ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com

International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE) Vol. 2,Special Issue 6, March 2016

EFFICIENT APPROACH FOR DATA RETRIEVABILITY ON CLOUD STORAGE SYSTEM

MAHALAKSHMI.S[1],PAVITHRA.R[2],SELVARANI.R[3],THILAGAM.J[4]

[1,2,3,4] DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING,MAGNA COLLEGE OF ENGINEERING

ABSTRACT-Cloud storage is a model of data storage in which the digital data is stored in logical pools. It allows users to store their data in a remote server to get rid of expensive local storage and management costs and then access data of interest anytime anywhere. We propose an enhanced dynamic proof of retrievability scheme supporting public audit ability and communication-efficient recovery from data corruptions. We split up the data into small data blocks and encode that data block using network coding. To eliminate the communication overhead for small data corruptions within a server, each encoded data block is further encoded. Based on the encoded data blocks, we utilize tree structure to enforce the data sequence for dynamic operations, preventing the cloud service provider from manipulating data block to pass the integrity check in the dynamic scenario. We also effective analyze with our encrypted Base64 method for the effectiveness of the proposed construction in defending against pollution attacks during data retrievability.

I.INTRODUCTION

Cloud Computing," to put it simply, means "Internet Computing." The Internet is commonly visualized as clouds; hence the term "cloud computing" for computation done through the Internet. With Cloud Computing users can access database resources via the Internet from anywhere, for as long as they need, without worrying about any maintenance or management of actual resources. Besides, databases in cloud are very dynamic and scalable. Cloud computing is unlike grid computing, utility computing, or autonomic computing. In fact, it is a very independent platform in terms of computing. The best example of cloud computing is Google Apps where any application can be accessed using a browser and it can be deployed on thousands of computer through the Internet.A cloud is just a combination of hardware (computer, other devices), networks, storage, services and interfaces that helps in delivering computing as a service. It has mainly three users.

II.LITERATURE SURVEY

Remote data integrity checks for public cloud storage have been investigated in various systems and security models [6]–[10]. Considering the large size of the outsourced data and the owner's constrained resource capability, the cost to audit data integrity in the cloud environment could be formidable and expensive to the data owner.]. [6] was the first to introduce the "Provable Data Possession (PDP)" model and proposed an integrity verification scheme for static data using RSAbased homomorphic authenticators. At the same time, Juels et al. [8] proposed the "Proof of Retrievability (PoR)" model which is stronger than the PDP model in the sense that the system additionally guarantees the retrievability of outsourced data. Specifically, the authors proposed a spot-checking approach to guarantee possession of data files and employed error-correcting coding technologies to ensure the retrievability. A limitation of their scheme is that the number of challenges is constrained. Shacham et al. [10] utilized the homomorphic signatures to design an improved PoR scheme. Although the scheme supported public auditability of static data using publicly verifiable homomorphic authenticators, how to perform data recovery was not explicitly discussed. To achieve strong data retrievability, Bowers et al. [9] developed a PDP scheme with full data dynamics using skip list. Meantime, Wang et al. [7] proposed a scheme supporting public auditability and data dynamics using BLS based signatures and Merkle hash tree (MHT). Zhu et al.

2.1. PROBLEMS IN EXISTING SYSTEM

In existing system, while uploading, the entire data were uploaded as single block, so we couldn't find the particular data loss.Do not support efficient data dynamics and/or suffer from security vulnerabilities when involving dynamic data operations. Here they haven't used any network codes or erasure codes hence they faced many difficulties while finding the redundancies.No file audit report and file audit delegation. Data corruption caused by server hacks or Byzantine failures. Get network overload on every servers.Security issues such as data integrity and availability are the main obstacles in this system.

