CLEAR IJRMST *Volume-04* *Issue-08* *Jul-Dec 2014*
Online-ISSN: 2249 -3506
Print-ISSN: 2249 -3492
www.thaavan.org

Semi Trusted Proxy To Secure Public Auditing With Network Coding Based Storage In Multi Storage Cloud Architecture

K.Gnanathangavelu[1], S.Pradeep[2]
[1]Assistant Professor, Department of Information Technology, Arignar Anna Institute of Science and Technology, Chennai, Tamil Nadu, India. kgvelumtech@hotmail.com
[2]Assistant Professor, Department of Information Technology,Pannai College of Engineering and Technology , Sivagangai, Tamil Nadu, India. csspradeep@gmail.com

Abstract

Cloud computing setting in which probabilistic querying of outsourced data is a service provider. The data is to be revealed only to trusted users, not to the service provider or anyone else. Outsourcing offers the data owner scalability and a low initial investment. The need for privacy may be due to the data being sensitive or otherwise confidential. Security challenges and Data Loses are still among the biggest problem when considering the adoption of cloud services. This triggered a lot of research activities, resulting in a quantity of proposals targeting the various cloud security threats and repair Storage node but doesn't provide efficient security mechanism and fault isolation process. Now this proposed system will overcome for all existing problems. New public auditing scheme for regenerating-code-based cloud storage, to solve the regeneration problem of failed authenticators in the absence of data owners; to introduce a proxy, which is privileged to regenerate the authenticators, into the traditional public auditing system model. Moreover, to design a novel public verifiable authenticator is generated by a couple of keys and can be regenerated using partial keys. Thus, our scheme can completely release data owners from online burden. In addition, to randomize the encode coefficients with a pseudorandom function to preserve data privacy. Our scheme is highly efficient and can be feasibly integrated into the regenerating-code-based cloud storage.

Keywords—Cloud Stroage, Regenerating Codes, Public Audit, Proxy.

I. INTRODUCTION

Cloud computing is the delivery of computing as a service rather than a product, whereby shared resources, software, and information are provided to computers and other devices as a utility over a network .Cloud computing is Internet-based computing, whereby shared servers provide resources, software, and data to computers and other devices on demand, as with the electricity grid. The term "cloud", to have its origins in network diagrams that represented the internet, or various parts of it, as schematic clouds. Cloud computing refers to the many different types of services and applications being delivered in the internet cloud, and the fact that, in many cases, the devices used to access these services and applications do not require any special applications.**Cloud computing is a technology** that uses the internet and central remote servers to maintain data and applications. Cloud computing allows consumers and businesses to use applications without installation and access their personal files at any computer with internet access. This technology allows for much more inefficient computing by centralizing storage, memory, processing and bandwidth.A simple example of cloud computing is Yahoo email, Gmail, or Hotmail etc. You don't need software or a server to use them. All a consumer would need is just an internet connection and you can start sending emails. The server and email management software is all on the cloud (internet) and is totally managed by the cloud service provider Yahoo, Google etc. The consumer gets to use the software alone and enjoy the benefits. The analogy is, 'If you need milk, would you buy a cow?' All the users or consumers need is to get the benefits of using the software or hardware of the computer like

sending emails etc. Just to get this benefit (milk) why should a consumer buy a (cow) software /hardware.

Cloud computer works by hosting your information on computers 'out there' 'in the cloud'. The cloud is basically a batch of computers called data centers or servers that hold your information (files, images, videos, etc) and can be located anywhere. You're working in a cloud because you don't have to store software or files on your own computer.To understand it works, it is important to think of cloud computing in two levels. The front level is the user level which is what you use such as your Google Email or Face book or the webpage where you are downloading the latest version of Flash. The other level is the backend which is all of the hardware and the software architecture. It's here that the actual Gmail or Face book is held waiting to be accessed by you.Because all of the different servers are running together in cloud computing, one application can have the computer power of many servers. This allows something like Face book to run. Imagine trying to host Face book on your laptop. It just wouldn't work.Therefore, the cloud brings together a large number of computers to operate a single application.

