Verification of Storage Integraty using PDP Technique

J. Shilpa [α] & Prof. M. Madhavi [σ]

Abstract - **Cloud computing is mostly used for highly scalable applications which are very catchable now a day in the world of Internet as on primary needs. The important feature provided to the customers' data is done in unknown systems will do all different transmissions remotely. Using the transmission through remote systems of cloud computing makes users' scares of their data is that secured or not specially in some particular categories like online transmissions and health. These are the threats that create a significant barrier in the cloud computing services. To tackle this crisis, in this paper, we explored a novel highly decentralized information accountability framework. That maintains complete history of the usage of the registered user's data in the cloud. In this way, we explored an object-centered approach that creates enclosing our details of logging history combining with the users' data. We referrers couple of provably-secure PDP schemes which are most accurate when compared to old ones, not only that when compared with schemes that results less efficient than our proposed. In particular, the overhead at the server is low as opposed to linear in the size of the data. Researches using our implementation verify the practicality of PDP and reveal that the performance of PDP is bounded by disk I/O and not by cryptographic computation.**

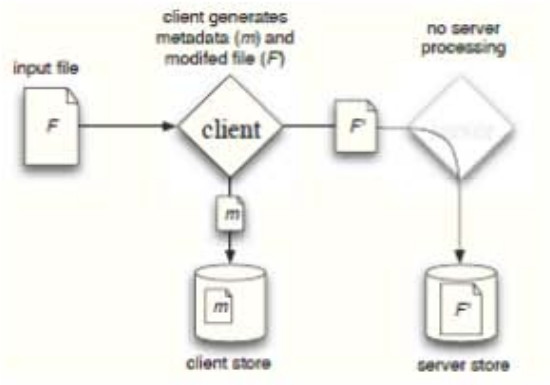
I. Introduction

However, archival storage requires guarantees about the authenticity of data on storage, namely that storage servers possess data. It is insufficient to detect that data have been modified or deleted when accessing the data, because it may be too late to recover lost or damaged data. Archival storage servers retain tremendous amounts of data, little of which are accessed. They also hold data for long periods of time during which there may be exposure to data loss from administration errors as the physical implementation of storage evolves, e.g., backup and restore, data migration to new systems, and changing memberships in peer-to-peer systems. Archival network storage presents unique performance demands.

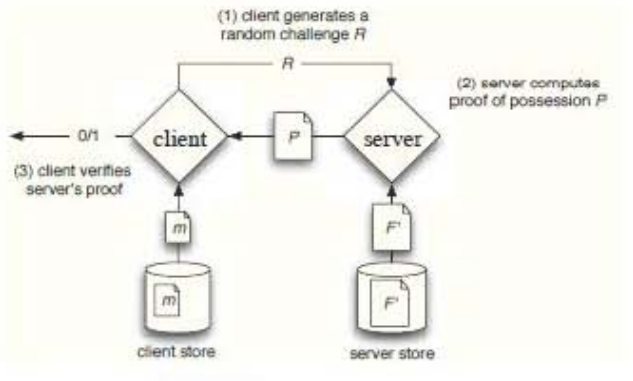
Given that file data are large and are stored at remote sites, accessing an entire file is expensive in I/O costs to the storage server and in transmitting the file across a network. Reading an entire archive, even periodically, greatly limits the scalability of network stores. (The growth in storage capacity has far outstripped the growth in storage access times and bandwidth [44]). Furthermore, I/O incurred to establish data possession interferes with on-demand bandwidth to store and retrieve data. We conclude that clients need to be able to verify that a server has retained file data without retrieving the data from the server and without having the server access the entire file. Previous solutions do not meet these requirements for proving data possession. Some schemes [20] provide a weaker guarantee by enforcing storage complexity: The server has to store an amount of data at least as large as the client's data, but not necessarily the same exact data. Moreover, all previous techniques require the server to access the entire file, which is not feasible when dealing with large amounts of data. We define a model for provable data possession (PDP) that provides probabilistic proof that a third party stores a file. The model is unique in that it allows the server to access small portions of the file in generating the proof; all other techniques must access the entire file. Within this model, we give the first provably-secure scheme for remote data checking. The client stores a small O (1) amount of metadata to verify the server's proof. Also, the scheme uses O (1) bandwidth1. The challenge and the response are each slightly more than 1 Kilobit. We also present a more efficient version of this scheme that proves data possession using a single modular exponentiation at the server, even though it provides a weaker guarantee.

