Annamaneni Samatha, Nimmala Jeevan Reddy, P.Pradeep Kumar / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 5, September- October 2012, pp.1050-1055

Data Storage Collateral In Cloud Computing

Annamaneni Samatha*,Nimmala Jeevan Reddy,P.Pradeep Kumar*****

*(Student,Department of Computer Science, JNTU University, Hyderabad-85)
** (Student,Department of Computer Science JNTU University, Hyderabad-85)
***(Head of Department ,Department of Computer Science JNTU University, Hyderabad-85)

ABSTRACT

Cloud Computing has been predicted as the next-generation architecture of IT Enterprise. In additional to traditional solutions, where the IT services are under proper physical, logical and personnel controls, Cloud Computing moves the application software and databases to the large data centers, where the management of data has many security challenges To Overcome this, in this paper we focused on data security in cloud computing, which has always been an important aspect of quality of service. To ensure the correctness of users' data in the cloud, we propose an effective and flexible distributed scheme with two salient features, opposing to its predecessors. By utilizing the homomorphic token with distributed verification of erasure-coded data, our scheme achieves the integration of storage correctness insurance and data error localization, i.e., the identification of misbehaving g server(s). Unlike most prior works, the new scheme further supports secure and efficient dynamic operations on data base, including data update, delete and insert. Extensive security and performance analysis shows that the proposed scheme is highly efficient and resilient against Byzantine failure, malicious data modification attack, and even it avoids server colluding attacks.

Keywords - Cloud **Computing, Security, Distributed scheme, homomorphic token, distributed verification.**

I. INTRODUCTION

Several trends are opening up the era of Cloud Computing, which is an Internet-based development and use of computer technology. The ever cheaper and more powerful processors, together with the software as a service (SaaS) computing archi-tecture, are transforming data centers into pools of computing service on a huge scale. The increasing network bandwidth and reliable yet flexible network connections make it even possi ble that users can now subscribe high quality services from data and software that reside solely on remote data centers.

Moving data into the cloud offers great convenience to users since they don't have to care about the complexities of direct hardware management. The pioneer of Cloud Com-puting vendors, Amazon Simple Storage Service (S3) and Amazon Elastic Compute Cloud (EC2) [1] are both well known examples. While these internet-based online services do provide huge amounts of storage space and customizable computing resources, this computing platform shift, however, is eliminating the responsibility of local machines for data maintenance at the same time. As a result, users are at the mercy of their cloud service providers for the availability and integrity of their data. Recent downtime of Amazon's S3 is such an example [2].

From the perspective of data security, which has always been an important aspect of quality of service, Cloud Com-puting inevitably poses new challenging security threats for number of reasons. Firstly, traditional cryptographic primitives for the purpose of data security protection can not be directly adopted due to the users' loss control of data under Cloud Computing. Therefore, verification of correct data storage in the cloud must be conducted without explicit knowledge of the whole data. Considering various kinds of data for each user stored in the cloud and the demand of long term continuous assurance of their data safety, the problem of verifying correctness of data storage in the cloud becomes even more challenging. Secondly, Cloud Computing is not just a third party data warehouse. The data stored in the cloud may be frequently updated by the users, including insertion, deletion, modification, appending, reordering, etc. To ens ure storage correctness under dynamic data update is hence of paramount importance. However, this dynamic feature also makes traditional integrity insurance techniques futile and entails new solutions. Last but not the least, the deployment of Cloud Computing is powered by data centers running in a simultaneous, cooperated and distributed manner. Individual user's data is redundantly stored in multiple physical loca-tions to further reduce the data integrity threats. Therefore, distributed protocols for storage correctness assurance will be of most importance in achieving a robust and secure cloud data storage system in the real world. However, such important area remains to be fully explored in the literature.

In this paper, we propose an effective and flexible distribut ed scheme with explicit dynamic data support to ensure the correctness of users' data in the cloud. We rely on erasure-correcting code in the file distribution preparation to prov ide redundancies and guarantee the data dependability. This con-struction drastically reduces the communication and storage overhead as compared to the traditional

replication-based file distribution techniques. By utilizing the homomorphic token with distributed verification of erasure-coded data, our sc heme achieves the storage correctness insurance as well as data error localization: whenever data corruption has been detected during the storage correctness verification, our scheme can almost guarantee the simultaneous localization of data errors, i.e., the identification of the misbehaving server(s)

Our work is among the first few ones in this field to consider distributed data storage in Cloud Computing. Our contribution can be summarized as the following three aspect.

