International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal, Volume 3, Special Issue 1, January 2013)

International Conference on Information Systems and Computing (ICISC-2013), INDIA.

SECURE FRAMEWORK FOR DATA SHARING IN CLOUD COMPUTING ENVIRONMENT

Nilutpal Bose [1], Mrs. G. Manimala [2]

M.E. (Computer Science & Eng), Sri Sai Ram Engineering College, West Tambaram Chennai India
Associate Professor (Computer Science & Eng), Sri Sai Ram Engineering College, West Tambaram Chennai India
Email: [1]*nilutpal.bose@yahoo.co.in*, [2]*manimala.cse@sairam.edu.in*

Abstract

Cloud computing is the use of computing of sources (hardware and software) that are delivered as a service over a network(typically the internet).It enables highly scalable services to be easily consumed over the Internet on an as-needed basis. A major characteristic of the cloud services is that users' data are usually processed remotely in unknown machines that users do not operate. It can become a substantial roadblock to the wide adoption of cloud services. To address this problem, we propose a highly decentralized answerability framework to keep track of the actual usage of the user's data in the cloud. 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. The proposed methodology will also take concern of the JAR file by converting the JAR into obfuscated code which will adds an additional layer of security to the infrastructure. Apart from that we are going to extend the security of user's data by provable data possessions for integrity verification.

***Keywords*-- Cloud computing, data sharing, information accountability framework, Provable data possession.**

I. INTRODUCTION

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. The JAR file includes a set of simple access control rules specifying whether and how the cloud servers and possibly other data stakeholders are authorized to access the content itself. Apart from that we are going to check the integrity of the JRE on the systems on which the logger components is initiated. This integrity checks are carried out by using oblivious hashing .The proposed methodology will also take concern of the JAR file by converting the JAR into obfuscated code which will adds an additional layer of security to the infrastructure. Apart from that we are going to extend the security of user's data by provable data possessions for integrity verification. 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.

II. PROPOSED SYSTEM

2.1 JAR Generation

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. 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.

2.2 Obfuscation

In software development, obfuscation is the deliberate act of creating obfuscated code, *i.e.* source or machine code that is difficult for humans to understand. Programmers may deliberately obfuscate code to conceal its purpose (security through obscurity) or its logic, in order to prevent tampering, deter reverse engineering , or as a puzzle or recreational challenge for someone reading the source code.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal, Volume 3, Special Issue 1, January 2013)
International Conference on Information Systems and Computing (ICISC-2013), INDIA.

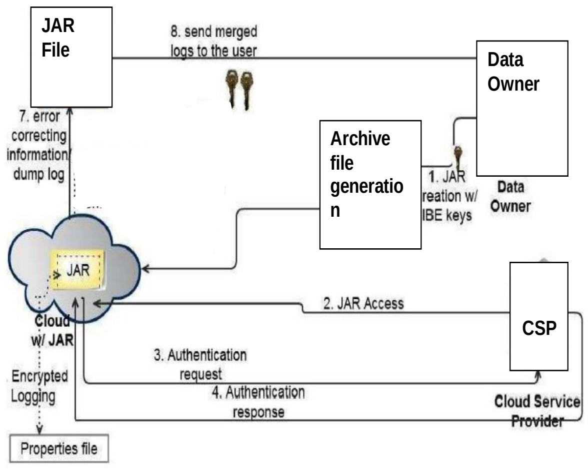
JAR File
8. send merged logs to the user
Data Owner
7. error correcting information dump log
Archive file generation
1. JAR creation w/ IBE keys
Data Owner
JAR
2. JAR Access
Cloud w/ JAR
CSP
3. Authentication request
4. Authentication response
Encrypted Logging
Cloud Service Provider
Properties file