Both schemes use Homomorphic verifiable tags. Because of the Homomorphic property, tags computed for multiple file blocks can be combined into a single value. The client pre-computes tags for each block of a file and then stores the file and its tags with a server. At a later time, the client can verify that the server possesses the file by generating a random challenge against a randomly selected set of file blocks. Using the queried blocks and their corresponding tags, the server generates a proof of possession. The client is thus convinced of data possession, without actually having to retrieve file blocks. The efficient PDP scheme is the fundamental construct underlying an archival introspection system that we are developing for the long-term preservation of Astronomy data.

Author α : (M.Tech), CSE Dept, ASRA Hyderabad
E-mail : javvajishilpa@gmail.com
Author σ : (M.Tech), CSE, Associate Professor, ASRA Hyderabad.
E-mail : madhavi_3101@yahoo.co.in



(a) Pre-Process and Store



(b) Verify Server Possession

We are taking possession of multi-terabyte Astronomy databases at a University library in order to preserve the information long after the research projects and instruments used to collect the data are gone. The database will be replicated at multiple sites. Sites include resource-sharing partners that exchange storage capacity to achieve reliability and scale. As such, the system is subject to freeloading in which partners attempt to use storage resources and contribute none of their own [20]. The location and physical implementation of these replicas are managed independently by each partner and will evolve over time. Partners may even out source storage to third-party storage server providers [23].

Efficient PDP schemes will ensure that the computational requirements of remote data checking do not unduly burden the remote storage sites. We implemented our more efficient scheme (E-PDP) and two other remote data checking protocols and evaluated their performance. Experiments show that probabilistic possession guarantees make it practical to verify possession of large data sets. With sampling, E-PDP verifies a 64MB file in about 0.4 seconds as compared to 1.8 seconds without sampling. Further, I/O bounds the performance of EPDP; it generates proofs as quickly as the disk produces data. Finally, E-PDP is 185 times faster than the previous secure protocol on 768 KB files.

II. Systems Overview

a) Provable Data Possession (PDP)

We describe a framework for provable data possession. This provides background for related work and for the specific description of our schemes. A PDP protocol (Fig. 1) checks that an outsourced storage site retains a file, which consists of a collection of n blocks. The client C (data owner) pre-processes the file, generating a piece of metadata that is stored locally, transmits the file to the server S, and may delete its local copy. The server stores the file and responds to challenges issued by the client. Storage at the server is in (n) and storage at the client is in O (1), conforming to our notion of an outsourced storage relationship.

i. Threat Model

The server S must answer challenges from the client C; failure to do so represents a data loss. However, the server is not trusted: Even though the file is totally or partially missing, the server may try to convince the client that it possesses the file. The server's motivation for misbehavior can that has not been or is rarely accessed (for monetary reasons), or be diverse and includes reclaiming storage by discarding data hiding a data loss incident (due to management errors, hardware failure, compromise by outside or inside attacks etc). The goal of a PDP scheme that achieves probabilistic proof of data possession is to detect server misbehavior when the server has deleted a fraction of the file.

III. Evaluation

a) Probabilistic Framework

Our PDP schemes allow the server to prove possession of select blocks of F. This "sampling" ability greatly reduces the workload on the server, while still achieving detection of server misbehavior with high probability. We now analyze the probabilistic guarantees offered by a scheme that supports block sampling. Assume S deletes t blocks out of the n-block file F. Let c be the number of different blocks for which C asks proof in a challenge. Let X be a discrete random variable that is defined to be the number of blocks chosen by C that match the blocks deleted by S. We compute PX, the probability that at least one of the blocks picked by C matches one of the blocks deleted by S. We have:

$$P_X = P\{X \geq 1\} = 1 - P\{X = 0\} =$$

$$1 - \frac{n-t}{n} \cdot \frac{n-1-t}{n-1} \cdot \frac{n-2-t}{n-2} \cdot \ldots \cdot \frac{n-c+1-t}{n-c+1}.$$

