ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 2, Issue 5, May 2015

Secure Process Data in Cloud Storage Using Data Integrity Protection

S.Muthukumari [1], D.Stanley[2], A.Ramesh Kumar [3]

Department Of Computer Applications, Francis Xavier Engineering College,Tirunelveli,TamilNadu, India[1]
Asst.Prof , Department Of Computer Applications, Francis Xavier Engineering College, Tirunelveli, TamilNadu, India [2]
Department Of Computer Applications, Francis Xavier Engineering College,Tirunelveli,TamilNadu, India [3]

Abstract: In this paper it is not an easy task to securely maintain all essential data where it has the need in many applications for clients in cloud. To maintain our data in cloud, it may not be fully trustworthy because client doesn't have copy of all stored data. But any authors don't tell us data integrity through its user and CSP level by comparison before and after the data update in cloud. So we have to establish new proposed system for this using our data reading protocol algorithm to check the integrity of data before and after the data insertion in cloud. Here the security of data before and after is checked by client with the help of CSP using our ―effective automatic data reading protocol from user as well as cloud level into the cloud with truthfulness. Also we have proposed the multi-server data comparison algorithm with the calculation of overall data in each update before its outsourced level for server restore access point for future data recovery from cloud data server. Our proposed scheme efficiently checks integrity in efficient manner so that data integrity as well as security can be maintained in all cases by considering drawbacks of existing methods.

Keywords: Cloud Storage, Data Integrity, Encryption, Security

I. Introduction

CLOUD storage offers an on-demand data outsourcing service model, and is gaining popularity due to its elasticity and low maintenance cost. However, security concerns arise when data storage is outsourced to third-party cloud storage providers. It is desirable to enable cloud clients to verify the integrity of their outsourced data, in case their data have been accidentally corrupted or maliciously compromised by insider/outsider attacks.

One major use of cloud storage is long-term archival, which represents a workload that is written once and rarely read. While the stored data are rarely read, it remains necessary to ensure its integrity for disaster recovery or compliance with legal requirements (e.g., [24]). Since it is typical to have a huge amount of archived data, whole-file checking becomes prohibitive. Proof of retrievability (POR) [16] and proof of data possession (PDP) [3] have thus been proposed to verify the integrity of a large file by spot- checking

It implements FMSR-DIP codes, and evaluate their overhead over the existing FMSR codes only a fraction of the file via various crypto- graphic primitives.

Suppose that we outsource storage to a server, which could be a storage site or a cloud-storage provider. If we detect corruptions in our outsourced data (e.g., when a server crashes or is compromised), then we should repair repair traffic saving over traditional erasure codes? A related approach is HAIL [4], which applies integrity protection for erasure codes. It constructs protection data on a per-file basis and distributes the protection data across different servers. To repair any lost data during a server failure, one needs to access the whole file, and this violates the design of regenerating codes. Thus, we need a different design of integrity protection tailored for regenerating codes

This paper design and implement a practical data integrity protection (DIP) scheme for regenerating-coding-based cloud storage. it augment the implementation of functional minimum-storage regenerating (FMSR) codes [15] and construct FMSR-DIP codes, which allow clients to remotely verify the integrity of random subsets of long-term archival data under a multiserver setting. FMSR-DIP codes preserve fault tolerance and repair traffic saving as in FMSR codes [15]. Also, we assume only a thin-cloud interface [23], meaning that servers only need to support standard read/ write functionalities. This adds to the portability of FMSR-DIP codes and allows simple deployment in general types of storage services. By combining integrity checking and efficient recovery, FMSR-DIP codes provide a low-cost solution for maintaining data availability in cloud storage.

It designs FMSR-DIP codes, which enable

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 2, Issue 5, May 2015

integrity protection, fault tolerance, and efficient recovery for cloud storage.

It exports several tunable parameters from FMSR-DIP codes, such that clients can make a trade-off between performance and security.

It conducts mathematical analysis on the security of FMSR-DIP codes for different parameter choices.