1) Compared to many of its predecessors, which only provide binary results about the storage state across the distributed servers, the challenge-response protocol in our work further provides the localization of data error.

2) Unlike most prior works for ensuring remote data integrity, the new scheme supports secure and efficient dynamic opera-tions on data blocks, including: update, delete and append.

3) Extensive security and performance analysis shows that the proposed scheme is highly efficient and resilient agains t Byzantine failure, malicious data modification attack, and even server colluding attacks.

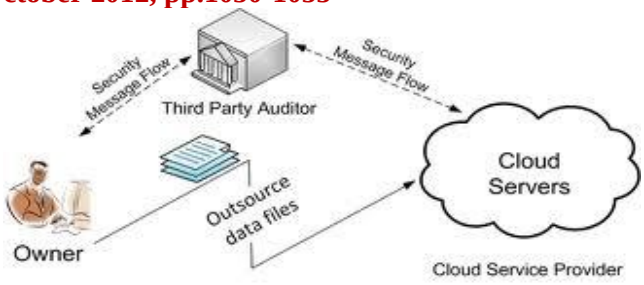
The rest of the paper is organized as follows. Section II introduces the system model, adversary model, our design goal and notations. Then we provide the detailed description of our scheme in Section III and IV. Section V gives the security analysis and performance evaluations, followed by Section VI which overviews the related work. Finally, Section VII gives the concluding remark of the whole paper.

II. PROBLEM STATEMENT

A. System Model

A representative network architecture for cloud data storage is illustrated in Figure 1. Three different network entities can be identified as follows:

- User: users, who have data to be stored in the cloud and rely on the cloud for data computation, consist of both individual consumers and organizations.
- Cloud Service Provider (CSP): a CSP, who has signif-icant resources and expertise in building and managing distributed cloud storage servers, owns and operates live Cloud Computing systems.
- Third Party Auditor (TPA): an optional TPA, who has expertise and capabilities that users may not have, is trusted to assess and expose risk of cloud storage services on behalf of the users upon request.



In cloud data storage, a user stores his data through a CSP into a set of cloud servers, which are running in a simulta-neous, cooperated and distributed manner. Data redundancy can be employed with technique of erasure-correcting code to further tolerate faults or server crash as user's data grows in size and importance. Thereafter, for application purposes, the user interacts with the cloud servers via CSP to access or retrieve his data. In some cases, the user may need to perform block level operations on his data. The most general forms of these operations we are considering are block update, delete, insert and append. As users no longer possess their data locally, it is of critical importance to assure users that their data are being correctly stored and maintained. That is, users should be equipped with security means so that they can make continuous correctness assurance of their stored data even without the existence of local copies. In case that users do not necessarily have the time, feasibility or resources to monitor their data, they can delegate the tasks to an optional trusted TPA of their respective choices.

B. Adversary Model

Security threats faced by cloud data storage can come from two different sources. On the one hand, a CSP can be self-interested, untrusted and possibly malicious. Not only does it desire to move data that has not been or is rarely accessed to a lower tier of storage than agreed for monetary reasons, but it may also attempt to hide a data loss incident due to management errors, Byzantine failures and so on. On the other hand, there may also exist an economically-motivated adversary, who has the capability to compromise a number of cloud data storage servers in different time intervals and subsequently is able to modify or delete users' data while remaining undetected by CSPs for a certain period. Specifically, we consider two types of adversary with differ ent levels of capability in this paper:

Weak Adversary: The adversary is interested in corrupting the user's data files stored on individual servers. Once a server is comprised, an adversary can pollute the original data files b y modifying or introducing its own fraudulent data to prevent the original data from being retrieved by the user.

