LT Codes-based Secure and Reliable Cloud Storage Service

Ning Cao* Shucheng Yu† Zhenyu Yang‡ Wenjing Lou§ Y. Thomas Hou§
* Worcester Polytechnic Institute, Worcester, MA, USA
† University of Arkansas, Little Rock, AR, USA
‡ Amazon Web Services LLC, Seattle, WA, USA
§ Virginia Polytechnic Institute and State University, VA, USA

***Abstract*—With the increasing adoption of cloud computing for data storage, assuring data service reliability, in terms of data correctness and availability, has been outstanding. While redundancy can be added into the data for reliability, the problem becomes challenging in the "pay-as-you-use" cloud paradigm where we always want to efficiently resolve it for both corruption detection and data repair. Prior distributed storage systems based on erasure codes or network coding techniques have either high decoding computational cost for data users, or too much burden of data repair and being online for data owners. In this paper, we design a secure cloud storage service which addresses the reliability issue with near-optimal overall performance. By allowing a third party to perform the public integrity verification, data owners are significantly released from the onerous work of periodically checking data integrity. To completely free the data owner from the burden of being online after data outsourcing, this paper proposes an exact repair solution so that no metadata needs to be generated on the fly for repaired data. The performance analysis and experimental results show that our designed service has comparable storage and communication cost, but much less computational cost during data retrieval than erasure codes-based storage solutions. It introduces less storage cost, much faster data retrieval, and comparable communication cost comparing to network coding-based distributed storage systems.**

I. Introduction

The many advantages of cloud computing are increasingly attracting individuals and organizations to move their data from local to remote cloud servers [1]. In addition to major cloud infrastructure providers [2], such as Amazon, Google, and Microsoft, more and more third-party cloud data service providers are emerging which are dedicated to offering more accessible and user friendly storage services to cloud customers. Examples include Dropbox [3] which already has millions of users. It is a clear trend that cloud storage is becoming a pervasive service.

Along with the widespread enthusiasm on cloud computing, however, concerns on data security with cloud storage are arising due to unreliability of the service. For example, recently more and more events on cloud service outage or server corruption with major cloud service providers are reported [4], [5], be it caused by Byzantine failures and/or malicious attacks. Such a reality demands for reliable data storage to tolerate certain outage/corruption. In particular, the cloud storage service should offer cloud customers with capabilities of: 1) timely detection of any server (and hence data) corruption event, 2) correct retrieval of data even if a limited number of servers are corrupted, and 3) repair of corrupted data from uncorrupted data. Although existing techniques have provided solutions for them individually, the main challenge for cloud storage service is to simultaneously provide these capabilities at minimal cost. This is because in cloud computing both data storage and transmission are charged in the "pay-as-you-use" manner. Solutions of high cost will discourage user engagement and be of less practical use. Moreover, it is important to set cloud customers free by minimizing the complexity imposed on them in terms of computation/communication cost and burden of being online.

Existing solutions address the reliability issue by adding data redundancy to multiple servers. These techniques can be categorized into replication-based solutions and erasure codes-based ones. Data replication is the most straightforward way of adding redundancy. The advantage of replication is its simplicity in data management. Repair of data on corrupted servers is also straightforward by simply copying the entire data from a healthy server. The main drawback of replication is its high storage cost. Moreover, replication-based solutions cannot satisfy the high-throughput requirement in distributed storage service like cloud computing, where a large number of users may access the service concurrently. This is because different users may want to access different pieces of data on a server, which would cause less cache hits but frequent disk I/O requests. [6] provides a detailed analysis on this drawback.

As compared to its replication-based counterparts, erasure codes-based solutions can achieve the required reliability level with much less data redundancy [7]. Different from replication-based solutions, erasure codes-based ones are more suitable for distributed storage systems with concurrent user access. This is because every block of data on a server is useful for decoding the original data, which leads to a high cache hit rate of the system. There have been a large number of related works on erasure codes-based distributed storage systems [6], [8], [9]. The main drawback of existing optimal erasure codes-based systems, however, is the high communication cost needed for repairing a corrupted storage server. It is commonly believed that the communication cost is equal to the size of the entire original data [10]. For example, Reed-Solomon codes [11] usually need to reconstruct all the original packets in order to generate a fragment of encoded packets. Taking into consideration the large amount of data outsourced, the entire data reconstruction is expensive which makes this solution less attractive. Similarly, existing distributed storage systems based on near-optimal erasure codes [6] do not have an efficient solution for the data repair problem or pay no attention to it.

Recently Chen et al. [12] proposed a network coding-based storage system which provides a decent solution for efficient data repair. This scheme, based on previous work [10], [13]–[15], reduces the communication cost for data repair to the information theoretic minimum. This is achieved by recoding encoded packets in the healthy servers during the repair procedure. However, as network coding utilizes Gaussian elimination for decoding, the data retrieval in terms of computation cost is more expensive than erasure codes-based systems. Moreover, [12] adopts so-called functional repair for data repair, i.e., corrupted data is recovered to a correct form, but not the exact original form. While this is good for reducing data repair cost, it requires the data owner to produce new verification tags, e.g., cryptographic message authentication code, for newly generated data blocks. As the computational cost of generating verification tags is linear to the number of data blocks, this design will inevitably introduce heavy computation/communication cost on the data owner. Moreover, the data owner has to stay online during data repair.

In this paper, we explore the problem of secure and reliable storage in the "pay-as-you-use" cloud computing paradigm, and design a cloud storage service with the efficiency consideration of both data repair and data retrieval. By utilizing a near-optimal erasure codes, specifically LT codes, our designed storage service has faster decoding during data retrieval than existing solutions. To minimize the data repair complexity, we employ the exact repair method to efficiently recover the exact form of any corrupted data. Such a design also reduces the data owner's cost during data repair since no verification tag needs to be generated (old verification tags can be recovered as data recovery). By enabling public integrity check, our designed LT codes based secure cloud storage service (LTCS) completely releases the data owner from the burden of being online. Our contributions are summarized as follows,

1) We are among the first to explore the problem of secure and reliable cloud storage with the efficiency consideration for both data repair and data retrieval.

2) Our proposed cloud storage service provides a better overall efficiency of data retrieval and repair than existing counterparts. It also greatly reduces cost and burden of being online for the data owner by enabling public integrity check and exact repair.

3) The advantages of our proposed service are validated via both numerical analysis and experimental results.