II. REVIEW OF LITERATURE

Henry C. H. Chen and Patrick P. C. Lee To protect outsourced data in cloud storage against corruptions, enabling integrity protection, fault tolerance, and efficient recovery for cloud storage becomes critical. Regenerating codes provide fault tolerance by striping data across multiple servers, while using less repair traffic than traditional erasure codes during failure recovery. Therefore, we study the problem of remotely checking the integrity of regenerating-coded data against corruptions under a real-life cloud storage setting. We design and implement a practical data integrity protection (DIP) scheme for a specific regenerating code, while preserving the intrinsic properties of fault tolerance and repair traffic saving. Our DIP scheme is designed under a Byzantine adversarial model, and enables a client to feasibly verify the integrity of random subsets of outsourced data against general or malicious corruptions. It works under the simple assumption of thin-cloud storage and allows different parameters to be fine-tuned for the performance-security trade-off. We implement and evaluate the overhead of our DIP scheme in a real cloud storage testbed under different parameter choices. We demonstrate that remote integrity checking can be feasibly integrated into regenerating codes in practical deployment.

Yuchong Hu†, Henry C. H. Chen†, Patrick P. C. Lee†, Yang TangTo provide fault tolerance for cloud storage, recent studies propose to stripe data across multiple cloud vendors. However, if a cloud suffers from a permanent failure and loses all its data, then we need to repair the lost data from other surviving clouds to preserve data redundancy. We present a proxy-based system for multiple-cloud storage called NCCloud, which aims to achieve cost-effective repair for a permanent single-cloud failure. NCCloud is built on top of network-coding-based storage schemes called regenerating codes. Specifically, we propose an implementable design for the functional minimumstorage regenerating code (F-MSR), which maintains the same data redundancy level and same storage requirement as in traditional erasure codes (e.g., RAID-6), but uses less repair traffic. We implement a proof-of-concept prototype of NCCloud and deploy it atop local and commercial clouds. We validate the cost effectiveness of FMSR in storage repair over RAID-6, and show that both schemes have comparable response time performance in normal cloud storage operations.

Cloud storage provides an on-demand remote backup solution. However, using a single cloud storage vendor raises concerns such as having a single point of failure and vendor lock-ins. As suggested, a plausible solution is to stripe data across different cloud vendors. While striping data with conventional erasure codes performs well when some clouds experience *short-term* failures or *foreseeable* permanent failures, there are real-life cases showing that permanent failures do occur and are not always foreseeable. This work focuses on *unexpected* cloud failures. When a cloud fails permanently, it is important to activate *storage repair* to maintain the level of data redundancy. A repair operation reads data from existing surviving clouds and reconstructs the lost data in a new cloud. It is desirable to reduce the repair traffic,

and hence the monetary cost, due to data migration.

G. Ateniese et al Introduce a model for provabledata possession (PDP) that allows a client that has stored data at an untrusted server to verify that the server possesses the original data without retrieving it. The model generates probabilistic proofs of possession by sampling random sets of blocks from the server, which drastically reduces I/O costs. The client maintains a constant amount of metadata to verify the proof. The challenge/response protocol transmits a small, constant amount of data, which minimizes network communication. Thus, the PDP model for remote data checking supports large data sets in widely-distributed storage system.We present two provably-secure PDP schemes that are more efficient than previous solutions, even when compared with schemes that achieve weaker guarantees. In particular, the overhead at the server is low (or even constant), as opposed to linear in the size of the data. Experiments using our implementation verify the practicality of PDP and reveal that the performance of PDP is bounded by disk I/O and not by cryptographic computation.

A. Juels and B. S. Kaliski, Jr Cloud computing promises In this paper, we define and explore *proofs of retrievability*(PORs). A POR scheme enables an archive or back-up service (prover) to produce a concise proof that a user (verifier) can retrieve a target file F, that is, that the archive retains and reliably transmits file data sufficient for the user to recover F in its entirety.A POR may be viewed as a kind of cryptographic proof of knowledge (POK), but one specially designed to handle a *large* file (or bitstring) F. We explore POR protocols here in which the communication costs, number of memory accesses for the prover, and storage requirements of the user (verifier) are small parameters essentially independent of the length of F. In addition to proposing new, practical POR constructions, we explore implementation considerations and optimizations that bear on previously explored, related schemes. In a POR, unlike a POK, neither the prover nor the verifier need actually have knowledge of F. PORs give rise to a new and unusual security definition whose formulation is another contribution of our work. We view PORs as an important tool for semi-trusted online archives. Existing cryptographic techniques help users ensure the privacy and integrity of files they retrieve. It is also natural, however, for users to want to verify that archives do not delete or modify files prior to retrieval. The goal of a POR is to accomplish these checks *without users having to download the files themselves*. A POR can also provide quality-of-service guarantees, i.e., show that a file is retrievable within a certaintimebound.

