IJDCST @October Issue- V-1, I-6, SW-31
ISSN-2320-7884 (Online)
ISSN-2321-0257 (Print)

Accountability for Data Sharing in a Cloud

[1]M.Ravi Teja, [2]K.Praveen Kumar

[1]Dept of CSE,Marri Laxman Reddy Institute of Technology ,Dundigal,Hyderabad-500043, A.P, INDIA

[2] Dept of CSE, Marri Laxman Reddy Institute of Technology ,Dundigal, Hyderabad-500043, A.P, INDIA

Abstract— Cloud computing poses a variety of challenges to conventional advanced ICT, mostly due to the fact of the unprecedented scale and heterogeneity of the required infrastructure. Users' data are usually processed remotely in unknown machines that users do not own or operate. It needs to safeguard the security and durability of service based on the demand of users. The users are also using the cloud flexibly because of this there will be many security problems occur. To address this problem, here, we propose a novel highly decentralized information accountability framework to keep track of the actual usage of the users' data in the cloud. By using object-centered approach that enables enclosing our logging mechanism together with users' data and policies. Virtualizations is an essential technological characteristic of clouds which hides the technological complexity from the user and enables enhanced flexibility (through aggregation, routing transactions) and by using programmable JAR files 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 effectiveness and efficiency for proposed approaches.

Keywords—cloud computing; accountability; datasharing; styling; cloud environment;

I. INTRODUCTION

A 'cloud' is an elastic execution environment of resources involving multiple stakeholders and providing a metered service at multiple granularities for a specified level of quality (of service). Cloud providers typically centre on one type of cloud functionality provisioning: Infrastructure, Platform or Software / Application, though there is potentially no striction to offer multiple type sat the same time, which can often be observed in PaaS (Platform as a Service) providers which offer specific applications too, such as Google App Engine in combination with Google Docs. Due this combinatorial ability, these types are also often referred to "components". To date, there are a number of notable commercial and individual cloud computing services, including Amazon, Google, Microsoft, Yahoo, and Sales force. Details of the services provided are abstracted from the users who no longer need to be experts of technology infrastructure. Moreover, users may not know the machines which actually process and host their data. 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. 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]. Unlike privacy protection technologies which are built on the hide-it-or-lose-it perspective, information accountability focuses on keeping the data usage transparent and tractable. 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 accountability 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 honored, 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 the CIA framework 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. 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).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 required. 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.. We conduct experiments on a real cloud testbed.The results demonstrate the efficiency, scalability, and granularity of our approach. We also provide a detailed security analysis and discuss the reliability and strength of our architecture. 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. Fourth, we have added a detailed discussion on related works to prepare readers with a better understanding of background knowledge. Finally, we have improved the presentation by adding more examples and illustration graphs.

II. SECRITY PROVIDE

In this section, we first review related works addressing the privacy and security issues in the cloud. Then, we briefly discuss works which adopt similar techniques as our approach but serve for different purposes.

PRIVACY: To overcome the above problems, we propose a novel method, namely Cloud Information Accountability (CIA) framework, based on the notion of information accountability. Data Owner can upload the data into the cloud server after encrypted the data. User can subscribe into the cloud server with certain access polices such as read, write and copy of the original data. The Loggers and Log Harmonizer will have a track of the access logs and reports to the data owner. This Process ensures security.