II. Problem Formulation

A. The System Model

Considering a cloud data storage service which provides both secure data outsourcing service and efficient data retrieval and repair service, including four different entities: the data owner, the data user, the cloud server, and the third party server. The data owner outsources the encoded fragments of the file $\mathcal{M}$ to n cloud servers denoted as storage servers. If the data owner requires to keep the data content confidential, the file $\mathcal{M}$ can be first encrypted before encoding. Outsourced data are attached by some metadata like verification tags to provide integrity check capability. After the data outsourcing, a data user can select any k storage servers to retrieve encoded segments, and recover the file $\mathcal{M}$, which can be further decrypted in case the file is encrypted. Meanwhile, the third party server periodically checks the integrity of data stored in cloud servers. Failed cloud servers can be repaired with the help of other healthy cloud servers.

B. The Threat Model

The cloud server is considered as "curious-and-vulnerable". Specifically, the cloud server is vulnerable to Byzantine failures and external attacks. While Byzantine failures may be made by hardware errors or the cloud maintenance personnel's misbehaviors, external attacks could be ranging from natural disasters, like fire and earthquake, to adversaries' malicious hacking. After the adversary gains the control of the cloud server, it may launch the pollution attack or the replay attack which aims to break the linear independence among encoded data, by replacing the data stored in corrupted cloud server with old encoded data. If the cloud server is not corrupted, it correctly follows the designated protocol specification, but it will try to infer and analyze data in its storage and interactions during the protocol execution so as to learn additional information. This represents a threat to the privacy of cloud users' data stored on the server.

C. Design Goals

To provide secure and reliable cloud data storage services, our design should simultaneously achieve performance guarantees during data retrieval and repair.

- **Availability and Reliability:** By accessing any k-combination of n storage servers, the data user could successfully retrieve encoded data and recover all the original data. The data retrieval service remains functional when up to $n-k$ storage servers are corrupted in one round, and corrupted servers can be repaired from other healthy servers.
- **Security:** The designed storage service protects the data confidentiality and periodically checks the integrity of data in cloud servers to prevent data dropout or corruption.
- **Offline Data Owner:** Data owners can go offline immediately after data outsourcing, which means they are not required to be involved in tasks such as data integrity check and repair at a later stage.
- **Efficiency:** Above goals should be achieved with low storage, computation and communication cost for the data owner, data users and cloud servers.

D. Notations

- $\mathcal{M}$: the outsourced file, consisting of m original packets, $\mathcal{M} = (M_1, \ldots, M_m)$.

- S_l : the l-th storage server, $1 \leq l \leq n$.
- C_{li} : the i-th encoded packet stored in the l-th storage server, $1 \leq i \leq \alpha$.
- Δ_{li} : the coding vector of the encoded packet C_{li}.
- φ_l : the coding tag, used to verify all the coding vectors Δ_{li} in S_l.
- ϕ_{li} : the retrieval tag, used to verify C_{li} in the retrieval and repair.
- σ_{lij} : the verification tag, used to verify C_{li} in the integrity check, $1 \leq j \leq t$.

E. Preliminary on LT Codes

LT codes [16] has a typical property that the encoding procedure can generate unlimited number of encoded packets, each of which is generated by conducting bitwise XOR operation on a subset of original packets. LT codes can recover m original packets from any $m + O(\sqrt{m}\ln^2(m/\delta))$ coded packets with probability $1-\delta$. The decoding procedure is performed by the efficient Belief Propagation decoder [17] with complexity $O(m\ln(m/\delta))$. Code degree d is defined as the number of original packets that are combined into one coded packet. In LT codes, the distribution of code degree is defined by Ideal Soliton distribution or Robust Soliton distribution. The Ideal Soliton distribution is $\rho(i)$, i.e., $P\{d = i\}$, where $\sum_{i=1}^{m}\rho(i) = 1$ and

$$\rho(i) = P\{d = i\} = \begin{cases} 1/m & \text{if } i = 1 \\ 1/i(i-1) & \text{if } i = 2, \dots, m. \end{cases}$$

Robust Soliton distribution is $\mu(i)$, where $\mu(i) = (\rho(i) + \tau(i))/\beta$ and $\beta = \sum_{i=1}^{m}\rho(i) + \tau(i)$. Let $R = c \cdot \ln(m/\delta)\sqrt{m}$, and define $\tau(i)$ as follows,

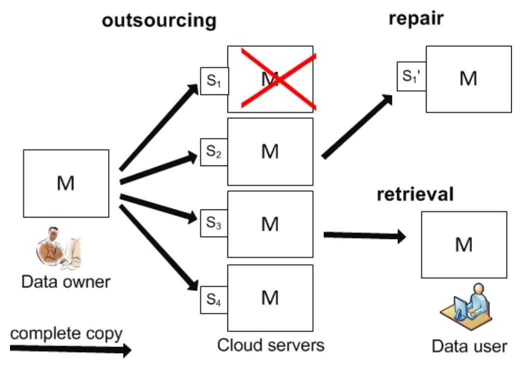
$$\tau(i) = \begin{cases} R/im & \text{if } i = 1, \dots, m/R - 1 \\ R\ln(R/\delta)/m & \text{if } i = m/R \\ 0 & \text{if } i = m/R + 1, \dots, m. \end{cases}$$

III. LTCS: Design Rationale

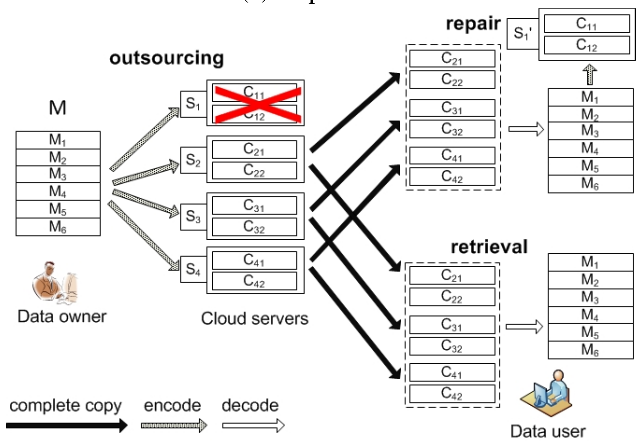
A. Enabling Reliability and Availability

To ensure the data reliability in distributed storage systems, various data redundancy techniques can be employed, such as replication, erasure codes, and network coding. Replication as shown in Fig. 1(a) is the most straightforward way of adding data redundancy where each of n storage servers stores a complete copy of the original data. Data users can retrieve the original data by accessing any one of the storage servers, and the corrupted server can be repaired by simply copying the entire data from a healthy server.