Strong Adversary: This is the worst case scenario, in which we assume that the adversary can compromise all the storage servers so that he can intentionally modify the data files as long as they are internally

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

C. Design Goals

To ensure the security and dependability for cloud data storage under the aforementioned adversary model, we aim to design efficient mechanisms for dynamic data verification and operation and achieve the following goals: (1) Storage correctness: to ensure users that their data are indeed stored appropriately and kept intact all the time in the cloud. (2) Fast localization of data error: to effectively locate the mal-functioning server when data corruption has been detected. (3) Dynamic data support: to maintain the same level of storage correctness assurance even if users modify, delete or append their data files in the cloud. (4) Dependability: to enhance d ata availability against Byzantine failures, malicious data modifi-cation and server colluding attacks, i.e. minimizing the effect brought by data errors or server failures. (5) Lightweight: to enable users to perform storage correctness checks with minimum overhead.

D. Notation and Preliminaries

- **F** – the data file to be stored. We assume that **F** can be denoted as a matrix of **m** equal-sized data vectors, each consisting of **l** blocks. Data blocks are all well represented as elements in Galois Field **GF (2^p)** for **p = 8** or **16**.
  - **A** – The dispersal matrix used for Reed-Solomon coding.
  - **G** – The encoded file matrix, which includes a set of **n = m + k** vectors, each consisting of **l** blocks.
- **f_{key} (·)** – pseudorandom function (PRF), which is defined as **$f : \{0, 1\}^* \times key \rightarrow GF(2^p)$**.
- **φ_{key} (·)** – pseudorandom permutation (PRP), which is defined as $\varphi : \{0, 1\}^{\log_2(l)} \times key \rightarrow \{0, 1\}^{\log_2(l)}$.
  - **ver** – a version number bound with the index for individ-ual blocks, which records the times the block has been modified. Initially we assume **ver** is 0 for all data blocks.
  - s^{ver}_{ij} – the seed for PRF, which depends on the file name, block index **i**, the server position **j** as well as the optional block version number **ver**.

III. ENSURING CLOUD DATA STORAGE

In cloud data storage system, users store their data in the cloud and no longer possess the data locally. Thus, the correctness and availability of the data files being stored o n the distributed cloud servers must be guaranteed. One of the key issues is to effectively detect any unauthorized data modifi ca-tion and corruption, possibly due to server compromise and/or random Byzantine failures. Besides, in the distributed case when such inconsistencies are successfully detected, to fin d which server the data error lies in is also of great significan ce, since it can be the first step to fast recover the storage errors.

Subsequently, it is also shown how to derive a challenge-response protocol for verifying the storage correctness as well as identifying misbehaving servers. Finally, the procedure for file retrieval and error recovery based on erasure-correcti ng code is outlined.

A. File Distribution Preparation

It is well known that erasure-correcting code may be used to tolerate multiple failures in distributed storage systems.

Algorithm 1 Token Pre-computation

1: **procedure**
2: Choose parameters l, n and function f, φ;
3: Choose the number t of tokens;
4: Choose the number r of indices per verification;
5: Generate master key K_{prp} and challenge k_{chal};
6: **for** vector $G^{(j)}$, j ← 1, n **do**
7: **for** round i← 1, t **do**
8: Derive $\alpha_i = f_{k^{(i)}_{chal}}(j)$ and $k^{(i)}_{prp}$ from K_{PRP}.
9: Compute $v_i = \sum_{q=1}^{r} \alpha_i * G^{(j)}[\varphi_{k^{(i)}_{prp}}(q)]$
10: **end for**
11: **end for**
12: Store all the v_is locally.
13: **end procedure**

In cloud data storage, we rely on this technique to disperse the data file **F** redundantly across a set of **n = m + k** distributed servers. A **(m + k, k)** Reed-Solomon erasure-correcting code is used to create **k** redundancy parity vectors from **m** data vectors in such a way that the original **m** data vectors can be reconstructed from any **m** out of the **m + k** data and parity vectors. By placing each of the **m + k** vectors on a different server, the original data file can survive the failure of any **k** of the **m + k** servers without any data loss, with a space overhead of **k/m**. For support of efficient sequential I/O to the original file, our file layout is systematic, i.e., the unmodified **m** data file vectors together with **k** parity vectors is distributed across **m + k** different servers.

