Cloud Computing: A Responsibility data sharing in the cloud computing

K.Phanindhar[#1], Ch.Suresh[*2] Asst Professor

***Abstract*—** Cloud computing provides highly efficient services to be easily accessed or used over the Internet on as needed basis. An important feature of the cloud services is that users' data are usually processed remotely in unknown machines that users do not own or operate. This convenience brought by this new emerging technology, users' fears of losing control of their own data (particularly, financial and health data) can become a significant barrier to the wide adoption of cloud services. To address this problem, in this paper, we propose a novel highly decentralized information accountability framework to keep track of the actual usage of the users' data in the cloud. In particular, we propose an object centred approach that enables enclosing our logging mechanism together with users' data and policies. We leverage the JAR programmable capabilities to both create a dynamic and travelling object, and to ensure that any access to users' data will trigger authentication and automated logging local to the JARs. To strengthen user's control, we also provide distributed auditing mechanisms. We provide extensive experimental studies that demonstrate the efficiency and effectiveness of the proposed approaches.

***Keywords*—** Cloud computing, accountability, data sharing.

I. INTRODUCTION

Cloud computing presents a new way to supplement the current consumption and delivery model for IT services based on the Internet, by providing for dynamically scalable and often virtualized resources as a service over the Internet. Moreover, users may not know the machines which actually process and host their data. While enjoying the convenience brought by this new technology, users also start worrying about losing control of their own data. The data processed on clouds are often outsourced, leading to a number of issues related to accountability, including the handling of personally identifiable information. Such fears are becoming a significant barrier to the wide adoption of cloud services.

To allay users' concerns, it is essential to provide an effective mechanism for users to monitor the usage of their data in the cloud. For example, users need to be able to ensure that their data are handled according to the service-level agreements made at the time they sign on for services in the cloud. Conventional access control approaches developed for closed domains such as databases and operating systems, or approaches using a centralized server in distributed environments, are not suitable, due to the following features characterizing cloud environments. First, data handling can be outsourced by the direct cloud service provider (CSP) to other entities in the cloud and theses entities can also delegate the tasks to others, and so on. Second, entities are allowed to join and leave the cloud in a flexible manner. As a result, data handling in the cloud goes through a complex and dynamic hierarchical service chain which does not exist in conventional environments.

To overcome the above problems, we propose a novel approach, namely Cloud Information Accountability (CIA) framework, based on the notion of information accountability [44]. Unlike privacy protection technologies which are built on the hide-it-or-lose-it perspective, information account-ability focuses on keeping the data usage transparent and trackable. Our proposed CIA framework provides end-to-end accountability in a highly distributed fashion. One of the main innovative features of the CIA framework lies in its ability of maintaining lightweight and powerful account-ability that combines aspects of access control, usage control and authentication. By means of the CIA, data owners can track not only whether or not the service-level agreements are being honoured, but also enforce access and usage control rules as needed. Associated with the accountability feature, we also develop two distinct modes for auditing: push mode and pull mode. The push mode refers to logs being periodically sent to the data owner or stakeholder while the pull mode refers to an alternative approach whereby the user (or another authorized party) can retrieve the logs as needed.

The design of CIA frame work presents substantial challenges, including uniquely identifying CSPs, ensuring the reliability of the log, adapting to a highly decentralized infrastructure, etc. Our basic approach toward addressing these issues is to leverage and extend the programmable capability of JAR (Java ARchives) files to automatically log the usage of the users' data by any entity in the cloud. Users will send their data along with any policies such as access control policies and logging policies that they want to enforce, enclosed in JAR files, to cloud service providers. Any access to the data will trigger an automated and authenticated logging mechanism local to the JARs. We refer to this type of enforcement as "strong binding" since the policies and the logging mechanism travel with the data. This strong binding exists even when copies of the JARs are created; thus, the user will have control over his data at any location. Such decentralized logging mechanism meets the dynamic nature of the cloud but also imposes challenges on ensuring the integrity of the logging. To cope with this issue, we provide the JARs with a central point of contact which forms a link between them and the user. It records the error correction information sent by the JARs, which allows it to

monitor the loss of any logs from any of the JARs. Moreover, if a JAR is not able to contact its central point, any access to its enclosed data will be denied.