II. Literature Survey

H. Abu-Libdeh L.Princehouse and H.Weatherspoon et.al., proposed a Cloud storage providers expose simple interfaces to developers. Amazon S3's data model provides flat namespaces ("buckets") into which named objects can be uploaded for later retrieval. Other storage services can be mounted as network file systems. There is no widely agreed-upon standard interface, but S3's REST API has been adopted by smaller providers and by the open-source Eucalyptus server software. These interfaces differ, but are similar enough to be considered interchangeable. Storage providers are forced to compete on price rather than by offering unique services. Cloud storage is a highly competitive market. A change in pricing scheme or the emergence of new competition can render a particular provider unfavorably expensive compared to its alternatives. Clients may not be able to pick an optimal cloud storage provider because the switching cost overrides the desired benefits. Thus, clients experience vendor lock-in if their stored data is large. The fundamental problem is that clients have to make an all-or-none decision in switching their data to new providers

By striping data across multiple providers and adding appropriate redundancy, clients can tolerate outages and operational failures, as well as adapt to changes in the economic landscape.The goal of RACS is slightly different than RAID-5. Cloud storage is assumed to be much more reliable than hard disks, so data loss prevention is a much less compelling reason to use error correcting codes. RACS lowers the cost of switching providers, e.g., as a result of economic failure. Only1m of all data needs to be moved to leave a vendor. By reducing the impact of vendor lock-in, RACS increases the leverage of customers when negotiating contracts with cloud providers. RACS is implemented as an HTTP proxy with the same interface as Amazon S3.

M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia et.al Proposed a Cloud Computing, the long-held dream of computing as a utility, has the potential to transform a large part of the IT industry, making software even more attractive as a service and shaping the way IT hardware is designed and purchased. Developers with innovative ideas for new Internet services no longer require the large capital outlays in hardware to deploy their service or the human expense to operate it.

Our goal in this paper to reduce that confusion by clarifying terms, providing simple figures to quantify comparisons between of cloud and conventional Computing, and identifying the top technical and non-technical obstacles and opportunities of Cloud Computing. Applications Software needs to both scale down rapidly as well as scale up, which is a new requirement. Such software also needs a pay-for-use licensing model to match needs of Cloud Computing. Infrastructure Software needs to be aware that it is no longer running on bare metal but on VMs. Moreover, billing needs to built in from the start. Hardware Systems should be designed at the scale of a container (at least a dozen racks), which will be is the minimum purchase size. Cost of operation will match performance and cost of purchase in importance, rewarding energy proportionality such as by putting idle portions of the memory, disk, and network into low power mode. Processors should work well with VMs and flash memory should be added to the memory hierarchy, and LAN switches and WAN routers must improve in bandwidth and cost.

III. Theory

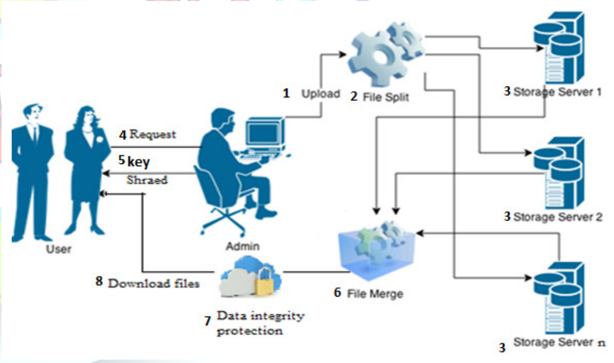


Figure 1

We adopt the adversarial model in [4] as our threat model. We assume that an adversary is mobile Byzantine, meaning that the adversary compromises a subset of servers in different time epochs (i.e., mobile) and exhibits arbitrary behaviors on the data stored in the compromised servers (i.e., Byzantine). To ensure meaningful file availability, we assume that the adversary can compromise and corrupt data in at most n k out of the n servers in any epoch, subject to the ðn; kÞ-MDS fault tolerance requirement. At the, end of each epoch, the client can ask for randomly chosen parts of remotely stored data and run a

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 2, Issue 5, May 2015

probabilistic checking protocol to verify the data integrity. Servers corrupted by the adversary may or may not correctly return data requested by the client. If corruption is detected, then the client may trigger the repair phase to repair corrupted data.