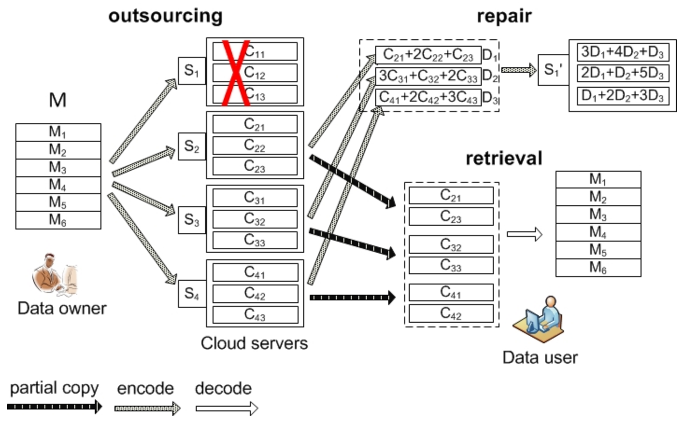
Given the same level of redundancy, the optimal erasure codes based distributed storage system as shown in Fig. 1(b) is more reliable by many orders of magnitude than the replication-based system [7]. Data users can recover the entire m original packets by retrieving the same number of encoded packets from any k-combination of n servers, and therefore every server only needs to store m/k encoded packets which is regarded as the property of optimal redundancy-reliability



(a) Replication



(b) Optimal Erasure codes



(c) Network coding

Fig. 1: Distributed storage systems based on different redundancy techniques.

tradeoff. However, its quadratic decoding complexity makes it very inefficient for data users to recover data during data retrieval. Moreover, the communication cost to repair a failed storage server is equal to the size of the entire original data in the optimal erasure codes-based distributed storage system [10], [15]. For example, as a typical optimal erasure codes, Reed-Solomon codes [11] usually need to reconstruct all the original packets in order to generate a fragment of encoded packets. In other words, one has to retrieve m encoded

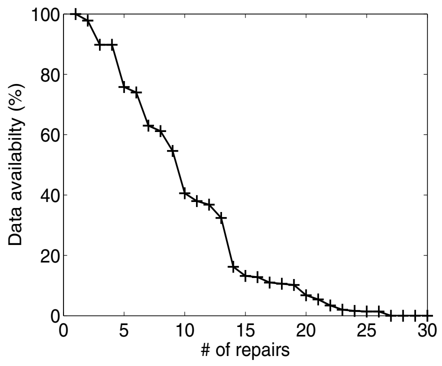


Fig. 2: Data availabity after functional repair as in LTNC.

packets in order to generate only m/k encoded packets for the corrupted server.

Network coding-based storage codes [10], [13]–[15] reduce the repair communication cost to the information theoretic minimum by combining encoded packets in the healthy servers during the repair procedure, where only m/k recoded packets are needed to generate the corrupted m/k encoded packets. Each server needs to store $2m/(k+1)$ encoded packets, which is more than optimal erasure codes, to guarantee that data users can retrieve m linearly independent encoded packets from any k-combination of n servers. Besides, the network coding-based storage codes have the similar inefficient decoding problem as optimal erasure codes due to the utilization of Gaussian elimination decoder.

To meet the efficient decoding requirement in the cloud data storage scenario where the data owner outsources huge amount of data for sharing with data users, our design is based on the near-optimal erasure codes, specifically LT codes, to store low-complexity encoded packets over n distributed servers. The fast Belief Propagation decoding for LT codes can be used during data retrieval in our LT codes based secure cloud storage service (LTCS). Data users can efficiently recover all the m of original packets from any $m(1+\epsilon)$ encoded packets which can be retrieved from any k-combination of n servers. To achieve so, every server needs to store at least $m(1+\varepsilon)/k$ encoded packets which is larger than the erasure codes but smaller than the network coding based storage codes.

B. Reducing Maintenance Cost

To prevent data dropout or corruption, the integrity of data stored in each server needs to be periodically checked. In [12], the data owner raises a challenge for every encoded packet to cloud servers. Taking into consideration the large number of encoded packets with substantial data redundancy in cloud servers, the cost of such private integrity check is somehow burdensome in terms of both computation and communication for data owners. LTCS utilizes the public integrity verification which enables the data owner to delegate the integrity check task to a third party server. Once there is a server failing to pass the integrity check, the third party server immediately reports it to the administrator of the cloud server who will then activate the repair process.

The repair task in our LT codes based storage service is accomplished by generating the exactly same packets as those previously stored in corrupted storage servers. Such repair method does not introduce any additional linear dependence among newly generated packets and those packets stored in healthy storage servers, and therefore maintains the data availability. Furthermore, we run the decoding over the encoded packets before outsourcing to guarantee the reliable data retrieval and recovery. Unlike the exact repair in our designed service, the functional repair is the other category of data repair, where the repair procedure generates correct encoded packets, but not the exactly same packets as those corrupted. Attempts to apply functional repair in the LT codes based distributed storage should first solve how to recode packets, because the random linear recoding in the functional repair of network coding-based storage codes cannot satisfy the degree distribution in LT codes. It seems that this problem can be solved by utilizing the recently proposed LT network codes (LTNC) which provides efficient decoding at the cost of slightly more communication in the single-source broadcasting scenario [18]. However, after several rounds of repair with same recoding operations regulated in LT network codes, data users experience decoding failure with high probability, as illustrated in Fig. 2, where data availability is the probability that data users could recover original data from any k-combination of n storage servers. The major reason is that recoding operations with the degree restriction in LT network codes introduce inneglectable linear dependence among recoded packets and existing packets in LT codes based storage service. Therefore, the functional repair is not suitable for LT codes-based storage service.