Currently, we focus on image files since images represent a very common content type for end users and organizations (as is proven by the popularity of Flickr) and are increasingly hosted in the cloud as part of the storage services offered by the utility computing paradigm featured by cloud computing. Further, images often reveal social and personal habits of users, or are used for archiving important files from organizations. In addition, our approach can handle personal identifiable information provided they are stored as image files (they contain an image of any textual content, for example, the SSN stored as a .jpg file).

In summary, our main contributions are as follows:

. We propose a novel automatic and enforceable logging mechanism in the cloud. To our knowledge, this is the first time a systematic approach to data accountability through the novel usage of JAR files is proposed.

. Our proposed architecture is platform independent and highly decentralized, in that it does not require any dedicated authentication or storage system in place.

. We go beyond traditional access control in that we provide a certain degree of usage control for the protected data after these are delivered to the receiver.

II. PROBLEM STATEMENT

We begin this section by considering an illustrative example which serves as the basis of our problem statement and will be used throughout the paper to demonstrate the main features of our system.

Example 1. Joy, a professional photographer, plans to sell her photographs by using the MyWeb Cloud Services. For her business in the cloud, she has the following requirements:

Her photographs are downloaded only by users who have paid for her services.

. Potential buyers are allowed to view her pictures first before they make the payment to obtain the download right.

. Due to the nature of some of her works, only users from certain countries can view or download some sets of photographs.

. For some of her works, users are allowed to only view them for a limited time, so that the users cannot reproduce her work easily.

. In case any dispute arises with a client, she wants to have all the access information of that client.

. She wants to ensure that the cloud service providers of MyWeb do not share her data with other service providers, so that the accountability provided for individual users can also be expected from the cloud service providers.

With the above scenario in mind, we identify the common requirements and develop several guidelines to achieve data accountability in the cloud. A user, who subscribed to a certain cloud service, usually needs to send his/her data as well as associated access control policies (if any) to the service provider. After the data are received by the cloud service provider, the service provider will have granted access rights, such as read, write, and copy, on the data. Using conventional access control mechanisms, once the access rights are granted, the data will be fully available at the service provider. In order to track the actual usage of the data, we aim to develop novel logging and auditing techniques which satisfy the following requirements:

1. The logging should be decentralized in order to adapt to the dynamic nature of the cloud. More specifically, log files should be tightly bounded with the corresponding data being controlled, and require minimal infrastructural support from any server.
2. Every access to the user's data should be correctly and automatically logged. This requires integrated techniques to authenticate the entity who accesses the data, verify, and record the actual operations on the data as well as the time that the data have been accessed.
3. Log files should be reliable and tamper proof to avoid illegal insertion, deletion, and modification by malicious parties. Recovery mechanisms are also desirable to restore damaged log files caused by technical problems.
4. Log files should be sent back to their data owners periodically to inform them of the current usage of their data. More importantly, log files should be retrievable anytime by their data owners when needed regardless the location where the files are stored.
5. The proposed technique should not intrusively monitor data recipients' systems, nor it should introduce heavy communication and computation overhead, which otherwise will hinder its feasibility and adoption in practice.

III. CLOUD INFORMATION ACCOUNTABILITY

Here we present an overview of the Cloud Information Accountability framework and discuss how the CIA framework meets the design requirements discussed in the previous section.

The Cloud Information Accountability framework

proposed in this work conducts automated logging and distributed auditing of relevant access performed by any entity, carried out at any point of time at any cloud service provider. It has two major components: logger and log harmonizer.