III.PROPOSED SYSTEM

We propose an enhanced dynamic proof of retrievability scheme supporting public audit ability and communication-efficient recovery from data

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com

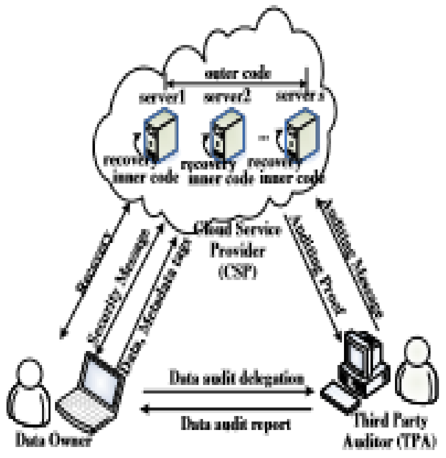
International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE) Vol. 2,Special Issue 6, March 2016

corruptions. To this end, we split up the data into small data blocks and encode each data block individually using network coding. Network coding and erasure codes are adopted to encode data blocks to achieve within server and cross server data redundancy, tolerating data corruption. By combing range based on encrypted Base64 method and improved version of aggregately signature based broadcast encryption, our construction can support efficient data dynamics while defending against data replay attack.

3.1.ADVANTAGES OF PROPOSED SYSTEM

- Improve data reliability and availability.
- The inter coding and outer coding of outsourced data enables efficient recovery when data corruption occurs.
- Using trusted Third Party Auditor(TPA) for data audit report and data audit delegation.
- Reduce server hacks or Byzantine failure to maintain reputation.
- Increase security by sending key to data owner to upload and retrive files.
- When one server is corrupted, the original data can be recovered by simply copying the entire data from one of the healthy servers.

3.2.SYSTEM ARCHITECTURE



Cloud Storage Public Auditing Architecture.

3.2.1.ARCHITECTURE DESCRIPTION

Encoding

We follow the same coding structure as HAIL [21] by exploiting both the within-server redundancy and cross-server redundancy to improve data reliability and availability. The key difference between the existing approaches and ours is that we adopt network coding instead of erasure codes for obtaining the cross-server redundancy. Specifically, we utilize the functional minimum storage regenerating (FMSR) code [19] as the cross-server code. FMSR belongs to Maximum Distance Separable (MDS) codes. An MDS code is defined by the parameters (s,k), where $k<s$. An (s,k)-MDS code means that the original data can be reconstructed from any k out of s servers. FMSR encodes a data file F of size $|F|$ into $s(s-k)$ data blocks of size $|F|/(k(s-k))$ each.

1) Outer code. Let $F = (m1,m2,\cdots,mn)$ be the data file, and $EM = [\alpha l,j]$ be an encoding matrix for some coefficients in the Galois field $GF(28)$ where $l = 1,\cdots,s(s-k),j = 1,\cdots,k(s-k)$. Each row vector of EM is an encoding coefficient vector (ECV) that contains $k(s-k)$ elements. We use $ECVi$ to denote the ith row vector of EM. For each block $mi(1 \le i \le n)$, we first divide it into $k(s-k)$ equal-size native blocks. Then, we encode these $k(s-k)$ native blocks into $s(s-k)$ encoded blocks, denoted by Pi,l which is computed by the scalar product of $ECVi$ and the native blocks vector mi, i.e. $Pi,l = \sum k(s-k)\ j{=}1\ \alpha l,jmi,j$, where $1 \le i \le n,1 \le l \le s(s-k)$. All arithmetic operations are performed over $GF(28)$. Each Pi,l is formed by a linear combination of the $k(s-k)$ native blocks. The encoded blocks Pi,l are then stored in the s storage servers, each having $s-k$ blocks. We use $Pi,t,j(1 \le i \le n,1 \le t \le s,1 \le j \le s-k)$ to denote the encoded block on a server, i.e. the j-th encoded block of mi on the t server. There are many ways of constructing EM, as long as it satisfies the MDS property and the repair MDS property [19].

2) Inner code. In order to save communication cost, we use an (s',k') erasure codes as the within-server code to encode each Pi,t,j into a new encoded block $P'i,t,j,q(1 \le q \le s')$. An (s',k') erasure code encodes k' fragments of data block into s' fragments such that up to $\lfloor (n'-k')/2 \rfloor$ errors, or up to $n'-k'$ erasures can be corrected. When a small fragment is corrupted, the server can recover the original data from the corruption locally instead of retrieving data blocks from other healthy servers.We use an (4,2)-FMSR code to achieve the cross-server redundancy and an (5,3) erasure code to achieve the within-server redundancy.