Fig. 1 Existing system Architecture

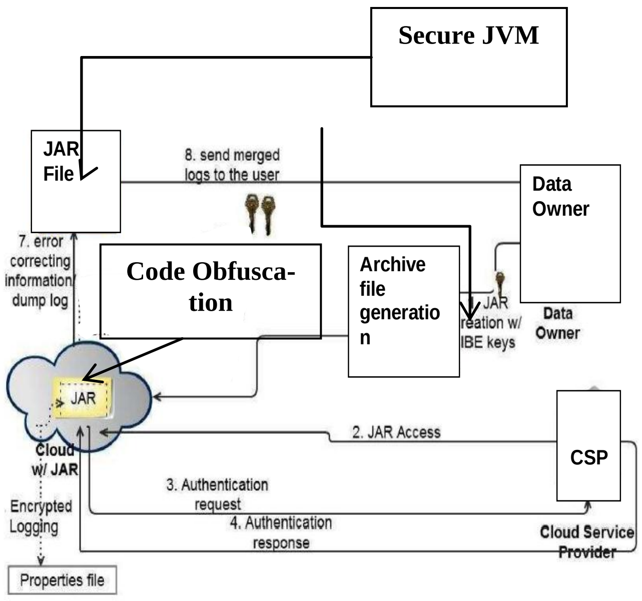

Fig. 2 Proposed system Architecture

Programs known as *obfuscators* transform readable code into obfuscated code using various techniques. However, even if you use a code obfuscator that forces all decompilers to fail completely, a byte code disassembler would still work. Remember that the JVM instruction set includes high-level.

Plain text Code:	Obfuscated code:
var a="Hello World!"; function MsgBox(msg) { alert(msg+"\n"+a); } MsgBox("OK");	var _0x8e48=["\x48\x65\x6C\x6C\x6F\x20\x57\x6F\x72\x6C\x64\x21","\x0A","\x4F\x4B"];var a=_0x8e48[0];function MsgBox(_0xab5dx3){alert(_0xab5dx3+_0x8e48[1]+a);};MsgBox(_0x8e48[2]);

Fig.3 Example of obfuscated code

Instructions, as opposed to real CPUs such as x86 or ARM, so disassembled Java is easier to understand than disassembled C++. It would therefore make sense to also "distort" the overall structure of the program. The more advanced obfuscation techniques include class hierarchy changes, method inlining and outlining, loop unrolling, array folding/flattening, etc.

2.3 Provable Data Possession

Provable data possession (PDP) (or proofs of retrievability (POR)) is such a probabilistic proof technique for a storage provider to prove the integrity and ownership of clients' data without downloading data.

Definition 1 (PDP): A provable data possession $\mathcal{S}$ = ($KeyGen$, $T\ agGen$, $Proof$) is a collection of two algorithms ($KeyGen$, $TagGen$) and an interactive proof system $Proof$, as follows:

$KeyGen(1\kappa)$**:** takes a security parameter κ as input, and returns a secret key sk or a public-secret keypair (pk, sk);

$TagGen(sk,\ F,\mathcal{P})$**:** takes as inputs a secret key sk, a file F, and a set of cloud storage providers $\mathcal{P}$ = {Pk}, and returns the triples (ζ,ψ, σ), where ζ is the secret in tags, ψ = (u,$\mathcal{H}$) is a set of verification parameters u and an index hierarchy $\mathcal{H}$ for F, σ = {$\sigma(k)$}$Pk\in\mathcal{P}$ denotes a set of all tags, $\sigma(k)$ is the tag of the fraction $F(k)$ of F in Pk;

Provable Data Possession (PDP) scheme is a collection of four polynomial-time algorithms

$\mathrm{KeyGen}(1^k) \rightarrow (pk,sk)$

$\mathrm{TagBlock}(pk,sk,m) \rightarrow T_m$

- $\sum$ is collection from T_m

$\mathrm{GenProof}(pk,F,chal,\sum) \rightarrow V$

$\mathrm{CheckProof}(pk,sk,chal,V) \rightarrow 0/1$

2.3.1 Pre-process and store

- C stores F on S
  - C:client
  - F:file, $F=(m_1,\ldots,m_n)$
  - S:server
- C keeps some data

