Achieving Secure cloud Data Sharing

A.SUMANGALI
M.TECH IN C.S.E, AVSCET
VENKATACHALAM, NELLORE (Dt), A.P.
a.sumangali@gmail.com

T.V.SAMPATH KUMAR
ASSOC. PROF IN CSE,AVSCET
VENKATACHALAM, NELLORE (Dt), A.P.
tanjavuru.sampath@gmail.com

Abstract—Cloud computing enables highly scalable services to be easily consumed over the Internet on an as-needed basis. A major feature of the cloud services is that users' data are usually processed remotely in unknown machines that users do not own or operate. While enjoying the convenience brought by this new emerging technology, users' fears of losing control of their own data (particularly, financial and health data) can become a significant barrier to the wide adoption of cloud services. To address this problem, in this paper, we propose a novel highly decentralized information accountability framework to keep track of the actual usage of the users' data in the cloud.

In particular, we propose an object-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

LOUD computing presents a new way to supplement the current consumption and delivery model for IT services based on the Internet, by providing for dynamically scalable and often virtualized resources as a service over the Internet. To date, there are a number of notable commercial and individual cloud computing services, including Ama- zon, Google, Microsoft, Yahoo, and Salesforce. 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. While enjoying the convenience brought by this new technology, users also start worrying about losing control of their own data. The data processed on clouds are often outsourced, leading to a number of issues related to accountability, including the handling of personally identifiable information. Such fears are becom- ing a significant barrier to the wide adoption of cloud service.

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

To overcome the above problems, we propose a novel approach, namely Cloud Information Accountability (CIA) framework, based on the notion ofinformation accountability[44]. Unlike privacy protection technologies which are built on the hide-it-or-lose-it perspective, information account-ability focuses on keeping the data usage transparent and trackable. Our proposed CIA framework provides end-to-end accountability in a highly distributed fashion. One of the main innovative features of the CIA framework lies in its ability of maintaining lightweight and powerful account-ability that combines aspects of access control, usage control and authentication. By means of the CIA, data owners can track not only whether or not the service-level agreements are being 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.

Currently, we focus on image files since images represent a very common content type for end users and organiza-tions (as is proven by the popularity of Flickr [14]) and are increasingly hosted in the cloud as part of the storage services offered by the utility computing paradigm featured by cloud computing. Further, images often reveal social and personal habits of users, or are used for archiving important files from organizations. In addition, our approach can handle personal identifiable information provided they are stored as image files (they contain an image of any textual content, for example, the SSN stored as a .jpg file).

We tested our CIA framework in a cloud testbed, theEmulab testbed [42], with Eucalyptus as middleware [41].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 compro-mised Java Running Environment (JRE).

In summary, our main contributions are as follows:

1.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.

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

3. 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.

4. 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.

II. RELATED WORK

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.

A. *Cloud Privacy and Security*

Cloud computing has raised a range of importantprivacy and security issues. 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 of end users and then develop a privacy manager.

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 account-ability, 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.

B. *Other Related Techniques*

With respect to Java-based techniques for security, our methods are related to self-defending objects (SDO). 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.

III. 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. Alice, a professional photographer, plans to sell her photographs by using the Sky High Cloud Services. For her business in the cloud, she has the following requirements:

1. Her photographs are downloaded only by users who have paid for her services.
2. Potential buyers are allowed to view her pictures first before they make the payment to obtain the download right.
3. Due to the nature of some of her works, only users from certain countries can view or download some sets of photographs.
4. For some of her works, users are allowed to onlyview them for a limited time, so that the users cannot reproduce her work easily.
5. In case any dispute arises with a client, she wants tohave all the access information of that client.

6. 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.

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

1. The logging should be decentralized in order to adapt to the dynamic nature of the cloud. More specifically, log files should be tightly bounded with the corresponding data being controlled, and require minimal infrastructural support from any server.

2. Every access to the user's data should be correctly and automatically logged. This requires integrated techniques to authenticate the entity who accesses the data, verify, and record the actual operations on the data as well as the time that the data have been accessed.

3. Log files should be reliable and tamper proof to avoid illegal insertion, deletion, and modification by malicious parties. Recovery mechanisms are also desirable to restore damaged log files caused by technical problems.

4. Log files should be sent back to their data owners periodically to inform them of the current usage of their data. More importantly, log files should be retrievable anytime by their data owners when needed regardless the location where the files are stored.

5. The proposed technique should not intrusively monitor data recipients' systems, nor it should introduce heavy communication and computation overhead, which otherwise will hinder its feasibility and adoption in practice.

IV. CLOUD INFORMATION ACCOUNTABILITY

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

The Cloud Information Accountability framework pro-posed in this work conducts automated logging and distributed auditing of relevant access performed by any entity, carried out at any point of time at any cloud service provider. It has two major components: logger and log harmonizer.