B. Challenge Token Precomputation

In order to achieve assurance of data storage correctness and data error localization simultaneously, our scheme entirely relies on the pre-computed verification tokens. The main ide a is as follows: before file distribution the user pre-compute s a certain number of short verification tokens on individual ve ctor $G^{(j)}$ ($j \in \{1, \ldots, n\}$), each token covering a random subset of data blocks. Later, when

the user wants to make sure the storage correctness for the data in the cloud, he challenges the cloud servers with a set of randomly generated block indices. Upon receiving challenge, each cloud server computes a short “signature” over the specified blocks and returns the m to the user. The values of these signatures should match the corresponding tokens pre-computed by the user. Meanwhile, as all servers operate over the same subset of the indices, the requested response values for integrity check must also be a valid codeword determined by secret matrix P.

Algorithm 2 Correctness Verification and Error Localization

1) **procedure** CHALLENGE(i)
2) Recompute $\alpha_i = f_{k\mathbf{chal}}(i)$ and $k_{prp}^{(i)}$ from K_{PRP};
3) Send $\{\alpha_i, k_{prp}^{(i)}\}$ to all the cloud servers;
4) Receive from servers:
$\{R_i^{(j)} = \sum_{q=1}^{r} \alpha_i^q * G^{(j)}[\varphi_{k_{prp}^{(i)}}(q)] | 1 \leq j \leq n\}$
5: **for** ($j \leftarrow m + 1, n$) **do**
6: $R^{(j)} \leftarrow R^{(j)} - \sum_{q=1}^{r} f_{k\mathbf{j}}(s_{I\mathbf{q},j}) \cdot \alpha_i^q$, $I_q = \varphi_{k_{prp}^{(\mathbf{i})}}(q)$
7: **end for**
8: **if** $((R_i^{(1)}, \ldots, R_i^{(m)}) \cdot P == (R_i^{(m+1)}, \ldots, R_i^{(n)}))$ **then**
9: Accept and ready for the next challenge.
10: **else**
11: **for** ($j \leftarrow 1, n$) **do**
12: **if** ($R_i^{(j)} != v_i^{(j)}$) **then**
13: **return** server j is misbehaving.
14: **end if**
15: **end for**
16: **end if**
17: **end procedure**

D. File Retrieval and Error Recovery

Since our layout of file matrix is systematic, the user can reconstruct the original file by downloading the data vector s assurance is a probabilistic one. However, by choosing system param-eters (e.g., r, l, t) appropriately and conducting enough times of verification, we can guarantee the successful file retriev al with high probability. On the other hand, whenever the data corruption is detected, the comparison of pre-computed tokens and received response values can guarantee the identificati on of misbehaving server(s), again with high probability, which will be discussed shortly. Therefore, the user can always ask servers to send back blocks of the r rows specified in the challenge and regenerate the correct blocks by erasure correction, shown in Algorithm 3, as long as there are at most k misbehaving servers are identified. The newly recovered blocks can then be redistributed to the misbehaving servers to maintain the correctness of storage.

IV. PROVIDING DYNAMIC DATA OPERATION SUPPORT

So far, we assumed that **F** represents static or archived data. This model may fit some application scenarios, such as libraries and scientific datasets. However, in cloud data storage, there are many potential scenarios where data stored in the cloud is dynamic, like electronic documents, photos, or log files etc. Therefore, it is crucial to consider the dynamic case, where a user may wish to perform various block-level

Algorithm 3 Error Recovery

1: **procedure**
% Assume the block corruptions have been detected among
% the specified r rows;
% Assume s ≤ k servers have been identified misbehaving
2: Download r rows of blocks from servers;
3: Treat s servers as erasures and recover the blocks.
4: Resend the recovered blocks to corresponding servers.
5: **end procedure**

operations of update, delete and append to modify the data fil e while maintaining the storage correctness assurance.
In this section, we will show how our scheme can explicitly and efficiently handle dynamic data operations for cloud data storage.

A. Update Operation

In cloud data storage, sometimes the user may need to modify some data block(s) stored in the cloud, from its current value f_{ij} to a new one, $f_{ij} + f_{ij}$. We refer this operation as data update. Due to the linear property of Reed-Solomon code, a user can perform the update operation and generate the updated parity blocks by using f_{ij} only, without involving any other unchanged blocks.