A. CIA Components

Two major components of the CIA are, the first being the logger, and the second being the log harmonizer. The logger is the component which is strongly coupled with the user's data, so that it is downloaded when the data are accessed, and is copied whenever the data are copied. It handles a particular instance or copy of the user's data and is responsible for logging access to that instance or copy. The log harmonizer forms the central component which allows the user access to the log files.

The logger is strongly coupled with user's data (either single or multiple data items). Its main tasks include automatically logging access to data items that it contains, encrypting the log record using the public key of the content owner, and periodically sending them to the log harmonizer. It may also be configured to ensure that access and usage control policies associated with the data are honoured. For example, a data owner can specify that user X is only allowed to view but not to modify the data. The logger will control the data access even after it is down-loaded by user X.

The logger requires only minimal support from the server (e.g., a valid Java virtual machine installed) in order to be deployed. The tight coupling between data and logger, results in a highly distributed logging system, therefore meeting our first design requirement. Furthermore, since the logger does not need to be installed on any system or require any special support from the server, it is not very intrusive in its actions, thus satisfying our fifth requirement. Finally, the logger is also responsible for generating the error correction information for each log record and sends the same to the log harmonizer. The error correction information combined with the encryption and authentication mechanism provides a robust and reliable recovery mechanism, therefore meeting the third requirement.

The log harmonizer is responsible for auditing.

Being the trusted component, the log harmonizer generates the master key. It holds on to the decryption key for the IBE key pair, as it is responsible for decrypting the logs. Alternatively, the decryption can be carried out on the client end if the path between the log harmonizer and the client is not trusted. In this case, the harmonizer sends the key to the client in a secure key exchange.

It supports two auditing strategies: push and pull. Under the push strategy, the log file is pushed back to the data owner periodically in an automated fashion. The pull mode is an on-demand approach, whereby the log file is obtained by the data owner as often as requested. These two modes allow us to satisfy the aforementioned fourth design requirement. In case there exists multiple loggers for the same set of data items, the log harmonizer will merge log records from them before sending back to the data owner. The log harmonizer is also responsible for handling log file corruption. In addition, the log harmonizer can itself carry out logging in addition to auditing. Separating the logging and auditing functions improves the performance. The logger and the log harmonizer are both implemented as lightweight and portable JAR files. The JAR file implementation provides automatic logging functions, which meets the second design requirement.

B. Flow of Data

The overall CIA framework, combining data, users, logger and harmonizer is sketched in Fig. 1. At the beginning, each user creates a pair of public and private keys based on Identity-Based Encryption [4] (step 1 in Fig. 1). This IBE scheme is a Weil-pairing-based IBE scheme, which protects us against one of the most prevalent attacks to our architecture. Using the generated key, the user will create a logger component which is a JAR file, to store its data items.

The JAR file includes a set of simple access control rules specifying whether and how the cloud servers, and possibly other data stakeholders (users, companies) are authorized to access the content itself. Then, he sends the JAR file to the cloud service provider that he subscribes to. To authenticate the CSP to the JAR (steps 3-5 in Fig. 1), we use Open SSL-based certificates, wherein a trusted certificate authority certifies the CSP. In the event that the access is requested by a user, we employ SAML-based authentication [8], wherein a trusted identity provider issues certificates verifying the user's identity based on his username.

Once the authentication succeeds, the service provider (or the user) will be allowed to access the data enclosed in the JAR. Depending on the configuration settings defined at the time of creation, the JAR will provide usage control associated with logging, or will provide only logging functionality. As for the logging, each time there is an access to the data; the JAR will automatically generate a log record, encrypt it using the public key distributed by the data owner, and store it along with the data (step 6 in Fig. 1). The encryption of the log file prevents unauthorized changes to the file by attackers. The data owner could opt to reuse the same key pair for all JARs or create different key pairs for separate JARs. Using separate keys can enhance the security (detailed discussion is in Section 7) without introducing any overhead except in the initialization phase. In addition, some error correction information will be sent to the log harmonizer to handle possible log file corruption

(step 7 in Fig. 1). To ensure trustworthiness of the logs, each record is signed by the entity accessing the content. Further, individual records are hashed together to create a chain structure, able to quickly detect possible errors or missing records. The encrypted log files can later be decrypted and their integrity verified. They can be accessed by the data owner or other authorized stakeholders at any time for auditing purposes with the aid of the log harmonizer (step 8 in Fig. 1).