A. Major Components

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 down-loaded by user X.

B. 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 as described. Using the generated key, the user will create a logger component which is a JAR file, to store its data items.

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

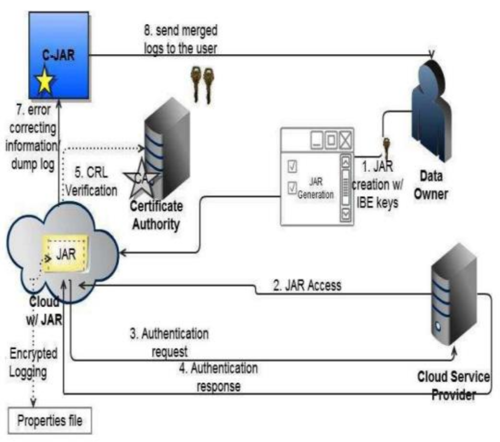


Fig 1: Overview of the cloud information accountability framework.

Once the authentication succeeds, the service provider (or the user) will be allowed to access the data enclosed in the JAR. Depending on the configuration settings defined at the time of creation, the JAR will provide usage control associated with logging, or will provide only logging functionality. As for the logging, each time there is an access to the data, the JAR will automatically generate a log record, encrypt it using the public key distributed by the data owner, and store it along with the data. The encryption of the log file prevents unauthorized changes to the file by attackers. The data owner could opt to reuse the same key pair for all JARs or create different key pairs for separate JARs. Using separate keys can enhance the security without introducing any overhead except in the initialization phase. In addition, some error correction information will be sent to the log harmonizer to handle possible log file corruption. To ensure trustworthiness of the logs, each record is signed by the entity accessing the content. Further, individual records are hashed together to create a chain structure, able to quickly detect possible errors or missing records.

The encrypted log files can later be decrypted and their integrity verified. They can be accessed by the data owner or other authorized stakeholders at any time for auditing purposes with the aid of the log harmonizer. Our proposed framework prevents various attacks such as detecting illegal copies of users' data. Note that our work is different from traditional logging methods which use encryption to protect log files. With only encryption, their logging mechanisms are neither automatic nor distributed. They require the data to stay within the boundaries of the centralized system for the logging to be possible, which is however not suitable in the cloud.

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

V. AUTOMATED LOGGING MECHANISM

We first elaborate on the automated logging mechanism and then present techniques to guaran-tee dependability.

A. The Logger Structure

We leverage the programmable capability of JARs to conduct automated logging. A logger component is a nested Java JAR file which stores a user's data items and corresponding log files. As shown in Fig. 2, our proposed JAR file consists of one outer JAR enclosing one or more inner JARs.

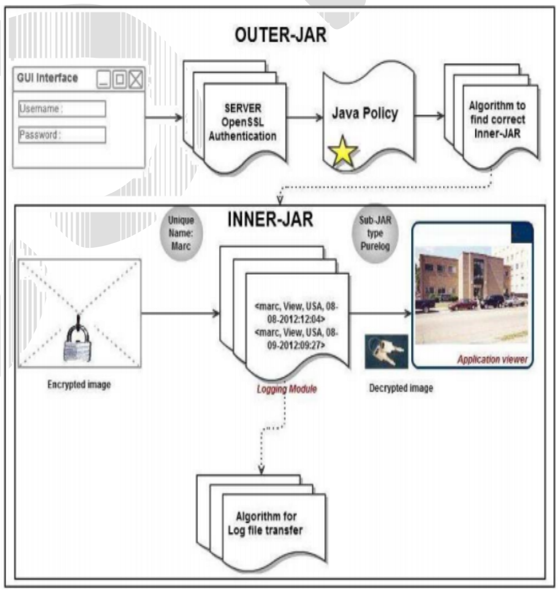

OUTER-JAR
GUI Interface
Username:
Password:
SERVER OpenSSL Authentication
Java Policy
Algorithm to find correct Inner-JAR
INNER-JAR
Unique Name: Marc
Sub-JAR type: Purelog
<marc, View, USA, 08-08-2012:12:04>
<marc, View, USA, 08-09-2012:09:27>
Encrypted image
Logging Module
Decrypted image
Application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

Fig. 2. The structure of the JAR file.

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 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.

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

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

1. PureLog. Its main task is to record every access to the data. The log files are used for pure auditing purpose.
2. AccessLog. It has two functions: logging actions and enforcing access control. In case an access request is denied, the JAR will record the time when the request is made. If the access request is granted, the JAR will additionally record the access information along with the duration for which the access is allowed.

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

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

B. 5.2 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:

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

Here, r i indicates that an entity identified by I D has per-formed an action Act on the user's data at time T at location Loc. The component
h((ID, Act, T, Loc)|ri-1| . . .| ri), 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 ObjI D field is added to each record. An example of log record for a single file is shown below.