Many of the tasks necessary with cloud computing must be automated. For example, to protect the integrity of the data, information stored on a single computer in the cloud must be replicated on other computers in the cloud. If that one computer goes offline, the cloud's programming automatically redistributes that computer's data to new computers in the cloud. Computing in the cloud may provide additional Such issues are due to the fact that, in the cloud, users' data and applications reside—at least for a certain amount of time—on the cloud cluster which is owned and maintained by a third party. Concerns arise since in the cloud it is not always clear to individuals why their personal information is requested or how it will be used or passed on to other parties. To date, little work has been done in this space, in particular with respect to accountability. Pearson et al. have proposed accountability mechanisms to address privacy concerns offend users [30] and then develop a privacy manager [31]. Their basic idea is that the user's private data are sent to the cloud in an encrypted form, and the processing is done on the encrypted data. The output of the processing is deobfuscated by the privacy manager to reveal the correct result. However, the privacy manager provides only limited features in that it does not guarantee protection once the data are being disclosed. In [7], the authors present a layered architecture for addressing the end-to-end trust management and accountability problem in federated systems. The authors' focus is very different from ours, in that they mainly leverage trust relationships for accountability, along with authentication and anomaly detection. Further, their solution requires third-party services to complete the monitoring and focuses on lower level monitoring of system resources. Researchers have investigated accountability mostly as a provable property through cryptographic mechanisms, particularly in the context of electronic commerce A representative work in this area is given by related to accountability in case of delegation. Delegation is complementary to our work, in that we do not aim at controlling the information workflow in the clouds. In a summary, all these works stay at a theoretical level and do not include any algorithm for tasks like mandatory logging. To the best of our knowledge, the only work proposing a distributed approach to accountability is from Lee and colleagues. The authors have proposed an agent-based system specific to grid computing. Distributed jobs, along with the resource consumption at local machines are tracked by static software agents. The notion of accountability policies in is related to ours, but it is mainly focused on resource consumption and on tracking of sub jobs processed at multiple computing nodes.

III. TECHNOLOGY USED FOR SECURITY

Java-based techniques for security, our methods are related to self-defending objects. Self-defending objects are an extension of the object-oriented programming paradigm, where software objects that offer sensitive functions or hold sensitive data are responsible for protecting those functions/data. Similarly, we also extend the concepts of object-oriented programming. The key difference in our implementations is that the authors still rely on a centralized database to maintain the access records, while the items being protected are held as separate files. In previous work, we provided a Java-based approach to prevent privacy leakage from indexing , which could be integrated with the CIA framework proposed in this work since they build on related architectures. In terms of authentication techniques, [Appel and Felten] proposed the Proof-Carrying authentication (PCA) framework. The PCA includes a high order logic language that allows quantification over predicates, and focuses on access control for web services. While related to ours to the extent that it helps maintaining safe, high-performance, mobile code, the PCA's goal is highly different from our research, as it focuses on validating code, rather than monitoring content. Another work is by Mont et al. who proposed an approach for strongly coupling content with access control, using Identity-Based Encryption (IBE) . We also leverage IBE techniques, but in a very different way. We do not rely on IBE to bind the content with the rules. Instead, we use it to provide strong guarantees for the encrypted content and the log files, such as protection against chosen plaintext and cipher text attacks. In addition, our work may look similar

to works on secure data provenance but in fact greatly differs from them in terms of goals, techniques, and application domains. Works on data provenance aim to guarantee data integrity by securing the data provenance. They ensure that no one can add or remove entries in the middle of a provenance chain without detection, so that data are correctly delivered to the receiver. Differently, our work is to provide data accountability, to monitor the usage of the data and ensure that any access to the data is tracked. Since it is in a distributed environment, we also log where the data go. However, this is not for verifying data integrity, but rather for auditing whether data receivers use the data following specified policies. Along the lines of extended content protection, usage control is being investigated as an extension of current access control mechanisms. Current efforts on usage control are primarily focused on conceptual analysis of usage control requirements and on languages to express constraints at various level of granularity. While some notable results have been achieved in this respect thus far, there is no concrete contribution addressing the problem of usage constraints enforcement, especially in distributed settings. The few existing solutions are partial, restricted to a single domain, and often specialized. Finally, general outsourcing techniques have been investigated over the past few years. Although only is specific to the cloud, some of the outsourcing protocols may also be applied in this realm. In this work, we do not cover issues of data storage security which are a complementary aspect of the privacy issues.