C. Offline Data Owner

In the repair procedure, network coding-based storage systems with functional repair generate new encoded packets to substitute corrupted data in the failed server. The data owner needs to stay online for generating necessary tags for these new packets [12]. In LTCS, all newly generated packets for the corrupted storage server in the repair procedure are exactly the same as old ones previously stored in the server, which means their corresponding metadata are also same. Like the distributed storage of data packets, these metadata can be stored in multiple servers and recovered in case of repairing corrupted servers. The replication or erasure codes (like Reed-Solomon codes) can be adopted to reliably backup these metadata. Hence, without the burden of generating tags and checking integrity, the data owner can stay offline immediately after outsourcing the data which makes LTCS more practical to be deployed in the cloud paradigm.

IV. LTCS: The Proposed Secure and Reliable Cloud Storage Service

In this section, we present the LT codes-based secure and reliable cloud storage service (LTCS), where n storage servers $\{S_l\}_{1 \leq l \leq n}$ are utilized to provide the data storage service for data owner and data users. Our data integrity technique is partially adapted from the BLS signature in POR [19].

A. Setup

Let $e : G \times G \rightarrow G_T$ be a bilinear map, where g is the generator of G, with a BLS hash function $H : \{0,1\}^* \rightarrow G$. The data owner generates a random number $\eta \leftarrow \mathbb{Z}_p$ and s random numbers $u_1, \ldots, u_s \leftarrow G$. The secret key sk is $\{\eta\}$, and the public key is pk $= \{u_1, \ldots, u_s, v\}$, where $v \leftarrow g^\eta$.

B. Data Outsourcing

The data outsourcing is to pre-process data and distribute them to multiple cloud servers. The file $\mathcal{M}$ is first equally split into m original packets, $M_1, \ldots, M_m$, with the same size of $\frac{|\mathcal{M}|}{m}$ bits. Following the Robust Soliton degree distribution in LT codes, m original packets are combined by exclusive-or (XOR) operations to generate $n\alpha$ encoded packets, where α is the number of packets outsourced to each storage server and set to $m/k \cdot (1+\varepsilon)$. For protecting data confidentiality, sensitive data could be encrypted before the encoding process. Existing data access control mechanisms [20] can be employed to prevent the cloud server from prying into outsourced data.

According to LT codes, all the m original packets can be recovered from any $m(1+\varepsilon)$ of encoded packets with probability $1-\delta$ by on average $O(m \cdot \ln(m/\delta))$ packet operations. However, the availability requirement specifies that data recovery should be always successful by accessing any k of healthy storage servers. To achieve this goal, the data owner checks the decodability of these encoded packets before outsourcing by executing the decoding algorithm. Specifically, all the $n\alpha$ encoded packets are divided into n groups, each of which consists of α packets, $\{\{C_{li}\}_{1\leq i\leq\alpha}\}_{1\leq l\leq n}$. The Belief Propagation decoding algorithm is then run on every k-combination of n groups. If the decoding fails in any combination, the data owner re-generates encoded packets and re-checks the decodability until every k-combination can recover all the m original packets. Once the encoding configuration successfully passes the decodability detection, it can be reused for all the storage services that specifies the same n and k.

For each encoded packet C_{li}, $1 \leq l \leq n$, $1 \leq i \leq \alpha$, three kinds of auxiliary data are attached, i.e., the coding vector, the retrieval tag, and verification tags. The coding vector Δ_{li} is a m-bit vector, where each bit represents whether the corresponding original packet is combined into C_{li} or not. The retrieval tag ϕ_{li}, computed by Eq. 1, is to verify the encoded packet C_{li} in data retrieval, and also in data repair if necessary.

$$\phi_{li} \leftarrow (H(l||i||C_{li}))^\eta \in G \tag{1}$$

To generate the verification tag for the purpose of integrity check, each encoded packet C_{li} is split into t segments, $\{C_{li1}, \ldots, C_{lit}\}$. Each segment C_{lij} includes s symbols in $\mathbb{Z}_p : \{C_{lij1}, \ldots, C_{lijs}\}$. For each segment C_{lij}, we generate a verification tag σ_{lij}, $1 \leq j \leq t$, in Eq. 2.

$$\sigma_{lij} \leftarrow (H(l||i||j) \cdot \prod_{\ell=1}^{s} u_\ell^{C_{lij\ell}})^\eta \in G \tag{2}$$

These data are outsourced to the l-th storage server in the form of $\{l, \{i, C_{li}, \Delta_{li}, \phi_{li}, \{\sigma_{lij}\}_{1\leq j\leq t}\}_{1\leq i\leq\alpha}, \varphi_l\}$, where φ_l is the coding tag to validate all the previously coding vectors. The computation of φ_l is shown in Eq. 3.

$$\varphi_l \leftarrow (H(l||\Delta_{l1}||\ldots||\Delta_{l\alpha}))^\eta \in G \tag{3}$$

C. Data Retrieval

Data users can recover original data by accessing any k of n cloud servers in the data retrieval. The data user first retrieves all the coding vectors and the coding tags stored in the selected k cloud servers, and performs the verification in Eq. 4. If the verification operation on any coding tag fails, the data user sends reports to the third party server and accesses one substitutive storage server.

$$e(\varphi_l, g) \stackrel{?}{=} e(H(l||\Delta_{l1}||\ldots||\Delta_{l\alpha}), v) \tag{4}$$

Once all the coding tags from k storage servers pass the validation, the data user partially executes the Belief Propagation decoding algorithm only with coding vectors, and records ids of coding vectors that are useful for the decoding. Meanwhile, the data user retrieves those corresponding useful encoded packets and their retrieval tags from corresponding storage servers, and verifies the integrity of encoded packets as shown in Eq. 5.

$$e(\phi_{li}, g) \stackrel{?}{=} e(H(l||i||C_{li}), v) \tag{5}$$

All the original packets in $\mathcal{M}$ can be recovered by performing the same XOR operations on encoded packets as those on coding vectors. Finally, the data user can decrypt the $\mathcal{M}$ and get the plaintext data if the file is encrypted before encoding. Note that if there exist some verification tags that fail in the integrity check procedure, the data user also reports them to the third party server and retrieves data from one substitutive storage server. When the third party server receives any failure reports from data users about either coding tags or verification tags, it will immediately challenge the corresponding server (details on challenge will be given in the following section).