3.1 Working principle of FMSR Code

It first reviews FMSR codes in NCCloud [15], on which our DIP scheme is developed. FMSR codes belong to maximum distance separable (MDS) codes. An MDS code is defined by the parameters (n-k) where k < n. It encodes a file F of size lFl into n pieces of size lFl/k each. An (n,k)-MDS code states that the original file can be reconstructed from any k out of n pieces (i.e., the total size of data required is lFl). An extra feature of FMSR codes is that a specific piece can be reconstructed from data of size less than lFl. FMSR codes are built on regenerating codes [11], which minimize the repair traffic while preserving the MDS property.

We consider a distributed storage setting in which a file is striped over n servers using an (n,k)-FMSR code. Each server can be a storage site or even a cloud storage provider, and is assumed to be independent of other servers. An (n,k)-FMSR code splits a file of size lFl evenly into k(n-k) native chunks, and encodes them into n(n-k) code chunks where each native and code chunk has size lFl/k(n-k) . Each code chunk, denoted by Pi (where 1<=i<=n(n-k)), is constructed by a random linear combination of the native chunks, similar to the idea in [20]. The n(n-k) code chunks are stored in n servers (i.e., n -k code chunks per server), where the k(n-k) code chunks from any k servers can be decoded to reconstruct the original data. Decoding can be done by inverting the encoding matrix [19].

Suppose that one server fails. Our goal is to reconstruct the lost data of the failed server in a new server, so as to maintain the (n,k)-MDS fault tolerance. We define the repair traffic as the amount of data read from the other surviving servers, so as to reconstruct the lost data. We assume that there is a proxy (NCCloud in our case) that handles the entire Repair operation.

The conventional repair method for a single-server failure is to simply reconstruct the whole file by contacting any k surviving servers, so the repair traffic is lFl. Note that this repair method applies to all (n,k)-MDS codes. On the other hand, in FMSR codes, we first randomly pick a chunk from each of the (n-1) surviving servers, and then generate (n-k) random linear combinations of these (n-1) chunks to store in a new server. To guarantee that the MDS fault tolerance is preserved after multiple rounds of repair, NCCloud performs two-phase checking on the new code chunks generated in the Repair operation [15]. Fig. 1 illustrates the Repair operation for the (4, 2)-FMSR code, in which the repair traffic is reduced by 25 percent to 0:75jFj. It is shown that the repair traffic of FMSR codes can be further reduced to 50 percent for k=n-2 if n is large [15].

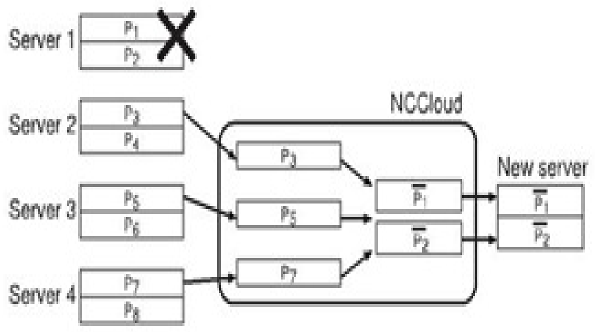


Figure 2

Example of how a file is repaired in (4,2)-FMSR codes. Each of the code chunks P1;...;P8 is a random linear combination of the native chunks. P1 and P2 are distinct random linear combinations of P3, P5, and P7.

We implement FMSR-DIP codes atop NCCloud [15]. In this section, we address how our implementation can fine-tune various design parameters to trade security for performance. Please refer to Section 5 of the supplementary file, available online, for additional implementation details on how we integrate FMSR-DIP codes into NCCloud and how we instantiate the cryptographic primitives.

FMSR-DIP codes operate in units of bytes. However, byte-level operations may make the implementation inefficient in practice, especially for large files. Here, we describe how FMSR-DIP codes can be extended to operate in units of blocks (i.e., a sequence of bytes) to trade security for performance. In the following, we describe the possible tunable parameters that are supported in FMSR-DIP codes.

