cloud data storage, which is a distributed storage system. We proposed a distributed scheme to ensure users that their data are indeed stored appropriately and kept intact all the time in the cloud. To provide redundancy we used erasure correcting code in the file distribution preparation. As we all know cloud is not just a third party data warehouse. So providing support for dynamic operations is very important. Our scheme maintains the same level of storage correctness assurance even if users modify, delete or append their data files in the cloud. Challenge response protocol along with pre-computed token is used to verify the storage correctness of user's data & to effectively locate the malfunctioning server when data corruption has been detected. Through detailed performance analysis, we show that our scheme is having very low communication overhead & guarantees to detect every single unauthorized data modification. Our scheme has no limitation on number of pre-computed tokens used for challenging the cloud servers. Unlimited number of challenges can be made. But we still believe that data storage security in Cloud computing is an area full of challenges and of paramount importance.

REFERENCES

[1] A. Juels, J. Burton, and S. Kaliski, "PORs: Proofs of Retrievability for Large Files," Proc. ACM CCS '07, Oct. 2007, pp. 584–97.

[2] G.Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable Data Possession at Untrusted Stores," Proc. ACM CCS '07, Oct. 2007, pp. 598–609.

[3] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik, "Scalable and Efficient Provable Data Possession," Proc. SecureComm '08, Sept. 2008.

[4] H. Shacham and B. Waters, "Compact Proofs of Retrievability," Proc. Asia-Crypt '08, LNCS, vol. 5350, Dec. 2008, pp. 90–107.

[5] K. D. Bowers, A. Juels, and A. Oprea, "Hail: A High-Availability and Integrity Layer for Cloud Storage," Proc. ACM CCS '09, Nov. 2009, pp. 187–98.

[6] C.Wang, Qian Wang, Kui Ren, Wenjing Lou, "Ensuring Data Storage Security in Cloud Computing," Proc. IWQoS '09, July 2009, pp. 1–9.

[7] Q. Wang, C.Wang, Wenjing Lou, Jin Li, "Enabling Public Verifiability and Data Dynamics for Storage Security in Cloud Computing," Proc. ESORICS '09, Sept. 2009, pp. 355–70.

[8] C. Erway, Alptekin, Charalampos Papamanthou, Roberto Tamassia, "Dynamic Provable Data Possession," Proc. ACM CCS '09, Nov. 2009, pp. 213–22.

[9] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "MR-PDP: Multiple-replica provable data possession," in Proc. of ICDCS'08. IEEE Computer Society, 2008, pp. 411–420.

[10] T. Schwarz and E. L. Miller, "Store, forget, and check: Using algebraic signatures to check remotely administered storage," in Proc. of ICDCS'06, 2006.

[11] N. Gohring, "Amazon's S3 down for several hours," Online at http://www.pcworld.com/businesscenter/article/142549/, amazons_s3_down_for_several hours.html, 2008.

[12] M. Arrington, "Gmail Disaster: Reports of Mass Email Deletions," Dec. 2006; http://www.techcrunch.com/2006/12/28/gmail-disaster-reports-of-massemail-deletions/

[13] Peter Mell, Tim Grance, "The NIST Definition of Cloud Computing", Online at http://www.nist.gov/itl/cloud/upload-def-v15.pdf.

[14] K. D. Bowers, A. Juels, and A. Oprea, "Proofs of Retrievability: Theory and Implementation," Cryptology ePrint Archive, Report 2008/175, 2008, http://eprint.iacr.org/.

[15] M. Lillibridge, S. Elnikety, A. Birrell, M. Burrows and M. Isard, "A Cooperative Internet Backup Scheme*", Proc. of the 2003, USENIX Annual Technical Conference (General Track)*, pp. 29–41, 2003.

[16] Qian Wang, Cong Wang, Kui Ren, Wenjing Lou, and Jin Li, "Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing" , IEEE Transactions on Parallel & Distributed Systems, Volume: 22, Issue: 5, pages: 847-859.

[17] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for storage security in cloud computing", in *Proc. of IEEE INFOCOM'10*, San Diego, CA, USA, March 2010.

[18] J. S. Plank and Y. Ding, "Note: Correction to the 1997 Tutorial on Reed-Solomon Coding," University of Tennessee, Tech. Rep. CS-03- 504, 2003.