Since $\frac{n-i-t}{n-i} \geq \frac{n-i-1-t}{n-i-1}$, it follows that:

$$1-\left(\frac{n-t}{n}\right)^c \leq P_X \leq 1-\left(\frac{n-c+1-t}{n-c+1}\right)^c.$$

PX indicates the probability that, if S deletes t blocks of the file, then C will detect server misbehavior after a challenge in which it asks proof for c blocks. Fig. 3 plots PX for different values of n, t, c. interestingly, when t is a fraction of the file, C can detect server misbehavior with a certain probability by asking proof for a constant amount of blocks, independently of the total number of file blocks: e.g., if t = 1% of n, then C asks for 460 blocks and 300 blocks in order to achieve PX of at least 99% and 95%, respectively.

b) Implementation and Experimental Results

We measure the performance of E-PDP and the benefits of sampling based on our implementation of E-PDP in Linux. As a basis for comparison, we have also implemented the scheme of Deswarte et al. [17] and Filho et al. [19] (B-PDP), and the more efficient scheme in [20] (MHT-SC) suggested by David Wagner (these schemes are described in Appendix B). All experiments were conducted on an Intel 2.8 GHz Pentium IV system with a 512 KB cache, an 800 MHz EPCI bus, and 1024 MB of RAM. The system runs Red Hat Linux 9, kernel version 2.4.22. Algorithms use the crypto library of Open SSL version 0.9.8b with a modulus N of size 1024 bits and files have 4KB blocks. Experiments that measure disk I/O performance do so by storing files on an ext3 file system on a Seagate Barracuda 7200.7 (ST380011A) 80GB Ultra ATA/100 drives. All experimental results represent the mean of 20 trials. Because results varied little across trials, we do not present confidence intervals.

i. *Server Computation*

The next experiments look at the worst-case performance of generating a proof of possession, which is useful for planning purposes to allow the server to allocate enough resources. For E-PDP, this means sampling every block in the file, while for MHT-SC this means computing the entire hash tree. We compare the computation complexity of E-PDP with other algorithms, which do not support sampling. All schemes perform an equivalent number of disk and memory accesses.

In step 3 of the Gen Proof algorithm of S-PDP, S has two ways of computing ρ: Either sum the values ajmij (as integers) and then exponentiation to this sum or exponentiation gs to each value ajmij and then multiply all values. We observed that the former choice takes considerable less time, as it only involves one exponentiation to a (|mi| + ℓ + log2(c))-bit number, as opposed to c exponentiations to a (|mi| + ℓ)-bit number (typically, ℓ = 160).

ii. *Pre-Processing*

In preparing a file for outsourced storage, the client generates its local metadata. In this experiment, we measure the processor time for metadata generation only. This does not include the I/O time to load data to the client or store metadata to disk, nor does it include the time to transfer the file to the server. Fig. 5(b) shows the pre-processing time as a function of file size for BPDP, MHT-SC and E-PDP. E-PDP exhibits slower preprocessing performance. The costs grow linearly with the file size at 162 KB/s. E-PDP performs an exponentiation on every block of the file in order to create the per block tags. For MHTSC, preprocessing performance mirrors challenge performance, because both protocol steps perform the same computation. It generates data at about 433 KB/s on average.

IV. Related Work

Deswarte et al. [17] and Filho et al. [19] provide techniques to verify that a remote server stores a file using RSA-based hash functions. Unlike other hash-based approaches, it allows a client to perform multiple challenges using the same metadata. In this protocol, communication and client storage complexity are both O(1). The limitation of the algorithm lies in the computational complexity at the server, which must exponentiation the entire file, accessing the entire file's blocks. Further, RSA over the entire file is extremely slow — 20 seconds per Megabyte for 1024-bit keys on a 3.0 GHz processor [19]. In fact, these limitations led us to study algorithms that allowed for sub-file access (sampling). We implement this protocol for comparison with our PDP scheme and refer to it as B-PDP (basic PDP). A description of B-PDP is provided in Appendix B. Shah et al. [42] use a similar technique for third-party auditing of data stored at online service providers and put forth some of the challenges associated with auditing online storage services. Schwarz and Miller [40] propose a scheme that allows a client to verify the storage of m/n erasure-coded data across multiple sites even if sites collude. The data possession guarantee is achieved using a special construct, called an "algebraic signature": A function that fingerprints a block and has the property that the signature of the parity block equals the parity of the signatures of the data blocks. The parameters of the scheme limit its applicability: The file access and computation complexity at the server and the communication complexity are all linear in the number of file blocks (n) per challenge. Additionally, the security of the scheme is not proven and remains in question.