Our proposed framework prevents various attacks such as detecting illegal copies of users' data. Note that our work is different from traditional logging methods which use encryption to protect log files. With only encryption, their logging mechanisms are neither automatic nor distributed. They require the data to stay within the boundaries of the centralized system for the logging to be possible, which is however not suitable in the cloud.

Example 2. Considering Example 1, Joy can enclose her photographs and access control policies in a JAR file and send the JAR file to the cloud service provider. With the aid of control associated logging (called AccessLog in Section 5.2), Joy will be able to enforce the first four requirements and record the actual data access. On a regular basis, the push-mode auditing mechanism will inform Alice about the activity on each of her photo-graphs as this allows her to keep track of her clients' demographics and the usage of her data by the cloud service provider. In the event of some dispute with her clients, Alice can rely on the pull-mode auditing mechanism to obtain log records.

IV. LOGGING MECHANISM

A. *Logger Structure*

We leverage the programmable capability of JARs to conduct automated logging. A logger component is a nested Java JAR file which stores a user's data items and corresponding log files. Our proposed JAR file consists of one outer JAR enclosing one or more inner JARs.

The main responsibility of the outer JAR is to handle authentication of entities which want to access the data stored in the JAR file. In our context, the data owners may not know the exact CSPs that are going to handle the data. Hence, authentication is specified according to the servers' functionality (which we assume to be known through a lookup service), rather than server's URL or identity.

mple 3. Consider Example 1. Suppose that Joy's photographs are classified into three categories according to the locations where the photos were taken. The three groups of photos are stored in three inner JAR J1, J2, and J3, respectively, associated with different access control policies. If some entities are allowed to access only one group of the photos, say J1 , the outer JAR will just render the corresponding inner JAR to the entity based on the policy evaluation result.

Each inner JAR contains the encrypted data, class files to facilitate retrieval of log files and display enclosed data in a suitable format, and a log file for each encrypted item. We support two options:

- PureLog. Its main task is to record every access to the data. The log files are used for pure auditing purpose.

- AccessLog. It has two functions: logging actions and enforcing access control. In case an access request is denied, the JAR will record the time when the request is made. If the access request is granted, the JAR will additionally record the access information along with the duration for which the access is allowed.

The two kinds of logging modules allow the data owner to enforce certain access conditions either proactively (in case of AccessLogs) or reactively (in case of PureLogs). For example, services like billing may just need to use PureLogs. AccessLogs will be necessary for services which need to enforce service-level agreements such as limiting the visibility to some sensitive content at a given location.

To carry out these functions, the inner JAR contains a class file for writing the log records, another class file which corresponds with the log harmonizer, the encrypted data, a third class file for displaying or downloading the data (based on whether we have a PureLog, or an AccessLog), and the public key of the IBE key pair that is necessary for encrypting the log records. No secret keys are ever stored in the system. The outer JAR may contain one or more inner JARs, in addition to a class file for authenticating the servers or the users, another class file finding the correct inner JAR, a third class file which checks the JVM's validity using oblivious hashing. Further, a class file is used for managing the GUI for user authentication and the Java Policy.

B. *Generation of Log Record*

Log records are generated by the logger component. Logging occurs at any access to the data in the JAR, and new log entries are appended sequentially, in order of creation. Each record r_i is encrypted individually and appended to the log file. In particular, a log record takes the following form:

C. *Log Dependability*

1. Availability of JARs:

To protect against attacks perpetrated on offline JARs, the CIA includes a log harmonizer which has two main responsibilities: to deal with copies of JARs and to recover corrupted logs.