PRP block size. Instead of permuting bytes, we can permute blocks of a tunable size (called the PRP block size). A larger PRP block size increases efficiency, but at the same time decreases security guarantees.

PRF block size. In a byte-level PRF operation, we can simply take the first byte of the AES-128 output as the PRF output. In fact, we can also compute a longer PRF and apply the PRF output to a block of bytes of a tunable size (called the PRF block size). To extend the PRF beyond the AES block size (16 bytes), we can pad the nonce with a chain of input blocks of 16 bytes each, and encrypt them using CBC mode. However, setting the PRF block size to

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 2, Issue 5, May 2015

larger than 16 bytes shows minimal performance improvement, as AES is invoked once for every 16 bytes of input in CBC mode and the total number of AES invocations remains the same for a larger PRF block size.

Check block size. Reading data from cloud storage is priced based on the number of GET requests. In the Check operation, downloading 1 byte per request will incur a huge monetary overhead. To reduce the number of GET requests, we can check a block of bytes of a tunable size (called the check block size). The checked blocks at the same offset of all code chunks will contain multiple rows of bytes. Although not necessary, it is recommended to set the check block size as a multiple of the PRF block size, so as to align with the PRF block operations.

AECC parameters. The AECC parameters (n',k') control the error tolerance within a code chunk and the domain size of the PRP being used in AECC. Given the same k', a larger n' implies better protection, but introduces a higher computational overhead.

Checking percentage. The checking percentage defines the percentage of a file to be checked in the Check operation. A larger implies more robust checking, at the expense of both higher monetary and performance over-heads with more data to download and check.

3.2 Cryptographic Primitives

Our DIP scheme is built on several cryptographic primi- tives, whose detailed descriptions can be found in [13], [14]. The primitives include:

1. symmetric encryption,
2. a family of pseudorandom functions (PRFs),
3 .a family of pseudorandom permutations (PRPs), and
4 .message authentication codes (MACs).

Each of the primitives takes a secret key. Intuitively, it means that it is computationally infeasible for an adversary to break the security of a primitive without knowing its corresponding secret key.

We also need a systematic adversarial error-correcting ;code (AECC) [5], [9] to protect against the corruption of a chunk. In conventional error-correcting codes (ECC), when a large file is encoded, it is first broken down into smaller stripes to which ECC is applied independently. AECC uses a family of PRPs as a building block to randomize the stripe structure so that it is computationally infeasible for an adversary to target and corrupt any particular stripe. Both FMSR codes and AECC provide fault tolerance. The difference is that we apply FMSR codes to a file striped across servers, while we apply AECC to a single code chunk stored within a server.

3.3 New Features of FMSR-DIP Implementation

In previous FMSR-DIP codes we have only a single link between the chunks it may not give integrity because if the single link available is lost then there is no way to get the information available in the chunk. To overcome this drawbacks it provides a XOR linked list in the XOR linked list to get the link from both previous and next chunk. If a single link is lost then we can get the information through another link. It provides a high integrity.

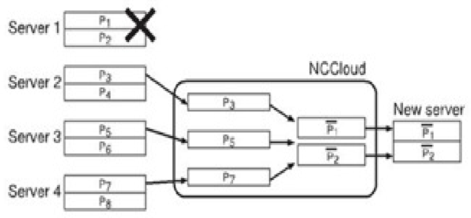


Figure 3

By having the double link through the each and every link the unauthorised person can get the file to avoid this we are using an RSA algorithm. The main purpose of this RSA algorithm is to enable a key generation by using key, the authorised person only can get the file this system gives a high integrity in the NC cloud.