D. Integrity Check

To monitor the integrity of data stored in the storage servers, the third party server periodically performs the integrity check over every storage server. The third party server first randomly picks $\alpha + t$ numbers, $a_1, \ldots, a_\alpha, b_1, \ldots, b_t \leftarrow \mathbb{Z}_p$, and then sends them to every storage server. The l-th storage server will compute s integrated symbols $\{\mu_{l\ell}\}_{1\leq\ell\leq s}$ and one integrated tag ς_l in Eq. 6. Note that a_i corresponds to the i-th encoded packet in every storage server, and b_j corresponds to the j-th segment in each encoded packet.

$$\mu_{l\ell} = \sum_{i=1}^{\alpha}\sum_{j=1}^{t} a_i b_j C_{lij\ell}, \; \varsigma_l = \prod_{i=1}^{\alpha}\prod_{j=1}^{t} {\sigma_{lij}}^{a_i b_j} \tag{6}$$

The third party server verifies these received integrated symbols $\{\mu_{l\ell}\}_{1\leq\ell\leq s}$ and the integrated verification tag ς_l, as

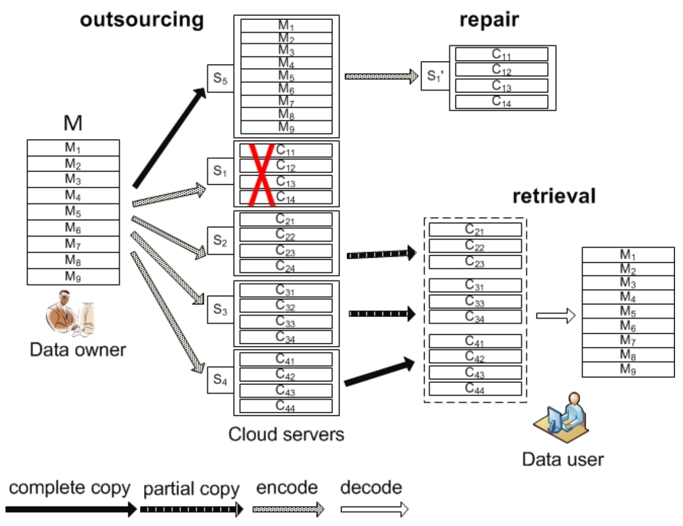


Fig. 3: LT codes-based cloud storage service (LTCS).

shown in Eq. 7.

$$e(\varsigma_l, g) \overset{?}{=} e(\prod_{i=1}^{\alpha}\prod_{j=1}^{t} H(l||i||j)^{a_i b_j} \cdot \prod_{\ell=1}^{s} u_\ell^{\mu_{l\ell}}, v) \tag{7}$$

If the verification fails, the third party server reports it to the data center, and the administrator of the storage server will reset the server software and start the data repair procedure.

E. Data Repair

It is commonly believed that all existing coding constructions must access the original data to generate coded packets, which means the communication cost of data repair for erasure codes is equal to the size of the entire original data [10]. A s-traightforward data repair method is therefore to recover all the original data packets whenever a storage server is corrupted. But such method will introduce much cost of both computation and communication. In LTCS as illustrated in Fig. 3, one repair server S_{n+1} is deployed to efficiently repair corrupted storage servers. Although other storage services based on optimal erasure codes or network coding can also integrate the repair server, they still introduce more computational cost during data retrieval (and storage cost for network coding-based service) than LTCS, which will be validated in section VI.

To accommodate the repair server, the data owner outsources all the original packets to the repair server S_{n+1} during data outsourcing. Each original packet is also attached by the verification tag which is generated in the same way as shown in Eq. 2. Besides, all the auxiliary data of storage servers are stored in the repair server as a backup. Similarly with the distributed data storage, the metadata including verification tags for original packets need to be reliably stored in n storage servers. Compared with the large size of encoded data, auxiliary data are quite small such that we can employ the simple replication or erasure codes to add redundancy.

To deal with the failure on the l-th storage server, the repair server uses all the corresponding coding vectors $\{\Delta_{li}\}_{1\leq i\leq\alpha}$ to generate encoded packets $\{C_{li}\}_{1\leq i\leq\alpha}$. Specifically, C_{li} is generated by the XOR combination of $|\Delta_{li}|$ original packets, as illustrated in Eq. 8, where $j_{li1}, \ldots, j_{li|\Delta_{li}|} \leftarrow \{1, \ldots, m\}$ correspond to the nonzero bits in the coding vector Δ_{li}. The repair server sends to S_l all the encoded packets with their tags in the form of $\{l, \{i, C_{li}, \Delta_{li}, \phi_{li}, \{\sigma_{lij}\}_{1\leq j\leq t}\}_{1\leq i\leq\alpha}, \varphi_l\}$.

$$C_{li} = M_{j_{li1}} \oplus \ldots \oplus M_{j_{li|\Delta_{li}|}} \tag{8}$$

The repaired server S_l authenticates received encoded packets $\{C_{li}\}_{1\leq i\leq\alpha}$ and auxiliary tags as in the data retrieval and integrity check. If the authentication fails, the repair server itself may be corrupted and need repair.

The third party server also challenges the repair server S_{n+1} to check the integrity of original packets. Since there are m packets stored in S_{n+1}, instead of α in storage servers, the third party server should generate $m + t$ random numbers, $a_1, \ldots, a_m, b_1, \ldots, b_t \leftarrow \mathbb{Z}_p$. The integrated symbols $\{\mu_{(n+1)\ell}\}_{1\leq\ell\leq s}$ are then generated from the m original packets, $\mu_{(n+1)\ell} = \sum_{i=1}^{m}\sum_{j=1}^{t} a_i b_j C_{(n+1)ij\ell}$, where $C_{(n+1)i} = M_i$. There are similar changes in the generation of the integrated verification tag, $\varsigma_{n+1} = \prod_{i=1}^{m}\prod_{j=1}^{t} \sigma_{(n+1)ij}{}^{a_i b_j}$. The repair server is less likely to be corrupted than storage servers, since it does not participate in the data retrieval service for data users. Even when the repair server is found to be corrupted and needs repair, all the original packets and auxiliary data can be recovered by performing data retrieval from any d of healthy storage servers. Therefore, there is no single point of failure.

