Authentication of Service Provider in the Cloud using Distributed Accountability

G.Ranjith kumar
CSE. Jyothishmathi Eng. College
Jyothishmathi Engineering College
Karimnagar, India
ranjitgotte@gmail.com

Y.Satyam
CSE. Jyothishmathi Eng. College
Jyothishmathi Engineering College
Karimnagar, India
yedla.satyam@gmail.com

P.Venkatram Reddy
CSE. Jyothishmathi Eng. College
Jyothishmathi Engineering College
Karimnagar, India
venki.palakala@gmail.com

Abstract - Cloud computing enables us to use services over Internet whenever we require. A major feature of the cloud services is that user's data are usually processed by unknown machines to the users, so users afraid of having no control on their own data. This is a barrier to the adoption of cloud services. To address this problem, in this paper, we propose a decentralized information accountability framework to keep track of the actual usage of the user's data in the cloud. In particular, we propose an object-centered 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 traveling 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 is 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. There are number of cloud computing services, such as Amazon, Google, Microsoft, Yahoo, and Salesforce [1]. The data processed on clouds are often outsourced and details of the services provided are abstracted from the users. Moreover, users may not know the machines which actually process and host their data. This leads to a number of issues related to accountability, including the handling of personally identifiable information. Because users won't have any control on their data. Such fears are becoming a significant barrier to the wide adoption of cloud services [2].

So it is required 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 agreements made at the time they sign on for services in the cloud. Conventional access control approaches developed for databases and operating systems, or approaches using a centralized server in distributed environments, are not suitable, due to the following features of cloud environments.

- 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.
- 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 environment.

To overcome the above problems, we propose a novel approach, namely Cloud Information Accountability (CIA) framework, based on the notion of information accountability [3]. Information accountability focuses on keeping the data usage transparent and trackable. Our proposed CIA framework provides end-to-end accountability. One of the main innovative features of the CIA framework lies in its ability of maintaining lightweight and powerful accountability that combines aspects of access control, usage control and authentication in a highly distributed fashion. By means of the CIA, data owners can track not only whether or not the service-level agreements are being honored, but also enforce access and usage control rules as needed. Besides accountability feature we also develop two distinct modes which are used to know logs in the cloud. Those are as follows:

- The *push mode* refers to logs being periodically sent to the data owner.
- The *pull mode* refers to an alternative approach whereby the user (or another authorized party) can retrieve the logs as needed.

The design of the CIA framework presents 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. 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.

Here, 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 [4]). 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).

We tested our CIA framework in a cloud testbed, the Emulab testbed [7], with Eucalyptus as middleware [6].Our experiments demonstrate the efficiency, scalability and granularity of our approach. In addition, we also provide a detailed security analysis and discuss the reliability and strength of our architecture in the face of various nontrivial attacks, launched by malicious users or due to compromised Java Running Environment (JRE).

This paper is an extension of conference paper [5]. We have made the following new contributions. First, we integrated integrity checks and oblivious hashing (OH) technique to our system in order to strengthen the dependability of our system in case of compromised JRE .We also updated the log records structure to provide additional guarantees of integrity and authenticity. Second, we extended the security analysis to cover more possible attack scenarios. Third, we report the results of new experiments and provide a thorough evaluation of the system performance. Finally, we have improved the presentation by adding more examples and illustration graphs.

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. An author of a book , plans to sell his book by using Cloud Services. For his business in the cloud, he 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.

From the above requirements we develop many 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 on the data. In order to track the actual usage of the we develop logging and auditing techniques which satisfy the following requirements:

- The logging should be decentralized in order to adapt to the dynamic nature of the cloud.
- Every access to the user's data should be correctly and automatically logged.
- Log files should be reliable and tamper proof to avoid illegal modification by malicious parties.
- Log files should be sent back to their data owners periodically to inform them of the current usage of their data.

III. CLOUD INFORMATION ACCOUNTABILITY

In this section, we present an overview of the Cloud Information Accountability (CIA) framework, which has two major components: *logger* and *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 tight coupling between data and logger, results in a highly distributed logging system, therefore meeting our first design requirement.

The *log harmonizer* is responsible for auditing. 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. Under the *pull mode* the log file is obtained by the data owner as often as requested. These two modes will satisfy the fourth design requirement.

The logger and the log harmonizer are both implemented as portable JAR files. The JAR file implementation provides automatic logging functions, which meets the second design requirement.