B. Delete Operation

Sometimes, after being stored in the cloud, certain data

blocks may need to be deleted. The delete operation we are considering is a general one, in which user replaces the data block with zero or some special reserved data symbol. From this point of view, the delete operation is actually a special case of the data update operation, where the original data blocks can be replaced with zeros or some predetermined special blocks. Therefore, we can rely on the update procedure to support delete operation, i.e., by setting f_{ij} in F to be $- f_{ij}$. Also, all the affected tokens have to be modified and the updated parity information has to be blinded using the same method specified in update operation.

C. Append Operation

In some cases, the user may want to increase the size of his stored data by adding blocks at the end of the data file, which we refer as data append. We anticipate that the most frequent append operation in cloud data storage is bulk append, in which the user needs to upload a large number of blocks (not a single block) at one time.

D. Insert Operation

An insert operation to the data file refers to an append operation at the desired index position while maintaining the same data block structure for the whole data file, i.e., inser ting a block F [j] corresponds to shifting all blocks starting with index j + 1 by one slot. An insert operation may affect many rows in the logical data file matrix **F**, and a substantial number of computations are required to renumber all the subsequent blocks as well as re-compute the challenge-response tokens. Therefore, an efficient insert operation is difficult to supp ort and thus we leave it for our future work.

V. SECURITY ANALYSIS AND PERFORMANCE EVALUATION

In this section, we analyze our proposed scheme in terms of security and efficiency. Our security analysis focuses on the adversary model defined in Section II. We also evaluate the efficiency of our scheme via implementation of both file distribution preparation and verification token precomputation.

A. Security Strength Against Weak Adversary

Detection Probability against data modification: In our in each correctness verification for the calculation of requested token. We will show that this "sampling" strategy on selecte d rows instead of all can greatly reduce scheme, servers are required to operate on specified rows the computational overhead on the server, while maintaining the detection of the data corruption with high probability.

B. Security Strength Against Strong Adversary

In this section, we analyze the security strength of our schemes against server colluding attack and explain why blind-ing the parity blocks can help improve the security strength of our proposed scheme.

Recall that in the file distribution preparation, the redun-dancy parity vectors are calculated via multiplying the file matrix **F** by **P**, where **P** is the secret parity generation matrix we later rely on for storage correctness assurance. If we disperse all the generated vectors directly after token precomputation, i.e., without blinding, malicious servers that collaborate can reconstruct the secret P matrix easily: they can pick blocks from the same rows among the data and parity vectors to establish a set of $m \cdot k$ linear equations and solve for the $m \cdot k$ entries of the parit generation matrix **P**. Once they have the knowledge of **P**, those malicious servers can consequently modify any part of the data blocks and calculate the corresponding parity blocks, and vice versa, making their codeword relationship always consistent. Therefore, our stor-age correctness challenge scheme would be undermined—even if those modified blocks are covered by the specified rows, the storage correctness check equation would always hold.

C. Performance Evaluation

1)File Distribution Preparation: We implemented the gen-eration of parity vectors for our scheme under field GF (2^8). Our experiment is conducted using C on a system with an Intel Core 2 processor running at 1.86 GHz, 2048 MB of RAM, and a 7200 RPM Western Digital 250 GB Serial ATA drive with an 8 MB buffer. We consider two sets of different parameters for the (m + k, m) Reed-Solomon encoding. Table I shows the average encoding cost over 10 trials for an 8 GB file.

2) Challenge Token Pre-computation: Although in our scheme the number of verification token t is a fixed priori determined before file distribution, we can overcome this issue by choosing sufficient large t in practice.

VI. RELATED WORK

Juels et al. [3] described a formal "proof of retrievability " (POR) model for ensuring the remote data integrity. Their scheme combines spot-cheking and error-correcting code to ensure both possession and retrievability of files on archiv e service systems. Shacham et al. [4] built on this model and constructed a random linear function based homomorphic authenticator which enables unlimited number of queries and requires less communication overhead. Bowers et al. [5] pro-posed an improved framework for POR protocols that general-izes both Juels and Shacham's work. Later in their subsequent work, Bowers et al. [10] extended POR model to distributed systems. However, all these schemes are focusing on static data. The effectiveness of their schemes rests primarily on the preprocessing steps

that the user conducts before outsourcing the data file F . Any change to the contents of F, even few bits, must propagate through the error-correcting code, thus introducing significant computation and communicatio n complexity.