V. Security Analysis

A. Protection of Data Confidentiality and Integrity

For protecting data confidentiality, existing encryption techniques or data access control schemes [20] can be utilized before the encoding process, which prevent the cloud server from prying into outsourced data. With respect to the data integrity, LTCS utilizes various cryptographic tags to resist the pollution attack during the data repair and retrieval procedures. LTCS is also secure against the replay attack which is presented in the network coding-based distributed storage system [12]. To lunch the replay attack, the adversary first corrupts some storage servers and backups encoded packets stored in these servers. After several rounds of data repair, the adversary corrupts the same storage servers as before, and then substitutes new encoded packets with specific old packets. Since the verification tag only binds the storage server id and the packet id, not the freshness of the packet, the substituted old packets could pass the integrity verification. As a result, such substitution makes encoded packets stored in specific k-combinations of n storage servers linearly dependable, and the data recovery would fail when all other $n - k$ storage servers are corrupted. Actually, if the data repair mechanism is designed to generate new packets which are different from the old packets stored in the same storage server, any coding-based distributed storage system is somehow vulnerable to such kind of attack. In other words, the functional repair itself

TABLE I: Performance complexity analysis of storage services based on different redundancy techniques.

	Network Coding	Reed-Solomon	LTCS
Total server storage	$O((2n/(k+1)) \cdot \lvert\mathcal{M}\rvert)$	$O((1+n/k) \cdot \lvert\mathcal{M}\rvert)$	$O((1+n(1+\varepsilon)/k) \cdot \lvert\mathcal{M}\rvert)$
Encoding computation	$O(2nm^2/(k+1))$	$O(nm^2/k)$	$O((nm(1+\varepsilon)\ln m)/k)$
Retrieval communication	$O(\lvert\mathcal{M}\rvert)$	$O(\lvert\mathcal{M}\rvert)$	$O(\lvert\mathcal{M}\rvert)$
Retrieval computation	$O(m^2)$	$O(m^2)$	$O(m \ln m)$
Repair communication	$O(2T/(k+1) \cdot \lvert\mathcal{M}\rvert)$	$O(T(1/k+1/n) \cdot \lvert\mathcal{M}\rvert)$	$O(T((1+\varepsilon)/k+1/n) \cdot \lvert\mathcal{M}\rvert)$

has the possibility to break the decodability. By contrast, LTCS employs the exact repair method where the newly generated packets are the same as those previously stored packets. The replay attack becomes invalid since there is no difference between old and new packets in the same storage server. Furthermore, LTCS examines the data decodability from any k-combination of storage servers before outsourcing, which guarantees that original data could be recovered even when the adversary corrupts both the repair server and at most $n-k$ storage servers in one round.

B. Verification Correctness in Integrity Check

The verification correctness in Eq. 7 is proved in Eq. 9.

$$\begin{aligned}
e(\varsigma_l, g) &= e(\prod_{i=1}^{\alpha}\prod_{j=1}^{t} {\sigma_{lij}}^{a_i b_j}, g) \\
&= e(\prod_{i=1}^{\alpha}\prod_{j=1}^{t} (H(l||i||j)^{a_i b_j} \cdot \prod_{\ell=1}^{s} u_\ell^{a_i b_j C_{lij\ell}}), g)^{\eta} \\
&= e(\prod_{i=1}^{\alpha}\prod_{j=1}^{t} H(l||i||j)^{a_i b_j} \cdot \prod_{\ell=1}^{s}\prod_{i=1}^{\alpha}\prod_{j=1}^{t} u_\ell^{a_i b_j C_{lij\ell}}, v) \\
&= e(\prod_{i=1}^{\alpha}\prod_{j=1}^{t} H(l||i||j)^{a_i b_j} \cdot \prod_{\ell=1}^{s} u_\ell^{\sum_{i=1}^{\alpha}\sum_{j=1}^{t} a_i b_j C_{lij\ell}}, v) \\
&= e(\prod_{i=1}^{\alpha}\prod_{j=1}^{t} H(l||i||j)^{a_i b_j} \cdot \prod_{\ell=1}^{s} u_\ell^{\mu_{l\ell}}, v). \qquad (9)
\end{aligned}$$

VI. Performance Analysis

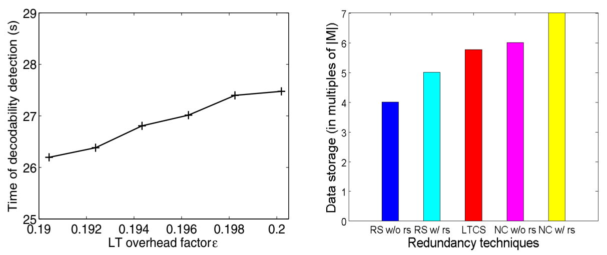
In this section, we demonstrate the performance of storage services based on different redundancy techniques by both theoretical complexity analysis and experimental evaluation. We set the same desired reliability level as network coding-based distributed storage system RDC-NC [12], where $n = 12$, $k = 3$. Other parameters are set from the consideration of specific properties of network coding (NC), Reed-Solomon codes (RS), and LT codes. For LTCS, $m = 3072, \alpha = m(1+\varepsilon)/k, d_s = 1, d_r = k, \beta = \alpha, \delta = 1, c = 0.1$, where $\varepsilon \sim O(\ln^2(m/\delta)/\sqrt{m})$ is the LT overhead factor. d_s and d_r represent the number of cloud servers participating in the repair of corrupted storage server and corrupted repair server, respectively. β represents the number of packets retrieved from each participating server during repair. For Reed-Solomon codes based storage system, $m = 6$ *or* $12, \alpha = m/k, d = k, \beta = \alpha$; for network coding based storage system, $m = 6$ *or* $12, \alpha = 2m/(k+1), d = k, \beta = \alpha/d$. The whole experiment system is implemented by C language on a Linux Server with Intel Xeon Processor 2.93GHz. Besides, the performance of network coding and Reed-Solomon codes is optimized by employing table lookup in the multiplication and division over $GF(2^8)$, and we evaluate their performance with or without repair server (rs), respectively. The performance complexity comparison among storage services based on different redundancy techniques with repair server is shown in Tab. I, where T is the number of corrupted storage servers in one round, $0 \leq T \leq n-k$.

A. Outsourcing

As described in section 4.2, the data owner detects the decodability in the encoding procedure to guarantee data availability. To check all k-combinations of n groups, the data owner has to execute $\binom{n}{k}$ times of the Belief Propagation decoding algorithm. For the efficiency purpose, this decoding process can be partially executed where only coding vectors follow the decoding steps and data packets are not involved. If there exists a combination that cannot recover all the original packets, the data owner will re-generate $n\alpha$ coding vectors according to LT codes and re-detect them, where α is equal to $m(1+\varepsilon)/k$. Once all the $\binom{n}{k}$ combinations successfully pass the decodability detection, corresponding coding vectors can be reused for all the storage services that specifies the same n and k. As illustrated in Fig. 4(a), the larger ε makes the decodability detection more costly because of the linear relation between ε and α, namely the number of coding vectors in each group. Considering that the larger ε leads to more storage cost and repair communication cost, the following evaluations are conducted by setting ε to the smallest one as 0.1904, which corresponds to $\alpha = 1219$.