IV. Simulation Result

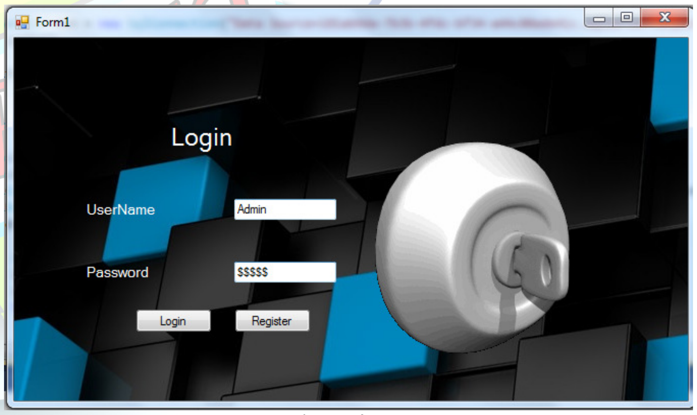
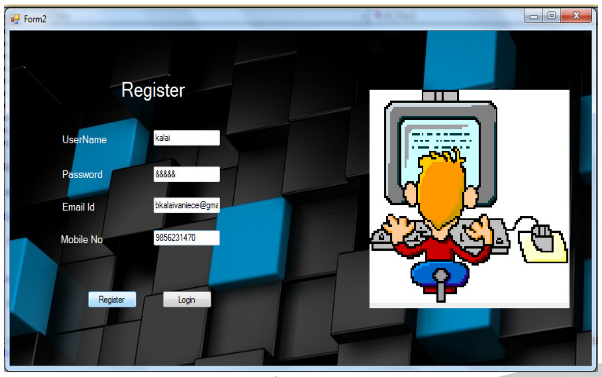


Figure 4

In this above figure show the user and admin login. Valid user only view the details of the files.



UserName	kalai
Password	&&&&&
Email Id	bkalaivaniece@gma
Mobile No	9856231470

Figure 5

In the above figure register to the new user. Then click the register button those who are only to view the files and then only the specified user to click the login button then go for next steps (i.e display the network details).

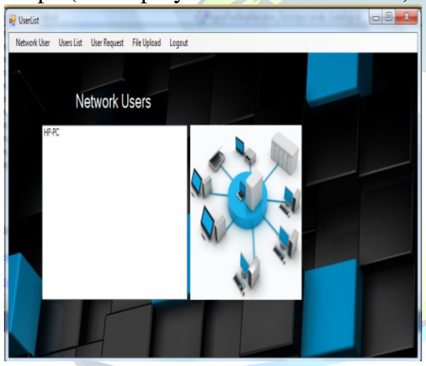
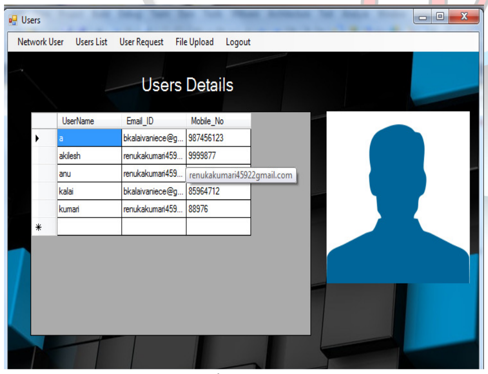


Figure 6

In the above figure show the network details. This show those who are connected to the same network.



	UserName	Email_ID	Mobile_No
▶	a	bkalaivaniece@g...	987456123
	akilesh	renukakumari459...	9999877
	anu	renukakumari459...	renukakumari45922gmail.com
	kalai	bkalaivaniece@g...	85964712
	kumari	renukakumari459...	88976
*			

Figure 7

In the above figure show the details of the registered users details only.

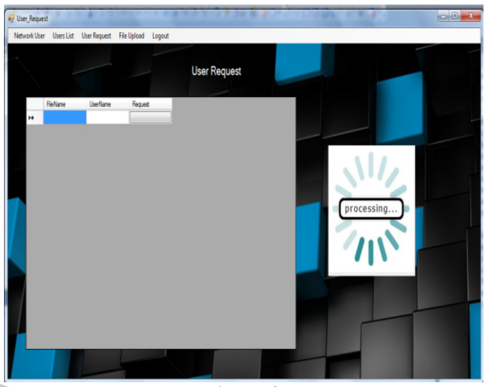
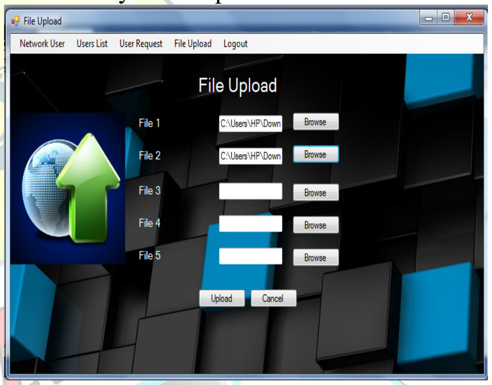