Initialization

Let G and GT be multiplicative cyclic groups of the same prime order p. A bilinear map is a map e: $G \times G \rightarrow GT$ with the following properties [31]: 1)Computable: there exists an efficiently computable algorithm for computing e; 2)Bilinear: for all $u,v \in \mathbb{Z}p$, it holds that $e(gu,gv) = e(g,g)uv$; 3)Non-degenerate: $e(g,g) \not= 1$ for any generator g of G; 4)for any $u1,u2,v \in G$, $e(u1u2,v) = e(u1,v) \cdot e(u2,v)$. Let $h(\cdot)$: $\{0,1\}* \rightarrow G$ be a secure hash function mapping a string to G uniformly. The system parameters and metadata tags are generated as follows. 1) KeyGen: The data owner randomly selects an element $r \in \mathbb{Z}*p,X \in G\backslash\{1\}$.

Then the data owner computes $R = g-r, A = e(X,g)$. The system public parameters are (g,h,p,G,GT,e), the public key is $pk = (R,A)$ and the secret key is $sk = (r,X)$. 2) TagGen: Given a file F, the data owner generates an identity fid for F and divides F into n blocks, i.e. $F = (m1,\cdots,mn)$ where $mi\in\mathbb{Z}*p$. For each block mi, the data owner encodes it into $s's(s-k)$ encoded blocks $Pi,l,q(1 \le i \le n, 1 \le l \le s(s-k), 1 \le q \le s')$ via a (s,k)-FMSR code and a (s',k')-erasure code. In order to tolerate cloud server data corruption, the data owner

stores $ns's(s-k)$ encoded blocks in s cloud severs. Each cloud server stores $ns'(s-k)$ encoded blocks. In each cloud server, these $ns'(s-k)$ encoded blocks are organized by an rb23Tree $Tt(1 \le t \le s)$. In each Tt, each node stores $s'(s-k)$ encoded blocks of the same original data block. The data owner then computes the hash value of the $s'(s-k)$ encoded blocks of data block $mi,t(1 \le i \le n, 1 \le t \le s)$ for each cloud server, i.e. $Hi,t = h(fid||Pen)$, where $Pen = P'i,t,1,1||\cdots||P'i,t,1,s'||\cdots||P'i,t,s-k,1||\cdots||P'i,t,s-k,s'$ Finally, the data owner computes the tag $\sigma i,t$ for the encoded blocks in each server $mi,t : \sigma i,t = (XPenHi,t)r$. Denote the set of all tags by $\phi t = \{\sigma i,t\}1\le i\le n,1\le t\le s$. Then, the data owner sends the encoded blocks $P'i,t,j,q$ with the information $aus,t = \{fid,\phi t,Tt\}$ to the corresponding server, sends the fid and tag value $v(root)t$ of root node of each rb23Tree to TPA and keeps the information $auo,t = \{fid,v(root)t\}$ with the encoding matrix EM locally. 3) rb23Tree: The range-based 2-3 tree or rb23Tree for short can not only offer the dynamic property of 2-3 trees with logarithmic complexity but also allows the verifier to verify the value and index of the leaf node. In the rb23Tree, each node stores three types of information: $\cdot l(k)$: the height of node k. The height of leaf node is defined 1. $\cdot r(k)$: the range value of node k, namely the number of leaves corresponding to the subtree rooted at k. If k is a leaf, $r(k)$ is 1 and if k is NULL, $r(k)$ is 0. The $r(k)$ of the root node is the number of leaves in the rb23Tree. $\cdot v(k)$the tag value of node k. $v(k)$ is defined as $H(l(k)||r(k)||v(ch1)||v(ch2)||v(ch3))$ or ek or 0 when $l(k) > 1$ or k is a leaf or k is NULL, respectively. Here, $||$ is the concatenation operation, $ch1,ch2,ch3$ are the tree left-to-right children of k (when k only has two children, $ch3$ is NULL), ek is the element value stored in leaf node k, and $H()$ is a collision-resistant hash function.We also define πi to be a proof path for locating the ith leaf by traversing the path starting at the root node. We also define the $min(k)$ and $max(k)$, which denote the minimum and maximum leaf indices that can be reached via node k, respectively. When locating an appointed leaf node whose index is i, we need to calculate the values of $min()$ and $max()$ from the root node to subjacent node step by step. Note that a node k is on the path from the ith leaf node to the root node if and only if $i\in \{min(k),max(k)\}$. Assuming the proof path $\pi i = \{k1,\cdots,kz\}$, where $k1$ is the ith leaf node, kz is the root node, and each node $kj\in\pi i$ is associated with an 8-element tuple $mark(kj) = \{l(kj),r(kj),r(c1),v(c1),r(c2),v(c2),r(c3),v(c3)\}$. When $j = 1$, kj is the leaf node and $r(kj)$ in $mark(kj)$ is $v(kj)$, i.e., the tag value of the leaf node. $c1,c2,c3$ are kj's three left-to-right children and $r(ct) = v(ct) = -1(1 \le t \le 3)$ if $ct\in\pi i$, $r(ct) = v(ct) = 0$ if ct does not exist. In Fig.4, we give an example of rb23Tree. We use ki,j to denote a node where i is the height and the j is the index. Each inner node k stores $(l(k),r(k),v(k))$. Suppose we want to verify the information of the 7th leaf node. The proof path is $\pi i = \{k1,7,k2,3,k3,2,k4,1\}$.