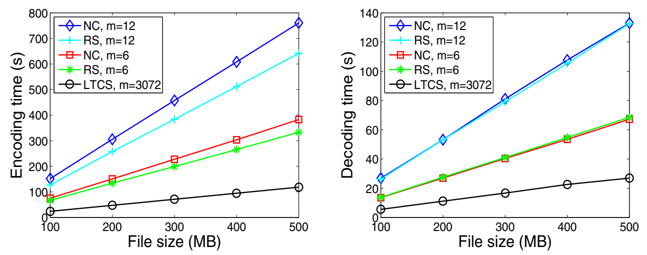
Once one set of $n\alpha$ coding vectors pass the decodability detection, encoding operations are performed on real data packets via the XOR combination. Although the number of encoded packets in LTCS, $n\alpha$, is several hundreds times larger than in other storage service based on network coding or Reed-Solomon codes, the computational cost of encoding in LTCS is much less than the later, as illustrated in Fig. 5(a). The main reason for such big advantage is that the average degree of encoded packets is $O(ln(m/\delta))$ and $O(m)$ in two services, respectively. Furthermore, the combination for encoding is the efficient XOR in LTCS while the linear combination in network coding or Reed-Solomon codes involves the multiplication operations with coefficients. The total number of encoded packets in Reed-Solomon codes-based service is less than network coding-based one so the encoding procedure introduces different computational cost in two services.

As for the data storage in LTCS, every storage server stores



(a)Time of detecting decodability (b)Data storage cost

Fig. 4: Outsourcing performance with different ε. n = 12, k = 3, m = 3072.



(a)Encoding (b)Decoding

Fig. 5: Encoding and decoding time for different size of file. n=12, k=3.

α encoded packets, each of which has the size of $|\mathcal{M}|/m$. And the repair server stores all the m original packets with the same size. The total data storage in cloud servers is the sum of all encoded packets in n storage servers and all original packets in the repair server, which is $O(n\alpha \cdot |\mathcal{M}|/m + |\mathcal{M}|)$, i.e., $O([1+n(1+\varepsilon)/k] \cdot |\mathcal{M}|)$. The data storage cost in LTC-S is larger than Reed-Solomon codes-based storage service because LT codes is a near-optimal erasure codes in terms of redundancy-reliability tradeoff. By contrast, the total data storage in existing network coding-based storage service is $O(|\mathcal{M}|[2n/(k+1)])$ as illustrated in Fig. 4(b). If we integrate the repair server into this service, the storage cost will be $O(|\mathcal{M}|[1+2n/(k+1)])$ which is much larger than LTCS.

B. Data Retrieval

The availability in data retrieval is guaranteed by the decodability detection before data outsourcing and the exact repair of corrupted data. Recall that the data user first retrieves $k\alpha$, i.e. $m(1+\varepsilon)$, of coding vectors from k storage servers, and then only retrieve m of encoded packets that are useful for decoding. Therefore, the communication cost during data retrieval in LTCS is the same $O(|\mathcal{M}|)$ as the network coding-based storage system where any m of encoded packets are linearly independent with high probability.

The computational complexity of Belief Propagation decoding in LTCS is $O(m \cdot \ln(m/\delta))$ for the data user, where δ is set to 1. By contrast, the other storage services based on network coding or Reed-Solomon codes usually use the costly decoding algorithms with higher computational complexity, $O(m^2)$. Although the total number of original packets, m, may be smaller in the other two storage services than in LTCS, the decoding process for the data user in LTCS performs at least two times faster than in the other two storage services, as illustrated in 5(b). This efficient decoding process demonstrates that LTCS is more appropriate than other redundancy-based storage services in the cloud storage paradigm, where data retrieval is a routine task for data users.

C. Integrity Check

To check the integrity of data stored in a storage server, the third party server needs to perform one integrated challenge in LTCS, which means only two bilinear maps in Eq. 7 are executed in order to check α encoded packets. Network coding-based service has to perform α times of challenges for each storage server where 2α bilinear maps are executed to check α of encoded packets. Similarly, the communication cost between the third party server and each storage server during one round of $Integrity\ Check$ in network coding-based service is almost α times more than that in LTCS.

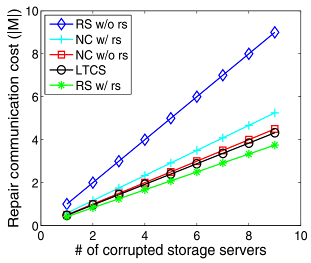


Fig. 6: Communication cost of repair. n=12, k=3.

D. Data Repair

When the repair server is corrupted, LTCS first retrieve β encoded packets from each of d_r healthy storage servers to recover all the original packets. In such case, the communication complexity from d_r healthy storage servers to the repair server is $O(d_r \cdot \beta \cdot |\mathcal{M}|/m)$, i.e., $O((1+\varepsilon) \cdot |\mathcal{M}|)$, where $d_r = k$, $\beta = \alpha$. If the repair server is not corrupted or has been repaired, the data repair of storage servers in LTCS is simply accomplished by the repair server generating β encoded packets for each corrupted storage server, where $d_s = 1$, $\beta = \alpha$. Assume the number of corrupted storage servers in one round is T, $0 \leq T \leq n-k$. The repair communication complexity in such scenario is $O(T\alpha \cdot |\mathcal{M}|/m)$, i.e., $O(T(1+\varepsilon)/k \cdot |\mathcal{M}|)$, where $|\mathcal{M}|/m$ is the size of each encoded packet.

Assume the corruption probability of the repair server is the same as storage servers, i.e., T/n. The total repair communication complexity is then calculated as $O(T(1+\varepsilon)/k \cdot |\mathcal{M}| + T/n \cdot |\mathcal{M}|)$, i.e., $O(T((1+\varepsilon)/k + 1/n) \cdot |\mathcal{M}|)$. As illustrated in Fig. 6, to repair different number of corrupted storage servers T, the communication cost in LTCS is only 15 percent more than Reed-Solomon codes-based service integrated with repair server, but smaller than that in network coding-based service.