C.Wang, Q. Wang, K. Ren, N. Cao, and W. Lou**,** Distributed data storage has gained increasing popularity for efficient and robust data management in wireless sensor networks (WSNs). But the distributed architecture also makes it challenging to build a highly secure and dependable yet lightweight data storage system. On the one hand, sensor data are subject to not only Byzantine failures, but also dynamic pollution attacks, as along the time the adversary may modify/pollute the stored data by compromising individual sensors. On the other hand, the resource constrained nature of WSNs precludes the applicability of heavyweight security designs.

III. RELATED WORKS

The first requirement from our storage code is the (n, k) property: a code will be storing information in n storage nodes and should be able to tolerate any combination of n - k failures without data loss. We refer to codes that have this reliability as "(n, k) erasure codes," or codes that have "the (n, k) property." One well-known class of erasure codes that have this property is the family of maximum distance separable (MDS) codes. In short, an MDS code is a way to take a data object of size M, split it into chunks of size M/k and create n chunks of the same size that have the (n, k) property. It can be seen that MDS codes achieve the (n, k) property with the minimum storage overhead possible: any k storage nodes jointly store M bits of useful information, which is the minimum possible to guarantee recovery

Our second requirement is efficient exact repair. When one node fails or becomes unavailable, the stored information should be easily reconstructable using other surviving nodes. Simple regenerating codes achieve the (n, k) property and simple repair simultaneously by separating the two problems. Large MDS codes are used to provide reliability against any n-k failures while very simple XORs applied over the MDS coded packets provide efficient exact repair when single node failures happen. File reconstruction of a (4,2,2)-SRC. We give an example of a single node repair of the (4; 2; 2)-SRC. We assume that node 1 is lost and a newcomer joins the system. To reconstruct x1, the newcomer has to download y1 and s1 from nodes 3 and 4. This simple repair scheme is possible due to the way that we placed the chunks in the 4 storage nodes: each node stores 3 chunks with different index. The newcomers reconstructs each lost chunk by downloading, accessing, and XORing2 other chunks. In this process the outer MDS codes are not used. The Repair Node in a(4,2,2)-SRC.

Network coding is a technique which can be used to improve a network's throughput, efficiency and scalability, as well as resilience to attacks and eavesdropping. Instead of simply relaying the packets of information they receive, the nodes of a network take *several* packets and combine them together for transmission. This can be used to attain the maximum possible information flow in a network.

It has been proven that linear coding is enough to achieve the upper bound in multicast problems with one or more sources.[1] However linear coding is not sufficient in general (e.g. multisource, multisink with arbitrary demands), even for more general versions of linearity such as convolutional coding and filter-bank coding. Finding optimal coding solutions for general network problems with arbitrary demands remains an open problem.

In a network coding problem, a group of nodes P are involved in moving the data from S source nodes to K sink nodes. Each node generates new packets which are linear combinations of earlier received packets, multiplying them by coefficients chosen from a finite field, typically of size $GF(2^s)$.Each node, p_k with in degree, $InDeg(p_k) = S$, generates a message X_k from the linear combination of received messages $\{M_i\}_{i=1}^{S}$ by the relation:

$$X_k = \sum_{i=1}^{S} g_k^i \cdot M_i$$

Where the values g_k^i are the coefficients selected from $GF(2^s)$. Note that, since operations are computed in a finite field, the generated message is of the same length as the original messages. Each node forwards the computed value X_k along with the coefficients, g_k^i, used in the k^{th} level, g_k^i.

Sink nodes receive these network coded messages, and collect them in a matrix. The original messages can be recovered by performing Gaussian elimination on the matrix. In reduced row echelon form, decoded packets correspond to the rows of the form $e_i = [0...010...0]$.