Data Integrity Verification

After the encoded data file with the metadata tags and the rb23Tree are outsourced to the server, TPA can periodically launch integrity checks on behalf of the data owner. On receiving the challenge, the server generates a proof and sends it to TPA.

1) ChalGen: In each auditing round, TPA first randomly selects a number $t1 \in\mathbb{Z}*p$ and computes $c1 = gt1$. Then, TPA randomly picks c elements $I = \{s1,\cdots,sc\}$ from the set $1,\cdots,n$ where n is the number of data blocks. Without loss of generality, we assume $s1 \le \cdots \le sc$ that can be generated using pseudo-random permutation. For each si in I, TPA chooses a random value $vi\in\mathbb{Z}*p$. Then, the TPA sends the challenge $chal = \{(i,vi)i\in I,c1\}$ to each server. According to the $chal$, each server returns $\{Hi,t(i\in I,1\le t\le s)\}$ to TPA. In response, TPA chooses a random element $m\in GT$ and computes $c2 = Rt1,\omega t = m\cdot e(\prod i\in IHvii,t,c2)$. Finally, the value ωt is sent back to each server.

2) ProofGen: All servers run this algorithm to generate proofs to prove the integrity of the checked encoded blocks. Specifically, each server executes the following computing independently:

$$\sigma t = \prod i\in I\sigma vii,t,\mu t = \sum i\in IviPen,$$

$$m*t = \omega t\cdot e(\sigma t,c1) \qquad (3)$$

Each server then sends the $m*t$ and μt as the proofs to TPA. All the proof paths $\pi i(i\in I)$ are also returned to TPA. So for each server the proof P is $P = \{m*t,\mu t,\pi i(i\in I)\}$.

3) VerifyProof: TPA runs this algorithm to validate the proof P from each server. For $t = 1,\cdots,s$ TPA first calculates $Bt = e(X,c-1\ 2\ )\mu t$ according to the μt returned by CSP. Then, TPA checks whether equation (4) holds and executes Algorithm 1 to verify whether $position[k] = i$ and $value[k] = v(root)t$.

$mBt? = m*t$ (4)

Algorithm 1 *Examine(v(root)t,πi,t)*

This algorithm allows an entity, who knows $v(root)t$, to verify the ith element ei of (ordered set) $S = \{e1,\cdots,en\}$ is stored exactly at the ith leaf by examining proof path (ordered set) $\pi i,t = v1,\cdots,vk$ provided by the server.

```
1)initialize an array $position[1\cdots k] = 0$ and array $value[1\cdots k] = 0$. //$position$ tracks the index of the leaf that will be checked with $v(\cdot) = ei$, $value$ tracks $v(vj)$ where $vj \in \pi i,t(i\in I)$.
 2)$position[1] \leftarrow 1$, $value[1] \leftarrow ei$.
 3)for $j = 2,\cdots,k$, do //$vj$ has three children $ch1,ch2,ch3$
     if $ch1 \in \pi i,t(i\in I)$, i.e., $r(ch1) = -1, v(ch1) = -1$, then
         $position[j] \leftarrow position[j-1]$, $value[j] \leftarrow H(l(kj)||r(kj)||value[j-1]||v(ch2)||v(ch3))$.
     end if
     if $ch2 \in \pi i,t(i\in I)$, i.e., $r(ch2) = -1, v(ch2) = -1$, then
         $position[j] \leftarrow position[j-1] + r(ch1)$, $value[j] \leftarrow H(l(kj)||r(kj)||v(ch1)||value[j-1]||v(ch3))$.
     end if
     if $ch3 \in \pi i,t(i\in I)$, i.e., $r(ch3) = -1, v(ch3) = -1$, then
         $position[j] \leftarrow position[j-1] + r(ch1) + r(ch2)$, $value[j] \leftarrow H(l(kj)||r(kj)||v(ch1)||v(ch2)||value[j-1])$.
     end if
     end for
4)if $position[k] = i$ and $value[k] = v(root)t$, then
 return TRUE
else
 return FALSE
 end if
```