VII. Related work

Network Coding-based Distributed Storage As a new data transmitting technique, network coding is different with traditional store-and-forward methods. Instead of simply forwarding previously received packets, network coding allows intermediate nodes to recode received packets before forwarding. It has been proved that random linear network coding over a sufficiently large finite field can achieve the multicast capacity [21], [22]. Since the data repair problem in the distributed storage is claimed to be mapped to a multicasting problem on the information flow graph [10], many network coding-based storage codes [10], [13]–[15], [23], [24] have been proposed to take advantage of this property of capacity achievability. By recoding encoded packets in healthy servers during the repair procedure, the repair communication cost is reduced to the information theoretic minimum. The achievable region of functional repair is characterized in [12], but a large part of the achievable region of exact repair remains open [15]. Furthermore, since network coding utilizes Gaussian elimination decoding algorithm, the data retrieval is more expensive than erasure codes-based system [12]. Therefore, these designs are only suitable in "read-rarely" storage scenarios, and cannot be efficiently deployed in the cloud storage system where data retrieval is a routine operation.

Remote Data Integrity Check The remote data integrity check problem is first explored in [25] which also checks the data availability and gives a formal security definition of "proof of retrievability". Subsequent works [19], [26], [27] improve this work by providing unlimited number of data integrity checking, taking less communication, or supporting secure and efficient dynamic data operations. Related works on "proof of data possession" [28], [29] focus on the data integrity check and perform more efficiently at the cost of security level. However, existing works based on "proof of retrievability" or "proof of data possession" do not have an efficient solution for the data repair problem or pay no attention to it.

VIII. Conclusions

In this paper, for the first time, we explore the problem of secure and reliable cloud storage with the efficiency consideration of both data repair and data retrieval, and design a LT codes-based cloud storage service (LTCS). To enable efficient decoding for data users in the data retrieval procedure, we adopt a low complexity LT codes for adding data redundancy in distributed cloud servers. Our proposed LTCS provides efficient data retrieval for data users by utilizing the fast Belief Propagation decoding algorithm, and releases the data owner from the burden of being online by enabling public data integrity check and employing exact repair. The performance analysis and experimental results show that LTCS has a comparable storage and communication cost, but a much faster data retrieval than the erasure codes-based solutions. It introduces less storage cost, much faster data retrieval, and comparable communication cost comparing to network coding-based storage services. Our future will address how to detect decodability more efficiently.

Acknowledgment

This work was supported in part by the US National Science Foundation under grants CNS-1117111, CNS-0831628, and CNS-0910531.

References

[1] S. Kamara and K. Lauter, "Cryptographic cloud storage," in *RLCPS*, 2010.

[2] M. Armbrust, A. Fox, and et al., "Above the clouds: A berkeley view of cloud computing," University of California, Berkeley, Tech. Rep. UCB-EECS-2009-28.

[3] http://www.dropbox.com/.

[4] "Summary of the amazon ec2 and amazon rds service disruption in the us east region," http://aws.amazon.com/message/65648/.

[5] "Microsoft cloud data breach heralds things to come," http://www.techworld.com.au/article/372111/microsoft_cloud_data_breach_heralds_things_come/.

[6] H. Xia and A. A. Chien, "Robustore: a distributed storage architecture with robust and high performance," in *Proc. of Supercomputing*, 2007.

[7] H. Weatherspoon and J. D. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *IPTPS*, 2002.

[8] J. Kubiatowicz, D. Bindel, Y. Chen, and et al, "OceanStore: an architecture for global-scale persistent storage," in *ASPLOS*. New York, NY, USA: ACM, 2000.

[9] R. B. Kiran, K. Tati, Y. chung Cheng, S. Savage, and G. M. Voelker, "Total recall: System support for automated availability management," in *Proc. of NSDI*, 2004.

[10] A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *ITIT*, 2010.

[11] I. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the SIAM*, 1960.

[12] B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote data checking for network coding-based distributed storage systems," in *Proc. of CCSW '10*, 2010.

[13] Y. Wu, "A construction of systematic mds codes with minimum repair bandwidth," in *IEEE Trans. Inf. Theory*, 2009.

[14] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran, "Exact regenerating codes for distributed storage," in *Proc. Allerton Conf. Control Comput. Commun.*, 2009, pp. 337–350.

[15] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A survey on network codes for distributed storage," *CoRR*, vol. abs/1004.4438, 2010.

[16] M. Luby, "Lt codes," in *Proc. of FoCS*, 2002, pp. 271–280.

[17] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Efficient erasure correcting codes," *ITIT*, no. 2, pp. 569–584, 2001.

[18] M.-L. Champel, K. Huguenin, A.-M. Kermarrec, and N. L. Scouarnec, "Lt network codes," *Proc. of ICDCS*, pp. 536–546, 2010.

[19] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proceedings of Asiacrypt*, 2008.

[20] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *Proc. of INFOCOM*, 2010.

[21] T. Ho, M. Medard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *ITIT*, 2006.

[22] P. Sanders, S. Egner, and L. Tolhuizen, "Polynomial time algorithms for network information flow," in *Proc. of SPAA*, 2003, pp. 286–294.

[23] J. Li, S. Yang, X. Wang, X. Xue, and B. Li, "Tree-structured data regeneration in distributed storage systems with regenerating codes," in *Proc. of IWQoS*, July 2009.

[24] A. Duminuco and E. Biersack, "A practical study of regenerating codes for peer-to-peer backup systems," in *Proc. of ICDCS*, June 2009.

[25] A. Juels and B. S. Kaliski, Jr., "Pors: proofs of retrievability for large files," in *Pro. of CCS*. New York, NY, USA: ACM, 2007, pp. 584–597.

[26] C. Wang, Q. Wang, K. Ren, and W. Lou, "Ensuring data storage security in cloud computing," in *Proc. of IWQoS*, 2009.

[27] K. D. Bowers, A. Juels, and A. Oprea, "Hail: a high-availability and integrity layer for cloud storage," in *Proc. of CCS*, 2009.

[28] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proceedings of CCS*. New York, NY, USA: ACM, 2007.

[29] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "Mr-pdp: Multiple-replica provable data possession," in *Proc. of ICDCS*, 2008.