Fault tolerance is the property that enables a system to continue operating properly in the event of the failure of (or one or more faults within) some of its components. If its operating quality decreases at all, the decrease is proportional to the severity of the failure, as compared to a naïvely designed system in which even a small failure can cause total breakdown. Fault tolerance is particularly sought after in high-availability or life-critical systems.

A fault-tolerant design enables a system to continue its intended operation, possibly at a reduced level, rather than failing completely, when some part of the system fails. The term is most commonly used to describe computer systems designed to continue more or less fully

operational with, perhaps, a reduction in throughput or an increase in response time in the event of some partial failure. That is, the system as a whole is not stopped due to problems either in the hardware or the software. An example in another field is a motor vehicle designed so it will continue to be drivable if one of the tires is punctured.

Recovery from errors in fault-tolerant systems can be characterized as either roll-forward or roll-back. When the system detects that it has made an error, roll-forward recovery takes the system state at that time and corrects it, to be able to move forward. Roll-back recovery reverts the system state back to some earlier, correct version, for example using check pointing, and moves forward from there. Roll-back recovery requires that the operations between the checkpoint and the detected erroneous state can be made idempotent. Some systems make use of both roll-forward and roll-back recovery for different errors or different parts of one error.

Within the scope of an *individual* system, fault tolerance can be achieved by anticipating exceptional conditions and building the system to cope with them, and, in general, aiming for self-stabilization so that the system converges towards an error-free state. However, if the consequences of a system failure are catastrophic, or the cost of making it sufficiently reliable is very high, a better solution may be to use some form of duplication. In any case, if the consequence of a system failure is so catastrophic, the system must be able to use reversion to fall back to a safe mode. This is similar to roll-back recovery but can be a human action if humans are present in the loop

IV. PROPOSED WORK

Focus on the integrity verification problem in regenerating-code-based cloud storage, especially with the functional repair strategy.The overhead of using cloud storage should be minimized as much as possible such that a user does not need to perform too many operations to their outsourced data (in additional to retrieving it).To fully ensure the data integrity and save the users' computation resources as well as online burden, propose a public auditing scheme for the regenerating-code-based cloud storage, in which the integrity checking and regeneration (of failed data blocks and authenticators) are implemented by a third-party auditor and a semi-trusted proxy separately on behalf of the data owner.*Fully Encrypt* the coefficients to protect data privacy against the auditor, which is more lightweight than applying the proof blind technique in [14] and [15] and data blind method.This framework design a homomorphic authenticator based on BLS signature, which can be generated by a couple of secret keys and verified publicly.The coefficients are masked by a PRF (Pseudorandom Function) during the Setup phase to avoid leakage of the original data. This method is lightweight and does not introduce any computational overhead to the cloud servers or TPA, Advantage of Improve the flexibility and efficiency of our auditing scheme; The storage overhead of servers, the computational overhead of the data owner and communication overhead during the audit phase can be effectively reduced. The details for implementing Co-Efficient Vector codes in multiple storage node, to specify three operations for Regeneration codes on a particular file object: 1) File upload, 2) File download, and 3) Repair. Each node repository is viewed as a logical storage node. Our implementation assumes a public cloud storage interface, such that the storage nodes (i.e., cloud repositories) only need to support basic read/write operations.Regeneration code generate a Co-Efficient encoding process this entity only supported in proxy based approach, some time proxy get one time failure and human attack these time doesn't retrieve any data to destroyed cloud. Another thought to lost above implementation like data store in multi cloud scenario for different logic and data separation process because of data security reason.

Windows Azure is Microsoft's application platform for the public cloud. You can use this platform in many different ways. For instance, you can use Windows Azure to build a web application that runs and stores its data in Windows Azure data centers. You can use Windows Azure just to store data, with the applications that use this data running on-

premises .You can use Windows Azure to create virtual machines for development and test or to run SharePoint and other applications. You can use Windows Azure to build massively scalable applications with lots and lots of users. Because the platform offers a wide range of services, all of these things-and more-are possible.To do any of them, though, you need to understand the basics. Even if you don't know anything about cloud computing, this article will walk you through the fundamentals of Windows Azure. The goal is to give you a foundation for understanding and using this cloud platform.