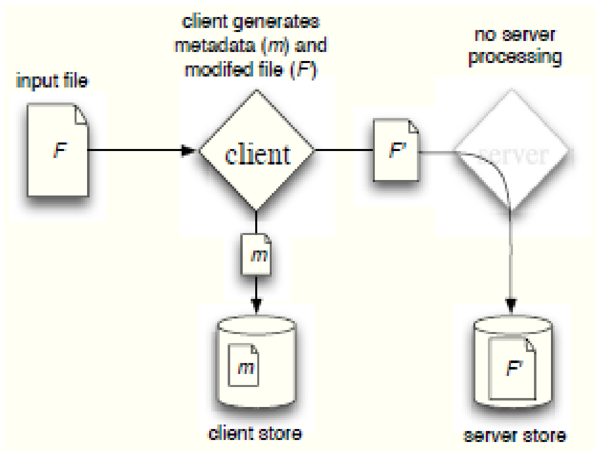

Fig. 4 Pre-process Structure

2.4 Logger Creation

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. 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 the server's URL or identity. The data owner can specify the permissions in user-centric terms as opposed to the usual code-centric security offered by Java, using Java Authentication and Authorization Services.

Moreover, the outer JAR is also in charge of selecting the correct inner JAR according to the identity of the entity who requests the data.

2.5 Log tape propagation

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 =<r1; . . . ; rk>. Each record ri is encrypted individually and appended to the log file. When there is a view-only access request, the inner JAR will decrypt the data on the fly and create a temporary decrypted file. The decrypted file will then be displayed to the entity using the Java application viewer in case the file is displayed to a human user. Presenting the data in the Java application, viewer disables the copying functions using right click or other hot keys such as Print Screen.

2.6 Mode Setting

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. Pull mode: This mode allows auditors to retrieve the logs anytime when they want to check the recent access to their own data.

III. PROPOSED METHODOLOGY

3.1 The Logging Mechanism

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 the server's URL or identity.

3.1.1 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 = (r1; . . . ; rk). Each record ri is encrypted individually and appended to the log file. In particular, a log record takes the following form:

$$r_i = \langle ID, Act, T, Loc, h((ID, Act, T, Loc)|r_i - 1|\ldots|r_1), sig\rangle.$$

Here, ri indicates that an entity identified by ID has performed an action Act on the user's data at time T at location Loc.

3.2 PDP (PROVABLE DATA POSSESSION)

3.2.1 Verify data possession

- C sends a challenge
  - chal: a challenge with random value
- S process V from chal and F
  - V: a proof
  - return V
- C checks V

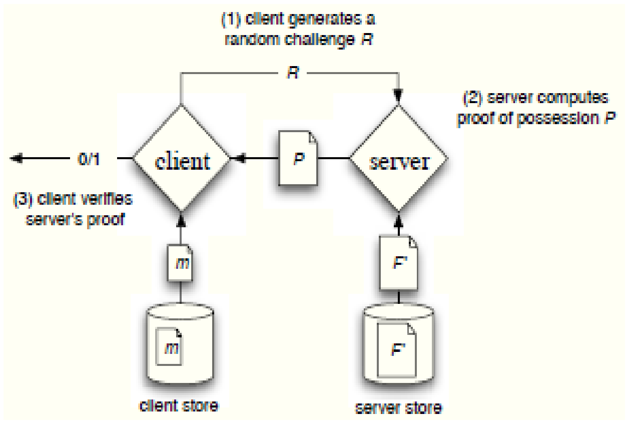
(1) client generates a random challenge R
R
(2) server computes proof of possession P
0/1
client
P
server
(3) client verifies server's proof
m
F'
m
F'
client store
server store

Fig. 5 Verify data possession

A PDP system can be constructed from a PDP scheme in two phases:

Setup

- C runs KeyGen, TagBlock and sends pk,F,∑ to S
  Challenge
- C sends chal to S
- S runs GenProof and sends V to C
- C check V with Check Proof

We also showed that our scheme provided all security properties required by zero knowledge interactive proof system, so that it can resist various attacks even if it is deployed as a public audit service in clouds.