The logger is also responsible for generating the error correction information for each log record and send 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 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 (IBE) [8] (step 1 in Fig. 1) , which prevents attacks to our system. Using these keys, the user will create a logger component which is a JAR file, to store its data items. This JAR file includes access control rules, using which user and cloud server are authorized. 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. Once the authentication succeeds, the service provider

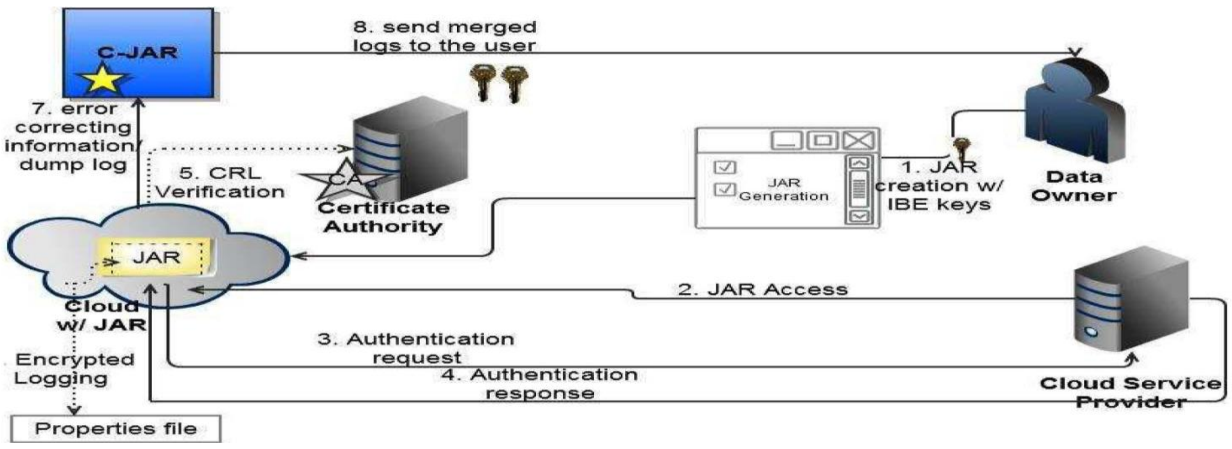

Fig. 1. Overview of the cloud information accountability framework [10]

the service provider will be allowed to access the data enclosed in the JAR. As per 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 reuse the same key pair for all JARs or create different key pairs for separate JARs. Usage of separate keys will enhance the security. In addition, some error correction information will be sent to the log harmonizer to handle possible log file corruption (step 7 in Fig. 1). The encrypted log files can later be decrypted and their integrity verified. They can be accessed by the data owner at any time for auditing purposes with the help of the log harmonizer (step 8 in Fig. 1).

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. Outer JAR will handle authentication of entities which want to access the data stored in the JAR file. Authentication is specified according to the server's functionality (which we assume to be known through a lookup service), rather than the server's identity, because data owner may not know the exact CSP that is going to handle the data. Here we use java policy access control functionality, which will be included in outer JAR. the data owner can specify the permissions in user-centric terms as opposed to the usual code-centric security offered by Java.

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:* It is used 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.

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.

IV. Log Record Generation

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 LR = $(r_1,\ldots,r_k)$. Each record r_i is encrypted individually and appended to the log file. In particular, a log record takes the following form:

$r_i = (ID, Act, T, Loc, h((ID, Act ,T, Loc) |r_i\text{-}1|\ldots\ldots|r_1),Sig)$.

Here, r_i indicates that an entity identified by I D has performed an action Act on the user's data at time T at location Loc. The component $h((ID,Act,T,Loc)|r_i\text{-}1| \ldots |r_1)$ corresponds to the checksum of the records preceding the newly inserted one, concatenated with the main content of the record itself (we use I to denote concatenation). The checksum is computed using a collision-free hash function [9]. The component sig denotes the signature of the record created by the server. If more than one file is handled by the same logger, an additional ObjIDfield is added to each record.

V. SECURITY IN CLOUD INFORMATION ACCOUNTABILITY(CIA)

In this section we discuss possible attacks in CIA and their prevention. The following are few of attacks that are expected in our CIA. Those are:

- Copying JAR file.
- Disassembling JAR file.
- Man –in-the-Middle attack.
- ***Copying JAR file:*** In this attack an attacker will copy the entire JAR file. However, such attack will be detected by our auditing mechanism, where we use push and pull modes. Every JAR file is required to send log records to the harmonizer. In particular, with the push mode, the harmonizer will send the logs to data owners periodically. If copies of JAR file move to the place where harmonizer can't be connected, the copies of JARs will soon become inaccessible and contents of JAR file will be disable.