a) Command Line-Based Data Processing

The systems described in this section are implemented by monitoring a command line interpreter which allows them to passively capture and store the information necessary to assemble a retrospective view

on data processing. As defined by Merriam-Webster [2001], an audit trail is a record of a sequence of events (as actions performed by a computer) from which a history can be reconstructed, and thus serves as a form of lineage. Becker and Chambers [1988] describe a system for auditing data analyses steps for a particular implementation of S, a language and interactive environment for statistical analysis and display. Their intention is to provide a tool for a user to investigate the dependencies among steps following an exploratory S analysis session. User-entered statements evaluated by S, including the associated creation and modification of data objects resulting from those statements, are dynamically recorded in an audit file. An audit facility parses the audit file into a linked list structure, which it then uses to respond to ad hoc queries and generate custom so called audit plots to display analysis step dependencies. The prototype audit facility Becker and Chambers describe has not been implemented in contemporary versions of the S system such as S-Plus [Insightful Corporation 2003].

b) Script- and Program-Based Data Processing

The systems described in this section assemble a retrospective view on processing using information encoded directly in user-supplied scripts or programs. ESSW [Frew and Bose 2001] captures lineage metadata for objects involved in scientific processing performed with application-specific scripts as well as general scripting languages such as Perl. ESSW uses custom application programming interface (API) commands within Perl wrapper scripts—code that circumscribes the functions, algorithms, or other data transformations of interest—to construct lineage. The lineage of an item is queried through a web application, and results are displayed diagrammatically using the Webdot Web service interface included with the Graphviz set of graphing tools [AT&T 2001].

c) WFMS-Based Data Processing

Extending some of the concepts in GOOSE, the Geo-Opera extension of the OPERA kernel [Alonso and Hagen 1997b; Alonso et al. 1998] provides a management system for distributed geoprocessing that incorporates elements of workflow management, transaction processing, and lineage tracking for an Earth Science example of hydrologic modeling. Data files and transformations used by hydrologic models reside outside of the system. Once transformations are registered in Geo-Opera, they are tracked as task objects internal to the system. Lineage relationships between objects are established by defining the control flow between internal task objects and data. When data is located outside the system, it is registered in the system as an external object. Each external object includes a set of system-maintained attributes supporting automated versioning, change propagation, and lineage recording.

d) Query-Based Data Processing

Brown and Stonebraker [1995] and Woodruff and Stonebraker [1997] propose a method for providing detailed or finegrained lineage for scientific processing applications. A goal of their research is delivering to scientists, through data lineage, the ability to investigate the source of faulty or anomalous data sets and the ability to determine those derived data sets affected by faulty or anomalous inputs or algorithms. Specifically, Woodruff and Stonebraker [1997] address the problem of recovering the origins of single elements in large arrays of data that have undergone a series of transformations. Creating individual metadata entries to assist with such a task would require prohibitive effort and storage size.

e) Service-Based Data Processing

The Chimera Virtual Data System (VDS) matches the scope and ambition of the Grid, targeting invocations of data transformations in a "distributed, multi-user, multi-institutional environment" [Foster et al. 2003]. Chimera features a language, the Virtual Data Language (VDL), for defining and manipulating data derivation procedures which are stored in a Virtual Data Catalog (VDC). The VDL serves as a general wrapper for program execution, capable of accommodating Grid request planning. The language is also used to query the VDC to discover or invoke the lineage or pipeline of computations that created a particular data object. Chimera is described as a virtual data prototype because it is capable of creating a directed acyclic graph (DAG) of distributed computations that can be submitted to the Grid to regenerate a given data object.

V. Conclusion

We proposed innovative approaches for automatically logging any access to the data in the cloud together with an auditing mechanism. Our approach allows the data owner to not only audit his content but also enforce strong back-end protection if needed. Moreover, one of the main features of our work is that it enables the data owner to audit even those copies of its data that were made without his knowledge.