Each log harmonizer is in charge of copies of logger components containing the same set of data items. The harmonizer is implemented as a JAR file. It does not contain the user's data items being audited, but consists of class files for both a server and a client processes to allow it to communicate with its logger components. The harmonizer stores error correction information sent from its logger components, as well as the user's IBE decryption key, to decrypt the log records and handle any duplicate records. Duplicate records result from copies of the user's data JARs. Since user's data are strongly coupled with the logger component in a data JAR file, the logger will be copied together with the user's data. Consequently, the new copy of the logger contains the old log records with respect to the usage of data in the original data JAR file. Such old log records are redundant and irrelevant to the new copy of the data. To present the data owner an integrated view, the harmonizer will merge log records from all copies of the data JARs by eliminating redundancy.

2. Correctness of Log:

For the logs to be correctly recorded, it is essential that the JRE of the system on which the logger components are running remain unmodified. To verify the integrity of the logger component, we rely on a two-step process: 1) we repair the JRE before the logger is launched and any kind of access is given, so as to provide guarantees of integrity of the JRE. 2) We insert hash codes, which calculate the hash values of the program traces of the modules being executed by the logger component. This helps us detect modifications of the JRE once the logger component has been launched, and are useful to verify if the original code flow of execution is altered.

V. END TO END AUDITING

To allow users to be timely and accurately informed about their data usage, our distributed logging mechanism is complemented by an innovative auditing mechanism. We support two complementary auditing modes: 1) push mode; 2) pull mode.

Push mode: In this mode, the logs are periodically pushed to the data owner (or auditor) by the harmonizer. The push action will be triggered by either type of the following two events: one is that the time elapses for a certain period according to the temporal timer inserted as part of the JAR file; the other is that the JAR file exceeds the size stipulated by the content owner at the time of creation. After the logs are sent to the data owner, the log files will be dumped, so as to free the space for future access logs. Along with the log files, the error correcting information for those logs is also dumped.

Pull mode: This mode allows auditors to retrieve the logs anytime when they want to check the recent access to their own data. The pull message consists simply of an FTP pull command, which can be issues from the command line. For naive users, a wizard comprising a batch file can be easily built. The request will be sent to the harmonizer, and the user will be informed of the data's locations and obtain an integrated copy of the authentic and sealed log file.

VI. 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.

VII. REFERENCES

[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 Manage-ment 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/tc home.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.

[21] R. Kailar, "Accountability in Electronic Commerce Protocols," IEEE Trans. Software Eng., vol. 22, no. 5, pp. 313-328, May 1996.

[22] W. Lee, A. Cinzia Squicciarini, and E. Bertino, "The Design and Evaluation of Accountable Grid Computing System," Proc. 29th IEEE Int'l Conf. Distributed Computing Systems (ICDCS '09), pp. 145-154, 2009.

[23] J.H. Lin, R.L. Geiger, R.R. Smith, A.W. Chan, and S. Wanchoo, Method for Authenticating a Java Archive (jar) for Portable Devices, US Patent 6,766,353, July 2004.

[24] F. Martinelli and P. Mori, "On Usage Control for Grid Systems," Future Generation Computer Systems Vol. 26, no. 7. Pp. 1032-1042, 2010.

[25] T. Mather, S. Kumaraswamy, and S. Latif, Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance (Theory in Practice), first ed. O' Reilly, 2009.

[26] M.C. Mont, S. Pearson, and P. Bramhall, "Towards Accountable Management of Identity and Privacy: Sticky Policies and Enforce-able Tracing Services," Proc. Int'l Workshop Database and Expert Systems Applications (DEXA), pp. 377-382, 2003.

[27] S. Oaks, Java Security. O'Really, 2001.