- ***Disassembling JAR file:*** In this attack an attacker disassemble the JAR file of the logger and then try to extract useful information out of it. Once the JAR files are disassembled, the attacker is in possession of the public IBE key used for encrypting the log files, the encrypted log file itself, and all .class files. The attacker may try to identify which encrypted log records correspond to his actions by mounting a chosen plaintext attack to obtain some pairs of encrypted log records and plain texts. However, the adoption of the Weil Pairing algorithm ensures that the CIA framework has both chosen ciphertext security and chosen plaintext security in the random oracle model [8]. Therefore, the attacker will not be able to decrypt any data or log files in the disassembled JAR file.
- ***Man –in-the-Middle attack:*** There are two points in time that the attacker can replay the messages. In case of first one service provider has completely disconnected and ended a session with certificate authority. In the second case the actual service provider is disconnected but the session is not over, so the attacker may try to renegotiate the connection. From the above two cases we have attacker's threat mostly in second case. This second attack is not possible in our system, because renegotiation is banned in latest version of OpenSSL.

VI. PERFORMANCE STUDY

In our experiment, the overhead can occur at three points: during the authentication, during encryption of a log record, and during the merging of the logs. In the first round of experiments, we are interested in finding out the time taken to create a log file when there are entities continuously accessing the data, causing continuous logging. Results are shown in the following Fig. 2.

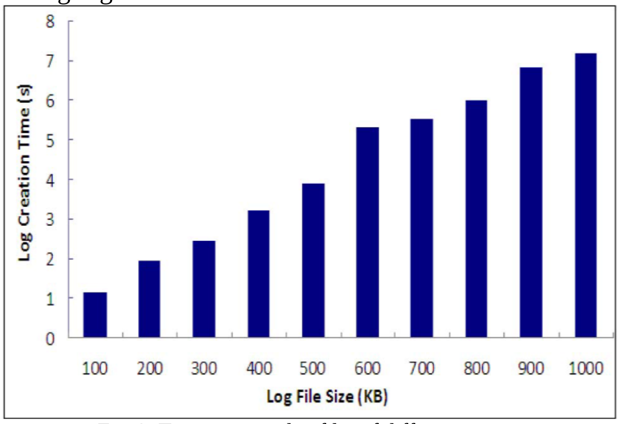

Fig. 2. Time to create log files of different sizes.

From the above figure we can conclude that time to create a log file will be increased linearly with the size of the log file.

The next overhead can occur during authentication of a CSP. If the time taken for this authentication is too long, then it becomes a bottleneck for accessing the enclosed data. To evaluate this, the head node issues OpenSSL certificates for the computing nodes and we measured the total time for the OpenSSL authenticaton to be completed and the certificate

The third overhead is in merging of logs. In this experiment each log file has some records in common with other logs. The exact number of common records was random for each repetition of experiment. The time was averaged over 10 repetitions. We tested the time to merge up to 70 log files of 100 KB, 300 KB, 500 KB, 700 KB, 900 KB, and 1 MB each. The results are shown in Fig. 3. We can observe that the time increases almost linearly to the number of files and size of files, with the least time being taken for merging two 100 KB log files at 59 ms, while the time to merge 70 1 MB files was 2.35 minutes.

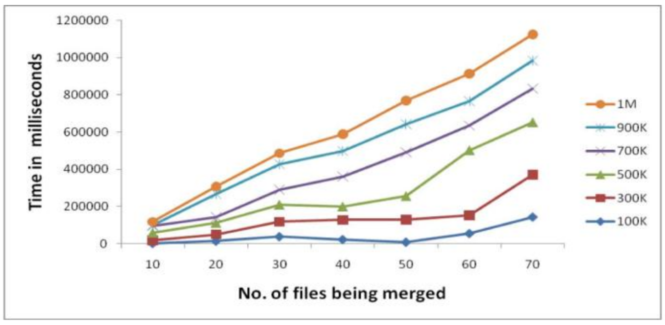

Fig. 3. Time to merge log files.

VII. CONCLUSION

In this paper 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.

REFERENCES

[1] 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.
[2] S. Pearson and A. Charlesworth, "Accountability as a Way Forward for Privacy Protection in the Cloud," Proc. First Int'l Conf. Cloud Computing, 2009.
[3] D.J. Weitzner, H. Abelson, T. Berners-Lee, J. Feigen-baum, J. Hendler, and G.J. Sussman, "Information Accountability," Comm. ACM, vol. 51, no. 6, pp. 82-87, 2008.
[4] Flickr, http://www.flickr.com/, 2012.
[5] S. Sundareswaran, A. Squicciarini, D. Lin, and S. Huang,"Promoting Distributed Accountability in the Cloud," Proc. IEEE Int'l Conf. Cloud Computing, 2011.
[6] Eucalyptus Systems, http://www.eucalyptus.com/, 2012.
[7] Emulab Network Emulation Testbed, www.emulab.net, 2012.
[8] 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.
[9] B. Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C. John Wiley & Sons, 1993.
[10] Pankaj Kumar Singh, Ajit kumar, and R.Karthikeyan, " Ensuring Distributed Accountability for Data Sharing in the Cloud " ijarcsse, Valume 3, Issue 3, march 2013.