In the future, we plan to refine our approach to verify the integrity of the JRE and the authentication of JARs [20]. For example, we will investigate whether it is possible to leverage the notion of a secure JVM [18] being developed by IBM. This research is aimed at providing software tamper resistance to Java applications. In the long term, we plan to design a comprehensive and more generic object-oriented approach to facilitate autonomous protection of traveling content. We would like to support a variety of security policies, like indexing policies for text files, usage control for executables, and generic accountability and provenance controls.

References Références Referencias

1. P. Ammann and S. Jajodia, "Distributed Timestamp Generation in Planar Lattice Networks," ACM Trans. Computer Systems, vol. 11, pp. 205-225, Aug. 1993.
2. G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable Data Possession at Untrusted Stores," Proc. ACM Conf. Computer and Comm. Security, pp. 598- 609, 2007.
3. E. Barka and A. Lakas, "Integrating Usage Control with SIP-Based Communications," J. Computer Systems, Networks, and Comm., vol. 2008, pp. 1-8, 2008.
4. D. Boneh and M.K. Franklin, "Identity-Based Encryption from the Weil Pairing," Proc. Int'l Cryptology Conf. Advances in Cryptology, pp. 213-229, 2001.
5. R. Bose and J. Frew, "Lineage Retrieval for Scientific Data Processing: A Survey," ACM Computing Surveys, vol. 37, pp. 1- 28, Mar. 2005.
6. P. Buneman, A. Chapman, and J. Cheney, "Provenance Management in Curated Databases," Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '06), pp. 539-550, 2006.
7. B. Chun and A.C. Bavier, "Decentralized Trust Management and Accountability in Federated Systems," Proc. Ann. Hawaii Int'l Conf. System Sciences (HICSS), 2004.
8. OASIS Security Services Technical Committee, "Security Assertion Markup Language (saml) 2.0," http://www.oasis-open.org/committees/tchome.php ?wg abbrev=security, 2012.
9. R. Corin, S. Etalle, J.I. den Hartog, G. Lenzini, and I. Staicu, "A Logic for Auditing Accountability in Decentralized Systems," Proc. IFIP TC1 WG1.7 Workshop Formal Aspects in Security and Trust, pp. 187-201, 2005.
10. B. Crispo and G. Ruffo, "Reasoning about Accountability within Delegation," Proc. Third Int'l Conf. Information and Comm. Security (ICICS), pp. 251-260, 2001.
11. Y. Chen et al., "Oblivious Hashing: A Stealthy Software Integrity Verification Primitive," Proc. Int'l Workshop Information Hiding, F. Petitcolas, ed., pp. 400-414, 2003.
12. S. Etalle and W.H. Winsborough, "A Posteriori Compliance Control," SACMAT '07: Proc. 12th ACM Symp. Access Control Models and Technologies, pp. 11-20, 2007.
13. X. Feng, Z. Ni, Z. Shao, and Y. Guo, "An Open Framework for Foundational Proof-Carrying Code," Proc. ACM SIGPLAN Int'l Workshop Types in Languages Design and Implementation, pp. 67-78, 2007.
14. Flickr, http://www.flickr.com/, 2012.
15. R. Hasan, R. Sion, and M. Winslett, "The Case of the Fake Picasso: Preventing History Forgery with Secure Provenance," Proc. Seventh Conf. File and Storage Technologies, pp. 1-14, 2009.
16. J. Hightower and G. Borriello, "Location Systems for Ubiquitous Computing," Computer, vol. 34, no. 8, pp. 57-66, Aug. 2001.
17. J.W. Holford, W.J. Caelli, and A.W. Rhodes, "Using Self- Defending Objects to Develop Security Aware Applications in Java," Proc. 27th Australasian Conf. Computer Science, vol. 26, pp. 341-349, 2004.
18. Trusted Java Virtual Machine IBM, http://www.almaden.ibm.com/cs/projects/jvm/, 2012.
19. P.T. Jaeger, J. Lin, and J.M. Grimes, "Cloud Computing and Information Policy: Computing in a Policy Cloud?," J. Information Technology and Politics, vol. 5, no. 3, pp. 269-283, 2009.
20. R. Jagadeesan, A. Jeffrey, C. Pitcher, and J. Riely, "Towards a Theory of Accountability and Audit," Proc. 14th European Conf. Research in Computer Security (ESORICS), pp. 152-167, 2009.

This page is intentionally left blank