IV. SOLUTION FOR PROBLEM

We tested our CIA framework by setting up a small cloud, using the Emu lab test bed In particular, the test environment consists of several Open SSL-enabled servers: 1. Notice that we do not consider the attack on the log harmonizer component, since it is saved separately in either a secure proxy or at the user end and the attacker typically cannot access it. As a result, we consider that the attacker cannot extract the decryption keys from the log harmonizer. One head node which is the certificate authority and several computing nodes. Each of the servers is installed with eucalyptus .Eucalyptus is an open source cloud implementation for Linux-based systems. It is loosely on the basis of Amazon EC2, so bringing the powerful functionalities of Amazon EC2 into the open source domain. We set to work Linux-based servers running Fedora 10 OS. Each server has a 64-bit Intel Quad Core Xeon E5530 processor, 4 GB RAM, and a 500 GB Hard Drive. Each of the servers is arrayed to run the Open JDK runtime environment with Iced Tea. In the experiments, we first examine the time taken to create log file and then measure the overhead in the system. With respect to time, the overhead can occur at three points: at the time of the authentication, during encryption of a log record, and at the time of the merging of the logs. Also, with respect to storage overhead, we notice that our architectures very lightweight, in that the only data to be stored are provided by the actual files and the associated logs. Further, JAR appear as a compressor of the files that it handles. In particular, as proposed, multiple files can be managed by the same logger component. To this extent, we checked whether a single logger component, used to manage more than one file, results in storage overhead. Example 1. Alice, a professional photographer, plans to sell her photographs by using the SkyHigh 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 SkyHigh 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.

We aim to develop novel logging and auditing techniques which satisfy the following requirements:

In the first round of experiments, we are concerned in finding out the time taken to create a log file when there are entities continuously accessing the data, causing continuous logging. It is not surprising to identify that the time to create a log file increases linearly with the size of the log file. Specifically, the time to develop a 100 Kb file is about 114.5 ms while the time to create a 1 MB file averages at 731 ms. With this experiment as the baseline, one can figure out the amount of time to be specified between dumps, keeping other variables like space constraints or network traffic in mind.

- 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.
- 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.
- 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.
- 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.
- 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.

V. ABOUT CIA

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.

VI. COMPONENTS OF CIA

There are two major components of the CIA, 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 honored. 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 downloaded 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.

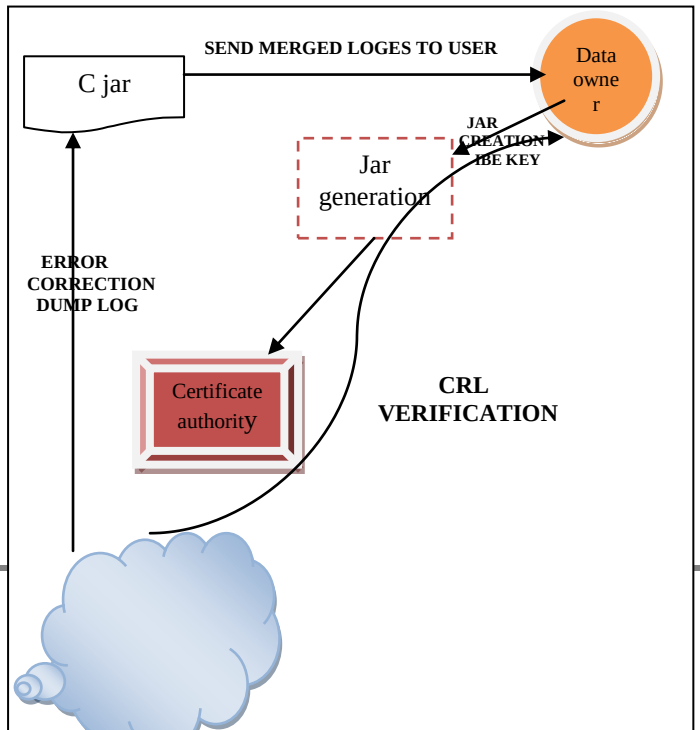
The log harmonizer is responsible for auditing. Being the trusted component, the log harmonizer generates the master key. It holds on to the decryption key for the IBE key pair, as it is responsible for decrypting the logs. Alternatively, the decryption can be carried out on the client end if the path 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 exist 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.

VII. SHARING 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. This IBE scheme is a Weil-pairing-based IBE scheme, which protects us against one of the most prevalent attacks. 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, we use OpenSSLbased 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.