VII. CONCLUSION

In this paper, we investigated the problem of data security in cloud data storage, which is essentially a distribute storage system. To ensure the correctness of users' data in cloud data storage, we proposed an effective and flexible distribu ted scheme with explicit dynamic data support, including block update, delete, and append. We rely on erasure-correcting code in the file distribution preparation to provide redundancy parity vectors and guarantee the data dependability. By utilizing the homomorphic token with distributed verification of erasure - coded data, our scheme achieves the integration of storage cor-rectness insurance and data error localization, i.e., whenever data corruption has been detected during the storage correct-ness verification across the distributed servers, we can alm ost guarantee the simultaneous identification of the misbehavi ng server(s). Through detailed security and performance analysis, we show that our scheme is highly efficient and resilient to Byzantine failure, malicious data modification attack, and even server colluding attacks.

ACKNOWLEDGEMENT

This work was supported in part by the US National Science Foundation under grant CNS-0831963, CNS-0626601, CNS-0716306, and CNS-0831628.

REFERENCE

[1] Amazon.com, “Amazon Web Services (AWS),” Online at http ://aws. amazon.com, 2008.

[2] N. Gohring, “Amazon's S3 down for several hours,” Online at http://www.pcworld.com/businesscenter/arti cle/142549/amazons s3 down for several hours.html, 2008.

A. Juels and J. Burton S. Kaliski, “PORs: Proofs of Retrie vability for Large Files,” *Proc. of CCS '07*, pp. 584–597, 2007.

[4] H. Shacham and B. Waters, “Compact Proofs of Retrievabil ity,” *Proc. of Asiacrypt '08*, Dec. 2008.

[5] K. D. Bowers, A. Juels, and A. Oprea, “Proofs of Retrievab ility: Theory and Implementation,” Cryptology ePrint Archive, Report 20 08/175, 2008, http://eprint.iacr.org/.

[6] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, “Provable Data Possession at Untrusted Stores, ” *Proc. of CCS '07*, pp. 598–609, 2007.

[7] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik, “S calable and Efficient Provable Data Possession,” *Proc. of SecureComm '08*, pp. 1– 10, 2008.

[8] T. S. J. Schwarz and E. L. Miller, “Store, Forget, and Chec k: Using Algebraic Signatures to Check Remotely Administered Storage,” *Proc. of ICDCS '06*, pp. 12–12, 2006.

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

[10] K. D. Bowers, A. Juels, and A. Oprea, “HAIL: A High-Avail ability and Integrity Layer for Cloud Storage,” Cryptology ePrint Arch ive, Report 2008/489, 2008, http://eprint.iacr.org/.

[11] L. Carter and M. Wegman, “Universal Hash Functions,” *Journal of Computer and System Sciences*, vol. 18, no. 2, pp. 143–154, 1979.

[12] J. Hendricks, G. Ganger, and M. Reiter, “Verifying Dist ributed Erasure-coded Data,” *Proc. 26th ACM Symposium on Principles of Distributed Computing*, pp. 139–146, 2007.

[13] J. S. Plank and Y. Ding, “Note: Correction to the 1997 Tut orial on Reed-Solomon Coding,” University of Tennessee, Tech. Rep. CS-03-504, 2003.

[14] Q. Wang, K. Ren, W. Lou, and Y. Zhang, “Dependable and Sec ure Sensor Data Storage with Dynamic Integrity Assurance,” *Proc. of IEEE INFOCOM*, 2009.

[15] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, “MR-PDP : Multiple-Replica Provable Data Possession,” *Proc. of ICDCS '08*, pp. 411–420, 2008.

[16] D. L. G. Filho and P. S. L. M. Barreto, “Demonstrating Dat a Possession and Uncheatable Data Transfer,” Cryptology ePrint Archive , Report 2006/150, 2006, http://eprint.iacr.org/.

[17] M. A. Shah, M. Baker, J. C. Mogul, and R. Swaminathan, “Au diting to Keep Online Storage Services Honest,” *Proc. 11th USENIX Workshop on Hot Topics in Operating Sy*