3.3 Push and Pull Mode

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 techniques: 1) push mode; 2) pull mode.

3.3.1 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. This push mode is the basic mode which can be adopted by both the Pure Log and the Access Log, regardless of whether there is a request from the data owner for the log files. This mode serves two essential functions in the logging architecture: 1) it ensures that the size of the log files does not explode and 2) it enables timely detection and correction of any loss or damage to the log files. Concerning the latter function, we notice that the auditor, upon receiving the log file, will verify its cryptographic guarantees, by checking the records' integrity and authenticity. By construction of the records, the auditor, will be able to quickly detect forgery of entries, using the checksum added to each and every record.

3.3.2 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. 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.

3.4 Data Flow

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 (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.

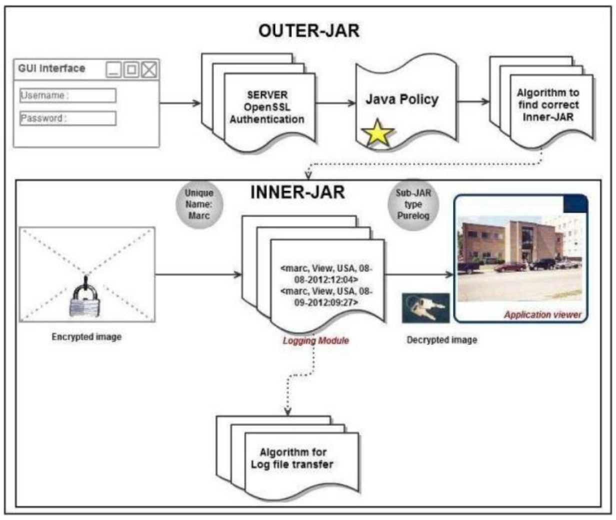
OUTER-JAR
GUI Interface
Username:
Password:
SERVER OpenSSL Authentication
Java Policy
Algorithm to find correct Inner-JAR
INNER-JAR
Unique Name: Marc
Sub-JAR type Purelog
<marc, View, USA, 08-08-2012:12:04>
<marc, View, USA, 08-09-2012:09:27>
Encrypted image
Logging Module
Decrypted image
Application viewer
Algorithm for Log file transfer

Fig. 6 The structure of the JAR file

Once the authentication succeeds, the service provider (or the user) will be allowed to access the data enclosed in the JAR. 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.

IV. Security Analysis

4.1 Copying Attack

The most intuitive attack is that the attacker copies entire JAR files. The adversary may assume that doing so allows accessing the data in the JAR file without being noticed by the data owner. However, such attack will be detected by our auditing mechanism. Recall that 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. That is, even if the data owner is not aware of the existence of the additional copies of its JAR files, he will still be able to receive log files from all existing copies. If attackers move copies of JARs to places where the harmonizer cannot connect, the copies of JARs will soon become inaccessible.

4.2 Data Leakage Attack

An Attack by which an adversary can easily obtain the stored data through verification process after running or wiretapping sufficient verification communications. An attacker uses well-formed requests to an application, service, or device that results in the inadvertent disclosure of sensitive information by exploiting weaknesses in the design or configuration of the target resulting in the target revealing more information to an attacker than intended. The attacker may collect this information through a variety of methods including active querying as well as passive observation. Information may include details regarding the configuration or capabilities of the target, clues as to the timing or nature of activities, or otherwise sensitive information. Often this sort of attack is undertaken in preparation for some other type of attack, although the collection of information may be the end goal of the attacker in some cases. Information retrieved may aid the attacker in making inferences about potential weaknesses, vulnerabilities, or techniques that assist the attacker's objectives. Data leaks may come various forms, including confidential information stored in insecure directories, or via services that provide rich error or diagnostic messages in response to normal queries.