The next point that the overhead can occur is during the authentication of a CSP. If the time taken for this authentication is too long; it may become a bottleneck for accessing the enclosed data. To evaluate this, the head node issued OpenSSL certificates for the computing nodes and we measured the total time for the OpenSSL authentication to be completed and the certificate revocation to be investigated. Considering one access at the time, we got that the authentication time averages around 920 ms which proves that not too much overhead is added during this phase. As of present, the authentication takes place each time the CSP needs to access the data. The performance can be further improved by caching the certificates. The time for authenticating an end user is about the same when we consider only the actions required by the JAR, viz. acquiring a SAML certificate and then evaluating it. This is due to both the OpenSSL and the SAML certificates are handled in a similar fashion by the JAR. When we consider the user actions (i.e., submitting his username to the JAR), it averages at 1.2 minutes.



IJDCST @October Issue- V-1, I-6, SW-31
ISSN-2320-7884 (Online)
ISSN-2321-0257 (Print)

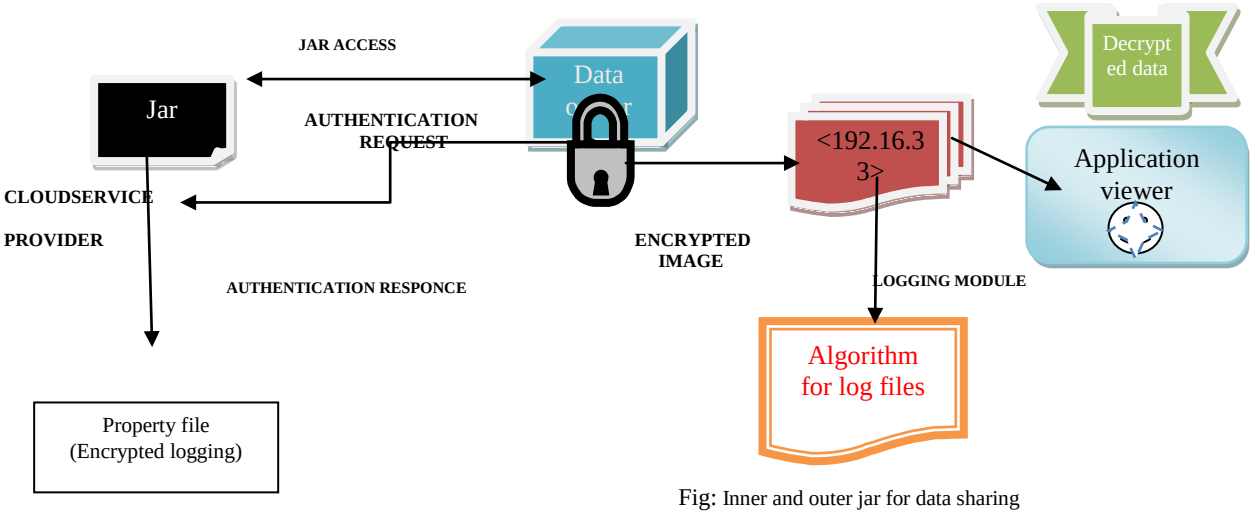


Fig: Input design for data sharing

VIII. AUTOMATIC LOGGING USING LOGGER

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. As shown in Fig. 2, 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 the server's URL or identity. For example, a policy may state that Server X is allowed to download the data if it is a storage server. As discussed below, the outer JAR may also have the access control functionality to enforce the data owner's requirements, specified as Java policies, on the usage of the data. A Java policy specifies which permissions are available for a particular piece of code in a Java application environment. The permissions expressed in the Java policy are in terms of File System Permissions. However, the data owner can specify the permissions in user-centric terms as opposed to the usualcode-centric security offered by Java, using Java Authentication

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.