Secure Data Updates

In this subsection, we discuss the dynamic update operations including block modification, block insertion and block deletion.

1)UpdateRequest: The data owner sends the update request $\{op,I2 = \{i-1,i,i+1\}\}$ to each server where $op \in \{M,I,D\}$ is the update operation, i is the update index. After receiving the update request, each server returns the corresponding proof path $\pi j,t(j\in I2)$ to the data owner. The data owner then calls the $Examine(v(root)t,\pi j,t)(j\in I2)$ to verify the validity of the path. If all verifications have been passed, the data owner executes the following operations according to the op (Without loss of generality, we assume $2 \le i \le n$.). $op = M$: The data owner downloads $s'k(s-k)$ encoded blocks of mi from any k of the s servers and decodes them to recover the original data block mi (see the Decoding Procedure). Then data owner encodes the new block $m*i$ using the original encoding matrix EM stored locally, generate new encoded blocks $P*i,t,j,q$ and compute the new tags $\sigma*i,t$. The new encoded blocks, the new tags are sent to the each corresponding server. $op = I$: The data owner generates the encoded blocks $P*i,t,j,q$ for the new block $m*i$ and computes the tags $\sigma*i,t$. The new encoded blocks, the new tags are then sent to the each corresponding server. $op = D$: The data owner sends the deletion instruction and the index i to the each corresponding server.

2) Update: After receiving the update request, each server will adjust his own rb23Tree Tt according to the request. According to the proof path $\pi j,t(j\in I2)$ received from each server, the data owner can construct a partial rb23Tree and update the information on $\pi j,t(j\in I2)$ by himself. We use $\pi jnew,t(j\in I2)$ to denote the new proof path the data owner maintains after updating path $\pi j,t$ himself. The data owner can compute a new $v(root)\ new,t$ according to the $\pi jnew,t$. In addition, we use $T'\ t$ to denote the new rb23Tree in the server t after updates. After adjustment, each server will send a new path information $\pi'\ i-1,t$ or $\pi'\ i+1,t$ to the data owner according to the new rb23Tree $T'\ t$. Then, the data owner calls the $Examine(v(root)t,\pi j',t)(j' = \{(i-1)',(i+1)'\})$ to compute the new root value $v(root)'\ t$ of $T'\ t$, which is further compared with the $v(root)\ new,t$ computed by the data owner himself to verify the correctness of the update request execution and the rb23Tree update.

Data Recovery

By periodical integrity checking, the TPA may find out the outsourced data is corrupted. Then, the TPA can locate the corrupted fragments or failed server via binary verification just like the binary search and return the positions to the data owner. When a server is still available but some small fragments of data are corrupted, i.e., for the encoded block $P'i,t,j,q(1 \le i \le n, 1 \le t \le s, 1 \le j \le s-k, 1 \le q \le s')$, the number of errors is less than $\lfloor (s'-k')/2 \rfloor$, or the number of erasures is less than $s'-k'$. The server then can correct the errors or erasures locally by the erasure codes, which involves no communication cost, and there is no need to recompute the metadata tags. When a server is down, the data owner can execute the iterative recovery procedure to recover the failure and generate new encoded blocks and the corresponding metadata tags. Iterative Recovery Procedure: The recovery process for a permanent single-server data corruption is as follows:

1) Select $s-1$ $ECVs$ randomly. Suppose the encoded blocks of the mi on the server t are corrupted, i.e., $P'i,t,1,1,\cdots,P'i,t,1,s',\cdots,P'i,t,s-k,1,\cdots,P'i,t,s-k,s'$. The data owner selects $s-1$ $ECVs$ from encoded

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com

matrix EM as follows: exclude the $s - kECVs$ from $(i-1)(s-k)+1$ to $(i-1)(s-k)+s-k$ where $i = 1,\cdots,s$. For each $j = 1,\cdots,s$ and $j \neq i$, choose one ECV from $(j-1)(s-k)+1$ to $(j-1)(s-k)+s-k$ randomly. Then $s - 1$ $ECVs$ are selected. Each ECV in the encoded matrix EM is corresponding to one of the encoded blocks. We denote these $ECVs$ by $ECVi1,ECVi2,\cdots,ECVis-1$.