One of the most common things that people do in the cloud is run web sites and web applications. Windows Azure Virtual Machines allows this, but it still leaves you with the responsibility of administering one or more VMs. What if you just want a web site where somebody else takes care of the administrative work for you?This is exactly what Windows Azure Web Sites provides. This execution model offers a managed web environment using the Windows Azure Management portal as well as APIs. You can move an existing web site into Windows Azure Web Sites unchanged, or you can create a new one directly in the cloud. Once a web site is running, you can add or remove instances dynamically, relying on Windows Azure Web Sites to load balance requests across them. The standard option also lets you increase the size (computing power) of your instances if needed.Windows Azure Web Sites is intended to be useful for corporations, developers, and web design agencies. For corporations, it's an easy-to-manage, scalable, highly secure, and highly available solution for running presence web sites. For development, it supports .NET, PHP, Node.js, and Python along with SQL Database and MySQL (from ClearDB, a Microsoft partner) for relational storage. It also provides built-in support for several popular applications, including WordPress, Joomla, and Drupal. The goal is to provide a low-cost, scalable, and broadly useful platform for creating web sites and web applications in the public cloud.

1) Architecture Diagram

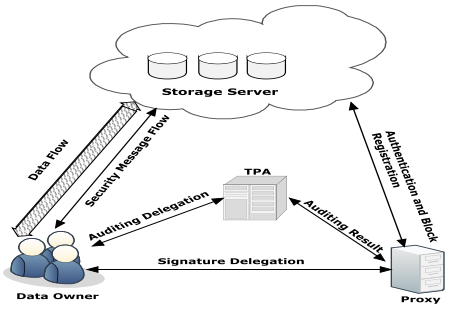


Figure 1 Architecture diagram

2)System Modules

The system comprises of the following modules which include,

1. Setup Creation Module
2. File Upload Module
3.Public Auditing Module
4.Storage Node Re-Generate Module

.

1.Setup Creation Module

The Setup creation module fully worked on data owner side and maintains this procedure to initialize the auditing scheme.

KeyGen(1κ)→ *(pk, sk):* This polynomial-time algorithm is run by the data owner to initialize its public and secret parameters by taking a security parameter *κ* as input.
Degelation(sk) → (x): This algorithm represents the interaction between the data owner and proxy. The data owner delivers partial secret key *x* to the proxy through a secure approach.
SigAndBlockGen(sk, F) → (ϕ,ψ, t): This polynomial time algorithm is run by the data owner and takes the secret parameter *sk*and the original file *F* as input, and then outputs a coded block set , an authenticator set and a file tag *t*.

2. File Upload Module

The file F is split into *m* blocks, and the original *m s*-dimensional vectors each original block *wi*is appended with the vector of length *m* containing

a single '1' in the *i*th position and is otherwise zero.
Then, the augmented vectors are encoded into coded blocks. Specifically, they are linearly combined and generate coded blocks with randomly chosen coefficients vector.
Original encoded files upload into several storage node and coefficient vector has been stored into proxy server. The data server doesn't response that time will check vector and regenerate the missing data into new storage.
.

3.Public Auditing

The cloud servers and TPA interact with one another to take a random sample on the blocks and check the data intactness in this procedure.
Challenge(Fin f o) → (C): This algorithm is performed by the TPA with the information of the file *Fin f o* as input and a challenge *C* as output.
Proof Gen(C,ϕ,ψ)→(P): This algorithm is run by each cloud server with input challenge *C*, coded block set and authenticator set , then it outputs a proof *P*.
*Veri f y(P, pk, C)→ (*0, 1*):* This algorithm is run by TPA immediately after a proof is received. Taking the proof *P*, public parameter *pk*and the corresponding challenge *C* as input, it outputs 1 if the verification passed and 0 otherwise.

4.Storage Node Re-Generate Module

In the absence of the data owner, the proxy interacts with the cloud servers during this procedure to repair the wrong server detected by the auditing process.