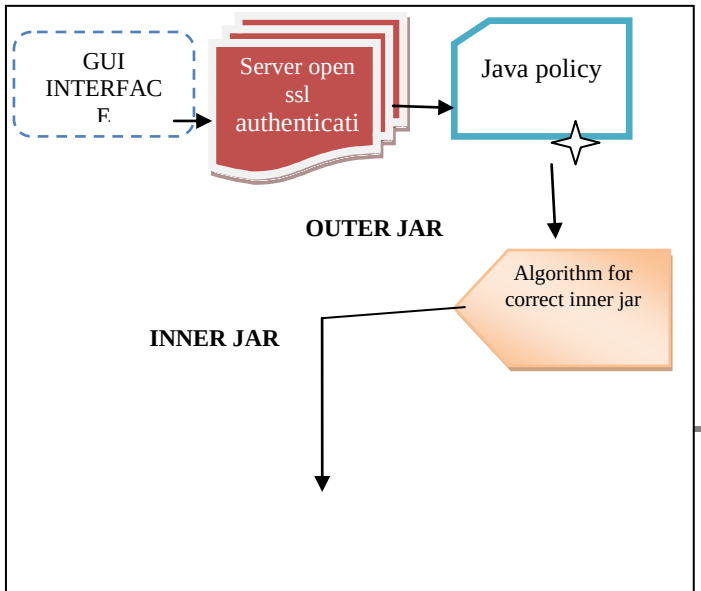


Fig: Inner and outer jar for data sharing

a. *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 LR ¼ hr1; . . . ; rki. Each record ri is encrypted individually and appended to the log file. In particular, a log record takes the following form:

ri ¼ hID; Act; T; Loc; hðð ID; Act; T; LocÞjri _ 1j . . . jr1Þ; sigi:

Here, ri 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 hðð ID; Act; T; LocÞjri _ 1j . . . jr1Þ 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. 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 Obj ID field is added to each record. Suppose that a cloud service provider with ID BABU, located in INDIA, read the image in a JAR file (but did not download it) at 2:32 pm on Nov 20, 2013. The corresponding log record is Raviteja, View, 2013-12-29 16:52:30,INDIA, 45rftT024g, r94gm30130ffi.

The location is converted from the IP address for improved readability. To ensure the correctness of the log records, we verify the access time, locations as well as actions. In particular, the time of access is determined using the Network Time Protocol (NTp) to avoid suppression of the correct time by a malicious entity. The location of the cloud service provider can be determined using IP address. The JAR can perform an IP lookup and use the range of the IP address to find the most probable location of the CSP. More advanced techniques for determining location can also be used. Similarly, if a trusted time stamp management infrastructure can be set up or leveraged, it can be used to record the time stamp in the accountability log [1]. The most critical part is to log the actions on the users' data. In the current system, we support four types of actions, i.e., Act has one of the following four values: view, download, timed access, and Location-based access. For each action, we propose a specific method to correctly record or enforce it depending on the type of the logging module, which are elaborated as follows:

- *View:* The entity (e.g., the cloud service provider) can only read the data but is not allowed to save a raw copy of it anywhere permanently.

For this type of action, the PureLog will simply write a log record about the access, while the AccessLogs will enforce the action through the enclosed access control module. Recall that the data are encrypted and stored in the inner JAR. 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. Further, to prevent the use of some screen capture software, the data will be hidden whenever the application viewer screen is out of focus.

- *CSP Download*: The entity is allowed to save a raw copy of the data and the entity will have no control over this copy neither log records regarding access to the copy. If PureLog is adopted, the user's data will be directly downloadable in a pure form using a link.

When an entity clicks this download link, the JAR file associated with the data will decrypt the data and give it to the entity in raw form. In case of Access Logs, the entire JAR file will be given to the entity. If the entity is a human user, he/she just needs to double click the JAR file to obtain the data If the entity is a CSP, it can run a simple script to execute the JAR. Timed_access. This action is combined with the view-only access, and it indicates that the data are made available only for a certain period of time. The Purelog will just record the access starting time and its duration, while the AccessLog will enforce that the access is allowed only within the specified period of time. The duration for which the access is allowed is calculated using the Network Time Protocol. To enforce the limit on the duration, the Access Log records the start time using the NTP, and then uses a timer to stop the access. Naturally, this type of access can be enforced only when it is combined with the View access right and not when it is combined with the Download. Location-based_access. In this case, the Pure Log will record the location of the entities. The AccessLog will verify the location for each of such access. The access is granted and the data are made available only to entities located at locations specified by the data owner.