Example 4. Suppose that a cloud service provider with ID Kronos, located in USA, read the image in a JAR file at 4:52 pm on May 20, 2011. The corresponding log record is
<Kronos, View, 2011-05-29 16:52:30,USA, 45rftT024g, r94gm30130ff>.
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) [35] 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 infrastruc-ture can be set up or leveraged, it can be used to record the time stamp in the accountability log . 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.,Acthas 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

1. **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. The content is displayed using the headless mode in Java on the command line when it is presented to a CSP.

2. **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 AccessLogs, 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.
3. **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 AccessLog 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.
4. **Location-based_access**. In this case, the PureLog 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.

C. *Dependability of Logs*

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, corrupt-ing 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.

i. *JARs Availability*

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

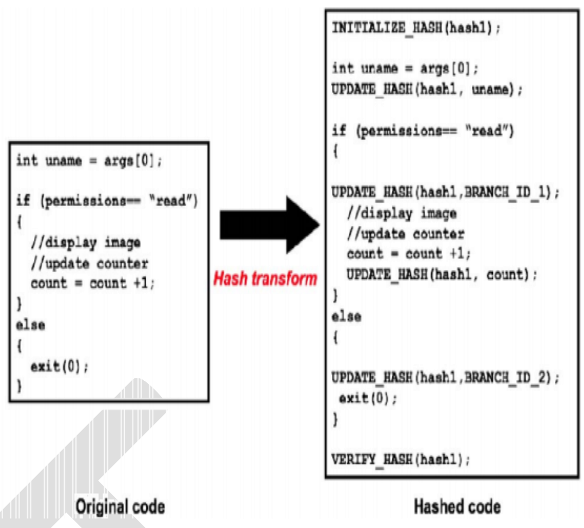
Each log harmonizer is in charge of copies of logger components containing the same set of data items. The harmonizer is implemented as a JAR file. It does not contain the user's data items being audited, but consists of class files for both a server and a client processes to allow it to communicate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

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 size2nand 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.

ii. Log Correctness

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.



```
int uname = args[0];

if (permissions== "read")
{
  //display image
  //update counter
  count = count +1;
}
else
{
  exit(0);
}
```

```
INITIALIZE_HASH(hash1);

int uname = args[0];
UPDATE_HASH(hash1, uname);

if (permissions== "read")
{

UPDATE_HASH(hash1,BRANCH_ID_1);
  //display image
  //update counter
  count = count +1;
  UPDATE_HASH(hash1, count);
}
else
{

UPDATE_HASH(hash1,BRANCH_ID_2);
 exit(0);
}

VERIFY_HASH(hash1);
```

Fig. 3. Oblivious hashing applied to the logger.

These tasks are carried out by the log harmonizer and the logger components in tandem with each other. The log 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 n map 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. An example of how the hashing transforms the code is shown in Fig. 3.

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.

REFERENCES

[1] J. N. Tsitsiklis, "Decentralized detection," in Advances in Signal Processing, H.V.Poor and J. B. Thomas, Eds.JAI Press, 1993, vol. 2, pp. 297–344.

[2] R. R. Tenney and J. Sandell, N. R., "Detection with distributed sensors," IEEE Transactions on Aerospace and Electronic Systems, vol. 17, pp. 501–510, Aug 1981.
.
[3] J.-F. Chamberland and V. Veeravalli, "Asymptotic results for decentralized detection in power constrained wireless sensor networks," IEEE Journal on Selected Areas in Communications, vol. 22, no. 6, pp. 1007 – 1015, 2004.

[4] S. Jayaweera, "Large sensor system performance of decentralized detection in noisy, bandlimited channels," in Proc. Vehicular Technology Conference, 2005, vol. 2, May 2005, pp. 1096– 1100.

[5] T. Quek, M. Win, and M. Chiani, "Distributed diversity in ultrawide bandwidth wireless sensor networks," in Proc. Vehicular Technology Conference, 2005, vol. 2, May 2005, pp. 1355– 1359.

[6] S. Wei, "Spreading sequence-based non-coherent sensor fusion and its resulting large deviation exponents," in Proc. Acoustics, Speech and Signal Processing, 2007, vol. 3, 2007, pp. III–177–III–180.

[7] M. Gastpar and M. Vetterli, "Source-channel communication in sensor networks," in 2nd International Workshop on Information Processing in Sensor Networks (IPSN'03), L. J. Guibas and F. Zhao, Eds. New York: Springer, April 2003, vol. 2634, pp. 162–177.

[8] D. Cassioli, M. Win, and A. Molisch, "The ultra-wide bandwidth indoor channel: from statistical model to simulations," IEEE Journal on Selected Areas in Communications, vol. 20, pp. 1247–1257, Aug. 2002.