Figure 8

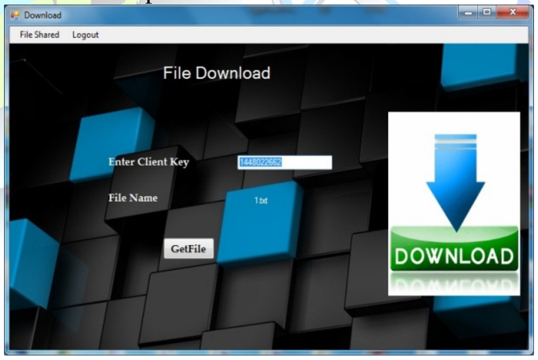
In the above figure show the user request. User request to the admin and accepted the request to user. Admin send the key to the specified valid user.



File 1	C:\Users\HP\Down
File 2	C:\Users\HP\Down
File 3	
File 4	
File 5	

Figure 9

This form is used to upload the file when admin uploaded file was splitted and stored in a different server.



Enter Client Key	1448022662
File Name	1.txt

Figure 10

The user view and download file using the valid key then only download the file otherwise neglet the file. Splitted file was merged and download the files.

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 2, Issue 5, May 2015

V. CONCLUSION

We design and implement a practical data integrity protection (DIP) scheme for regenerating-coding based cloud storage. We augment the implementation of functional minimum-storage regenerating (FMSR) codes and construct FMSR-DIP codes, which allow clients to remotely verify the integrity of random subsets of long-term archival data under a multiserver setting. FMSR-DIP codes preserve fault tolerance and repair traffic saving as in FMSR codes. Also, we assume only a thin-cloud interface, meaning that servers only need to support standard read/ write functionalities. This adds to the portability of FMSRDIP codes and allows simple deployment in general types of storage services. By combining integrity checking and efficient recovery, FMSR-DIP codes provide a low-cost solution for maintaining data availability in cloud storage. The problem of checking the integrity of static data, which is typical in long-term archival storage, The FMSR-DIP codes provide a low cost solution for maintaining data availability in cloud storage design FMSR-DIP codes, which enable integrity protection, fault tolerance, and efficient recovery for cloud storage. Conduct mathematical analysis on the security of FMSR-DIP codes for different parameter choices. It can implement FMSR-DIP codes and evaluate the running times of different basic operations, including Upload, Check, Download, and Repair, for different parameter choices. In this we are using security process in the form of code based derived key from the master key in the before system there is no security in the form encryption and decryption process. In this it uses dynamic public cloud and XOR Linked list to the recovery processes.

VI. FUTURE ENHANCEMENT

Using MD5 file encrypting with higher security and priority.Merge and split file using correlation game theory.It is used for splitting the files in speed manner.

References

[1]. H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon, "RACS: A Case for Cloud Storage Diversity," Proc. First ACM Symp. Cloud Computing (SoCC '10), 2010

[2]. .M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. [26] "TechCrunch," Online Backup Company Carbonite Loses Customers' Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of Cloud Computing," Comm. ACM, vol. 53, no. 4, pp 50-58, 2010.

[3]. G. Ateniese, R. Burns, R. Curtmola, J. Herring, O. Khan, L. [27] M. Vrable, S. Savage, and G. Voelker, "Cumulus: Filesystem Kissner, Z. Peterson, and D. Song, "Remote Data Checking Using Provable Data Possession," ACM Trans. Information and System Security, vol. 14, article 12, May 2011.

[4]. K. Bowers, A. Juels, and A. Oprea, "HAIL: A High-Availability and Integrity Layer for Cloud Storage," Proc. 16th ACM Conf. Computer and Comm. Security (CCS '09), 2009.