- *Log Dependability:* In this section, we discuss how we ensure the dependability of logs. In particular, we aim to prevent the following two types of attacks. First, an attacker may try to evade the auditing mechanism by storing the JARs remotely, corrupting the JAR, or trying to prevent them from communicating with the user. Second, the attacker may try to compromise the JRE used to run the JAR files.
- *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. For recovering purposes, logger components are required to send error correction information to the harmonizer after writing each log record.

Therefore, logger components always ping the harmonizer before they grant any access right. If the harmonizer is not reachable, the logger components will deny all access. In this way, the harmonizer helps prevent attacks which attempt to keep the data JARs offline for unnoticed usage. If the attacker took the data JAR offline after the harmonizer was pinged, the harmonizer still has the error correction information about this access and will quickly notice the missing record. In case of corruption of JAR files, the harmonizer will recover the logs with the aid of Reed-Solomon error correction code. Specifically, each individual logging JAR, when created, contains a Reed-Solomon-based encoder. For every n symbols in the log file, n redundancy symbols are added to the log harmonizer in the form of bits. This creates an error correcting code of size 2n and allows the error correction to detect and correct n errors. We choose the Reed- Solomon code as it achieves the equality in the Singleton Bound, making it a maximum distance separable code and hence leads to an optimal error correction. The log harmonizer is located at a known IP address. Typically, the harmonizer resides at the user's end as part of his local machine, or alternatively, it can either be stored in a user's desktop or in a proxy server.

- *Correcting The Logs:* 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.

These tasks are carried out by the log harmonizer and the logger components in tandem with each other harmonizer is solely responsible for checking the integrity of the JRE on the systems on which the logger components exist before the execution of the logger components is started. Trusting this task to the

log harmonizer allows us to remotely validate the system on which our infrastructure is working. The repair step is itself a two-step process where the harmonizer first recognizes the Operating System being used by the cloud machine and then tries to reinstall the JRE. The OS is identified using nmap commands. The JRE is reinstalled using commands such as sudo apt install for Linux-based systems or $ <jre>.exe [lang=] [s] [IEXPLORER=1] [MOZILLA=1] [INSTALLDIR=:] [STATIC=1] for Windows-based systems.

The logger and the log harmonizer work in tandem to carry out the integrity checks during runtime. These integrity checks are carried out using oblivious hashing . OH works by adding additional hash codes into the programs being executed. The hash function is initialized at the beginning of the program, the hash value of the result variable is cleared and the hash value is updated every time there is a variable assignment, branching, or looping. As shown, the hash code captures the computation results of each instruction and computes the oblivious-hash value as the computation proceeds. These hash codes are added to the logger components when they are created. They are present in both the inner and outer JARs. The log harmonizer stores the values for the hash computations. The values computed during execution are sent to it by the logger components. The log harmonizer proceeds to match these values against each other to verify if the JRE has been tampered with. If the JRE is tampered, the execution values will not match. Adding OH to the logger components also adds an additional layer of security to them in that any tampering of the logger components will also result in the OH values being corrupted.

IX. PUSH AND PULL CONCEPT

A). 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. For the every periodical time the Cloud Server will send the access details of the user to the data owner. So that the Data Owner may able to know who're all the accessing their data at the particular time period. During the registration phase, the Data owner will ask by the Cloud Server whether they're choosing the push or pull method. 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.

B). Pull mode:

In the Pull method, the data owner has to send the request to the Cloud Server regarding the access details of their data up to the particular time. Then the Cloud Server will send the response to the Data Owner regarding the user's access details. 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.

ALGORITHM:
Step 1: LET TS (NTP) be the network time protocol
Step 2: pull=0
Step 3: rec :=(UID, OID, Access type, Result, Time, Loc)
Step 4: Curtime: =TS (NTP)
Step 5: Lsize: = sizeos(log)// current size of log
Step 6: if((cuttime – tbeg)<time)&&((size<size)&&(pull==0)
Then
Step 7: log: = log + ENCRYPT (REC)//ENCRYPT is the encryption function used to encrypt the record
Step 8: PING TO CJAR// send a ping to harmonizer to check if it is alive
Step 9: if (PING-CJAR) then
Step 10: PUSH RS (rec)// write the error correcting bits
Step 11: else
Step 12: EXIT (1)//error if no ping is received
Step 13: end if
Step 14: end if
Step 15: if((cut time –tbeg)>time)||(size>=size)||(pull!=0) then
Step 16: // check if is received
Step 17: if (PING – CJAR) then
Step 18: PUSH Log // write the log file to harmonizer
Step 19: RS(log) = NULL//reset the error correction records
Step 20: tbeg: =TS (NTP)//reset the tbeg variable
Step 21: pull=0
Step 22: else
Step 23: EXIT (1) // error if no ping is received
Step 24: end if
Step 25: end if