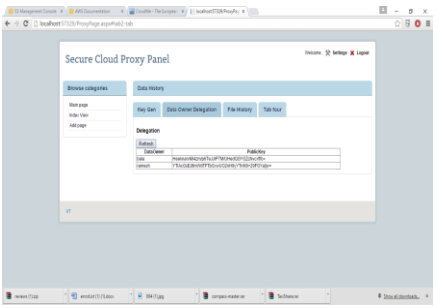
ClaimForRep(Fin f o) → (Cr): This algorithm is similar with the *Challenge()* algorithm in the Audit phase, but outputs a claim for repair *Cr* .

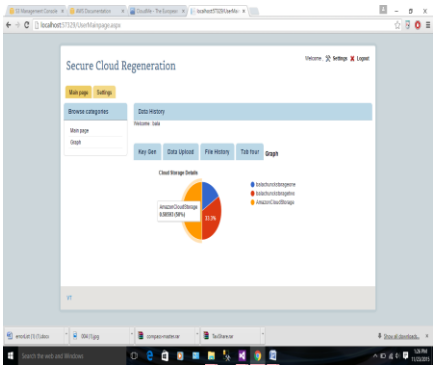
GenForRep(Cr,ϕ,ψ) → (BA): The cloud servers run this algorithm upon receiving the *Cr* and finally output the block and authenticators set *BA* with another two inputs ,.

BlockAndSigReGen(Cr , BA) → (ϕ',ψ',⊥*):* The proxy implements this algorithm with the claim *Cr* and responses *BA* from each server as input, and outputs a new coded block set $\psi'$and authenticator set ϕ'if successful, outputting ⊥if otherwise.

V. CONCLUSION

A public auditing scheme for the regenerating-code-based cloud storage system, where the data owners are privileged to delegate TPA for their data validity checking. To protect the original data privacy against the TPA, we randomize the coefficients in the beginning rather than applying the blind technique during the auditing process. Considering that the data owner cannot always stay online in practise, in order to keep the storage available and verifiable after a malicious corruption, we introduce a semi-trusted proxy into the system model and provide a privilege for the proxy to handle the reparation of the coded blocks and authenticators. To better appropriate for the regenerating-code-scenario, we design our authenticator based on the BLSsignature





VI FUTUREENHANCEMENT

To collect information from any remote location in the absence of network connectivity and to recover the files in case of the file deletion or if the cloud gets destroyed due to any reason.Client creates the file in cloud first time, it is stored at the main cloud. When it is stored in main server, the main file of client is being EXORed with the Seed Block of the particular client. And that EXORed file is stored at the remote server in the form of file'. If either unfortunately file in main cloud crashed / damaged or file is been deleted mistakenly, then the user will get the original file by EXORing file' with the seed block of the corresponding client to produce the original file and return the resulted file i.e. original file back to the requested client.

REFERENCES

[1] M. Armbrust*et al.*, "Above the clouds: A Berkeley view of cloud computing," Dept. Elect. Eng. Comput.Sci., Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2009-28, 2009.
[2] G. Atenieses*et al.*, "Provable data possession at untrusted stores," in *Proc. 14th ACM Conf. Comput. Commun.Secur.(CCS)*, New York, NY, USA, 2007, pp. 598–609.
[3] A. Juels and B. S. Kaliski, Jr., "PORs: Proofs of retrievability for large files," in *Proc. 14th ACM Conf. Comput. Commun.Secur.*, 2007, pp. 584–597.
[4] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "MR-PDP: Multiple-replica provable data possession," in *Proc. 28th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2008, pp. 411–420.
[5] K. D. Bowers, A. Juels, and A. Oprea, "HAIL: A high-availability and integrity layer for cloud storage," in *Proc. 16th ACM Conf. Comput. Commun.Secur.*, 2009, pp. 187–198.
[6] J. He, Y. Zhang, G. Huang, Y. Shi, and J. Cao, "Distributed data possession checking for securing multiple replicas in geographicallydispersed clouds," *J. Comput. Syst. Sci.*, vol. 78, no. 5, pp. 1345–1358, 2012.
[7] B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote data checking for network coding-based distributed storage systems," in *Proc. ACM Workshop Cloud Comput. Secur. Workshop*, 2010, pp. 31–42.
[8] H. C. H. Chen and P. P. C. Lee, "Enabling data integrity protection in regenerating-coding-based cloud storage: Theory and implementation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 2, pp. 407–416, Feb. 2014.
[9] 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.
[10] 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.