2) Generate a repair matrix. The data owner constructs a repair matrix $RM = [\gamma i,j]$, where $i = 1,\cdots,s - k, j = 1,\cdots,s - 1$. Each element $\gamma i,j$ is randomly selected from $GF(28)$.

3) Compute $ECVs'$ for the new encode blocks and generate a new encoding matrix EM'. The data owner multiplies the RM generated in 2) with the $ECVs$ picked in 1) to construct $s - k$ new $ECVs'$, which are denoted by $ECV\,'\,i = \sum s-1\ j=1\ \gamma i,jECVij$ for $i = 1,2,\cdots,s - k$. Generate a new encoding matrix denoted by EM' as follows: when server $i(1 \leq i \leq s)$ is a healthy server the corresponding $s - k$ row vectors of EM' is $ECVij$, where $(i - 1)(s - k) + 1 \leq j \leq (i - 1)(s - k) + s - k$ and each $ECVi$ is selected from the original EM. When server i is corrupted, the corresponding $s - k$ row vectors of EM' are $ECV\,'\,j$ where $1 \leq j \leq s - k$.

4) Check whether both the MDS and repair MDS properties are satisfied or not. If either check fails, the data owner returns to 1) and repeats the above steps.

5) Download the actual encoded blocks and regenerate new encoded blocks. Using the RM multiply the $s - 1$ blocks selected from $s - 1$ servers corresponding to the $s-1$ $ECVs$ selected in 1) to generate new encoded blocks, which are encoded again via a (s',k')-erasure codes. The encoded blocks, the corresponding metadata tags and rb23Tree are stored in a new server.

IV.CONCLUSION

In this paper, we proposed a new dynamic proof of retrievability scheme for coded cloud storage systems. Network coding and erasure codes are adopted to encode data blocks to achieve within-server and cross server data redundancy, tolerating data corruptions and supporting communication-efficient data recovery. By combing range-based 2-3 tree and an improved version of aggregatable signature-based broadcast (ASBB) encryption, our construction can support efficient data dynamics while defending against data replay attack and pollution attack. Security analysis and experimental evaluations demonstrated the practicality of our construction in coded cloud storage systems.

REFERENCES

[1] Zhengwei Ren, Lina Wang, Qian Wang, Member, IEEE, Rongwei Yu, and Ruyi Deng,"Dynamic Proofs of Retrievability for Coded Cloud Storage Systems"

[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, "Above the Clouds: A Berkeley View of Cloud Computing," Commun. ACM, vol. 53, no. 4, pp. 50-58, 2010.

[3] Amazon.com, "Amazon S3 Availability Event: July 20, 2008," http://status.aws.amazon.com/s3-20080720.html, July 2008.

[4] S. Wilson, "Appengine Outage," http://www.cioweblog.com/50226711/appengine outage.php, June 2008.

[5] B. Krebs, "Payment Processor Breach May Be Largest Ever," http://voices.washingtonpost.com/securityfix/2009/01/payment processor breach may b.html, 2009.

[6] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable Data Possession at Untrusted Stores," Proc. 14th ACM Conf. Computer and Comm. Security (CCS'07), pp. 598-609, 2007.

[7] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling Public Verifiability and Data Dynamic for Storage Security in Cloud Computing," Proc. 14th European Symp. Research in Computer Security (ESORICS'09), pp. 355-370, 2009.

[8] A. Juels and B.S. Kaliski, "PORs: Proofs of Retrievability for Large Files," Proc. 14th ACM Conf. Computer and Comm. Security (CCS'07), pp. 584-597, 2007.

[9] C. Erway, A. Kupcu, C. Papamanthou, and R. Tamassia, "Dynamic Provable Data Possession," Proc. 16th ACM Conf. Computer and Comm. Security (CCS'09), pp. 213-222, 2009.

[10] 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), pp. 90-107, 2008.