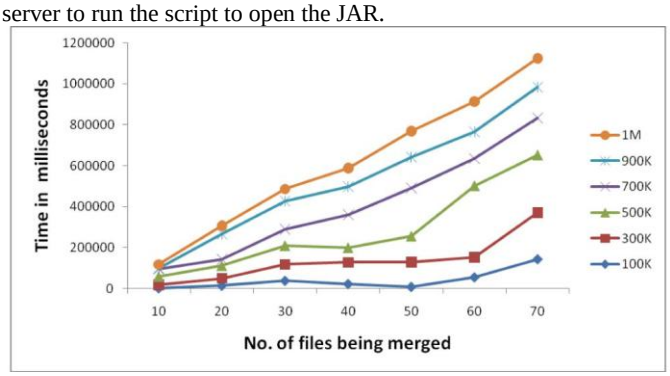
X. SECURITY ATTACKS

The most intuitive attack is that the attacker copies entire JAR files. The attacker 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. This is because each JAR is required to write redundancy information to the harmonizer periodically. If the JAR cannot contact the harmonizer, the access to the content in the JAR will be

disabled. Thus, the logger component provides more transparency than conventional log files encryption; it allows the data owner to detect when an attacker has created copies of a JAR, and it makes offline files unaccessible. 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. Given the ease of disassembling JAR files, this attack poses one of the most serious threats to our architecture. Since we cannot prevent an attacker to gain possession of the JARs, we rely on the strength of the cryptographic schemes applied to preserve the integrity and confidentiality of the logs. 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.1 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.

Middle attacks: An attacker may intercept messages during the authentication of a service provider with the certificate authority, and reply the messages in order to masquerade as a legitimate service provider. There are two points in time that the attacker can replay the messages. One is after the actual service provider has completely disconnected and ended a session with the certificate authority. The other is when the actual service provider is disconnected but the session is not over, so the attacker may try to renegotiate the connection. The first type of attack will not succeed since the certificate typically has a time stamp which will become obsolete at the time point of reuse. The second type of attack will also fail since renegotiation is banned in the latest version of OpenSSL and cryptographic checks have been added.

PROBLEM RESULT: 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. It is not surprising to see that the time to create a log file increases linearly with the size of the log file. Specifically, the time to create a 100 Kb file is about 114.5 ms while the time to create a 1 MB file averages at 731 ms. With this experiment as the baseline, one can decide the amount of time to be specified between dumps, keeping other variables like space constraints or network traffic in mind. The next point that the overhead can occur is during the authentication of a CSP. If the time taken for this authentication is too long, it may become a bottleneck for accessing the enclosed data. To evaluate this, the head node issued OpenSSL certificates for the computing nodes and we measured the total time for the OpenSSL authentication to be completed and the certificate revocation to be checked. Considering one access at the time, we find that the authentication time averages around 920 ms which proves that not too much overhead is added during this phase. As of present, the authentication takes place each time the CSP needs to access the data. The performance can be further improved by caching the certificates. The time for authenticating an end user is about the same when we consider only the actions required by the JAR, viz. obtaining a SAML certificate and then evaluating it. This is because both the OpenSSL and the SAML certificates are handled in a similar fashion by the JAR. When we consider the user actions (i.e., submitting his username to the JAR), it averages at 1.2 minutes. This set of experiments studies the effect of log file size on the logging performance. We measure the average time taken to grant an access plus the time to write the corresponding log record. The time for granting any access to the data items in a JAR file includes the time to evaluate and enforce the applicable policies and to locate the requested data items. In the experiment, we let multiple servers continuously access the same data JAR file for a minute and recorded the number of log records generated. Each access is just a view request and hence the time for executing the action is negligible. As a result, the average time to log an action is about 10 seconds, which includes the time taken by a user to double click the JAR or by a server to run the script to open the JAR.