4.2 Disassembling Attack

Another possible attack is to disassemble the JAR file of the logger and then attempt to extract useful information out of it or spoil the log records in 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 the *.class files. Therefore, the attacker has to rely on learning the private key or subverting the encryption to read the log records. To compromise the confidentiality of the log 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 cipher text security and chosen plaintext security in the random oracle model. Therefore, the attacker will not be able to decrypt any data or log files in the disassembled JAR file. Even if the attacker is an authorized user, he can only access the actual content file but he is not able to decrypt any other data including the log files which are viewable only to the data owner. From the disassembled JAR files, the attackers are not able to directly view the access control policies either, since the original source code is not included in the JAR files. If the attacker wants to infer access control policies, the only possible way is through analyzing the log file. This is, however, very hard to accomplish since, as mentioned earlier, log records are encrypted and breaking the encryption is computationally hard. Also, the attacker cannot modify the log files extracted from a disassembled JAR.

V. ADVANTAGE

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. Providing defenses against man in middle attack, dictionary attack, Disassembling Attack, Compromised JVM Attack, Data leakage attack.PDP allows the users to remotely verify the integrity of there data It's Suitable for limited and large number of storages.

VI. CONCLUSION

We proposed modern 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. Apart from that we have enclosed PDP methodology to enhance the integrity of owner's data. In future, we plan to refine our approach to verify the integrity of JRE. For that we will look into whether it is possible to leverage the advantage of secure JVM being developed by IBM and we would like to enhance our PDP architecture from user end which will allow the users to check data remotely in an efficient manner in multi cloud environment.

REFERENCES

1. Text Books

[1] Cloud computing for dummies by Judith Hurwit ,Robin Bloor.

[2] Cloud Computing, Principles and Paradigms by John Wiley & Sons.

2. Conference Proceedings

[1] Ensuring Distributed Accountability for Data Sharing in the Cloud Author, Smitha Sundareswaran, Anna C.Squicciarini, Member, IEEE, and Dan Lin, IEEE Transactions on Dependable and Secure Computing ,VOL 9,NO,4 July/August 2012

[2] P. Ammann and S. Jajodia, "Distributed Timestamp Generation in Planar Lattice Networks," ACM Trans. Computer Systems, vol. 11, pp 205-225, Aug. 1993

[3] 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.

[4] Provable Data Possession for Integrity Verification in Multi-Cloud Storage Author Van Zhu, Hongxin Hu, Gail-Joon Ahn, Senior Member, IEEE, Mengyang Yu

[5] 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.

[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] Trusted Java Virtual Machine IBM, http://www.almaden.ibm.com/cs/projects/jvm/, 2012.

[8] Hsio Ying Lin,Tzeng.W.G, "A Secure Erasure Code-Based Cloud Storage System with Secure Data Forwarding ",IEEE transactions on parallel and distributed systems,2012.

[9] Yan Zhu, Hongxin Hu, Gail Joon Ahn, *Mengyang Yu, "*Cooperative Provable Data Possession for Integrity Verification in Multi-Cloud Storage" , IEEE transactions on parallel and distributed systems,2012.

[10] Cong Wang,Ning Cao,Kui Ren,Wenjing Lou ,"Enabling Secure and Efficient Ranked Keyword Search over Outsourced Cloud Data ",IEEE transactions on parallel and distributed systems,2012.

[11] Hsio Ying Lin,Tzeng.W.G, "A Secure Erasure Code-Based Cloud Storage System with Secure Data Forwarding ",IEEE transactions on parallel and distributed systems,2012.

[12] Brickell,E.,Jiangtao Li, "Enhanced Privacy ID: A Direct Anonymous Attestation Scheme with Enhanced Revocation Capabilities ",IEEE transactions on dependable and secure computing

3. Generic Website

[1] http://www.excelsior-usa.com/articles/java-obfuscators.html#encrypt

[2] http://www.brandonparker.net/code_obf.php\http://kailaspatil.blogspot.in/2009/09/obfuscated-code-examples.html