[5]. K. Bowers, A. Juels, and A. Oprea, "Proofs of Retrievability: Theory and Implementation," Proc. ACM Workshop Cloud Comput- ing Security (CCSW '09), 2009

[6]. B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote Data Checking for Network Coding-Based Distributed Storage Sys- tems," Proc. ACM Workshop Cloud Computing Security (CCSW '10),2010.

[7]. H.C.H. Chen and P.P.C. Lee, "Enabling Data Integrity Protection in Regenerating-Coding-Based Cloud Storage," Proc. IEEE 31st Symp. Reliable Distributed Systems (SRDS '12), 2012.

[8]. L. Chen, "NIST Special Publication 800-108," Recommendation for Key Derivation Using Pseudorandom Functions (Revised), http:// csrc.nist.gov/publications/nistpubs/800-108/sp800-108.pdf, Oct. 2009.

[9]. R. Curtmola, O. Khan, and R. Burns, "Robust Remote Data Checking," Proc. ACM Fourth Int'l Workshop Storage Security and Survivability (StorageSS '08), 2008.

[10]. R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "MR-PDP: Multiple-Replica Provable Data Possession," Proc. IEEE 28th Int'l Conf. Distributed Computing Systems (ICDCS '08), 2008.

[11]. A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, "Network Coding for Distributed Storage Sys- tems," IEEE Trans. Information Theory, vol. 56, no. 9, 4539-4551, Sept. 2010.

[12]. D. Ford, F. Labelle, F.I. Popovici, M. Stokel, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in Globally Distributed Storage Systems," Proc. Ninth USENIX Symp. Operating Systems Design and Implementation (OSDI '10), Oct. 2010.

[13]. O. Goldreich, Foundations of Cryptography: Basic Tools. Cambridge Univ. Press, 2001.

[14]. O. Goldreich, Foundations of Cryptography: Basic Applications.Cambridge Univ. Press, 2004.

[15]. Y. Hu, H. Chen, P. Lee, and Y. Tang, "NCCloud: Applying Network Coding for the Storage Repair in a Cloud-of-Clouds,"Proc. 10th USENIX Conf. File and Storage Technologies (FAST '12),2012.

[16]. A. Juels and B. Kaliski Jr., "PORs: Proofs of Retrievability for Large Files," Proc. 14th ACM Conf. Computer and Comm. Security (CCS '07), 2007.

[17]. H. Krawczyk, "Cryptographic Extraction and Key Derivation: The HKDF Scheme," Proc. 30th Ann. Conf. Advances in Cryptology (CRYPTO '10), 2010.

[18]. E. Naone, "Are We Safeguarding Social Data?" http:// www.technologyreview.com/blog/editors/22924/, Feb. 2009.

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 2, Issue 5, May 2015

[19]. J.S. Plank, "A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-Like Systems," Software - Practice & Experience, vol. 27, no. 9, pp. 995-1012, Sept. 1997.

[20]. M.O. Rabin, "Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance," J. ACM, vol. 36, no. 2, pp. 335- 348, Apr. 1989.

[21]. I. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," J. Soc. Industrial and Applied Math., vol. 8, no. 2, pp. 300- 304, 1960.

[22]. B. Schroeder, S. Damouras, and P. Gill, "Understanding Latent Sector Errors and How to Protect against Them," Proc. USENIX Conf. File and Storage Technologies (FAST '10), Feb. 2010.

[23]. B. Schroeder and G.A. Gibson, "Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You?" Proc. Fifth USENIX Conf. File and Storage Technologies (FAST '07), Feb. 2007.

[24]. T. Schwarz and E. Miller, "Store, Forget, and Check: Using Algebraic Signatures to Check Remotely Administered Storage," Proc. IEEE 26th Int'l Conf. Distributed Computing Systems, (ICDCS '06), 2006.

[25]. H. Shacham and B. Waters, "Compact Proofs of Retrievability,"Proc. 14th Int'l Conf. Theory and Application of Cryptology and Information Security: Advances in Cryptology (ASIACRYPT '08),2008.