We also measured the log encryption time which is about 300 ms (per record) and is seemingly unrelated from the log size. To check if the log harmonizer can be a bottleneck, we measure the amount of time required to merge log files. In this experiment, we ensured that each of the log files had 10 to 25 percent of the records in common with one other. The exact number of records in common was random for each repetition of the 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. 6. 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.

XI. SIZE OF COMPLETE JAR FILE

Finally, we investigate whether a single logger, used to handle more than one file, results in storage overhead. We measure the size of the loggers (JARs) by varying the number and size of data items held by them. We tested the

increase in size of the logger containing 10 content files (i.e.,images) of the same size as the file size increases. Intuitively, in case of larger size of data items held by a logger, the overall logger also increases in size. The size of logger grows from 3,500 to 4,035 KB when the size of content items changes from 200 KB to 1 MB. Overall, due to the compression provided by JAR files, the size of the logger is dictated by the size of the largest files it contains. Notice that we purposely did not include large log files (less than 5 KB), so as to focus on the overhead added by having multiple content files in a single JAR We investigate the overhead added by both the JRE installation/repair process, and by the time taken for computation of hash codes. The time taken for JRE installation/repair averages around 6,500 ms. This time was measured by taking the system time stamp at the beginning and end of the installation/repair. To calculate the time overhead added by the hash codes, we simply measure the time taken for each hash calculation. This time is found to average around 9 ms. The number of hash commands varies based on the size of the code in the code does not change with the content, the number of hash commands remain constant.

XII. CONCLUSION AND FUTURE ENHANCEMENT

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. We introduced 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.

In the future, we plan to refine our approach to verify the integrity of the JRE and the authentication of JARs [23]. 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

[1] P. Ammann and S. Jajodia, "Distributed Timestamp Generation inPlanar Lattice Networks," ACM Trans. Computer Systems, vol.1,p. 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.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.

[4] R. Bose and J. Frew, "Lineage Retrieval for Scientific Data Processing: A Survey," ACM Computing Surveys, vol. 37, pp. 1-28, Mar. 2005.

[5] 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.

[6] B.Chun and A.C. Bavier,"Decentralized Trust Management and Accountability in Federated Systems," Proc. Ann. Hawaii Int'l Conf.System Sciences (HICSS), 2004.

[7] OASIS Security Services Technical Committee, " Security Assertion Markup Language (saml) 2.0," http://www.oasis-open.org/ committees/tc ome.php?wg abbrev=security, 2012.

[8] R. Corin, S. Etalle, J.I. den Hartog, G. Lenzini, and I. Staicu,"ALogicforAuditing Accountability in Decentralized Systems," Proc. IFIP TC1 WG1.7 Workshop Formal Aspects in Security and Trust, pp. 187-201, 2005.

[9] B. Crispo and G. Ruffo, "Reasoning about Accountability within Delegation," Proc. Third Int'l Conf. Information and Comm. Security (ICICS), pp. 251-260, 2001.

[10] Y. Chen et al., "Oblivious Hashing: A Stealthy Software Integrity Verification Primitive," Proc. Int'l Workshop formation Hiding, F. Petitcolas, ed., pp. 400-414, 2003.

[11] 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.

[12] 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.

[13] 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.

[14] J.HightowerandG.Borriello,"Location Systems for ubiquitous Computing,"Computer,vol.34,no.8,pp.57-66, Aug. 2001.

[15] J.W. Holford, W.J. Caelli, and A.W. Rhodes, "Using Self-Defending Objects to Develop Security AwareApplications in Java," Proc. 27th Australasian Conf. Computer Science, vol. 26, pp. 341-349, 2004.

[16] TrustedJavairtualMachineIBM,http://www.almaden.ibm.com/cs/projects/jvm/, 2012.

[17] 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.

[18] 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.