Ensuring Accountability in Cloud for Portable Devices

Reshma Sadasivan, Sangeetha, Dr S.Karthik

Abstract-Cloud computing is about moving computing from the single desktop pc/data centres to internet. In cloud computing, user's data can be put it in the cloud storage and it can be access from the cloud, by the users whenever and wherever they needed. The major feature of the cloud is that user's data are processed in remote machines, which are unknown to the data owners. Here the security problems are raised. Users fear about their data control, so that they needed to account their data, which are stored in cloud. Accountability is the obligation to act as a responsible steward of the personal information of others, to take responsibility for the protection and appropriate use of that information beyond mere legal requirements, and to be accountable for any misuse of that information.

Keywords- Cloud Computing, Accountability

INTRODUCTION

Cloud Computing is a subscription-based service where you can obtain networked storage space and computer resources. In cloud computing model customers plug into the cloud to access IT resources which are priced and provided on- demand services. This cloud model composed of five essential characteristics, three service models and four deployment models. Users can store their data in cloud and there is a lot of personal information and potentially secure data that people store on their computers, and this information is now being transferred to the cloud. Here we must ensure the security of user's data, which is in cloud. Users prefer only the cloud which can be trusted. In order to increase the trust in cloud storage, the concept of accountability can be use. Accountability is likely to become a core concept in cloud that increase the trust in cloud computing. It helps to trace the user's data, protecting sensitive and confidential information, enhancing user's trust in cloud computing. In the context of cloud, accountability is a set of approaches to addresses two key problems. They are Lack of consumer trust in cloud service providers and Difficulty faced by cloud service providers with compliance across geographic boundaries.Accountability can also be implemented in portable devices. Using portable devices can increase the risk of data loss and data exposure. Accountability is likely to become a core concept in the cloud and to underpin new mechanisms that help increase trust in the cloud. Accountability is especially helpful for protecting sensitive or confidential information, enhancing consumer trust, clarifying the legal situation in cloud computing and facilitating cross border transfers of data. Accountability mechanisms complement preventive security and privacy mechanisms by detecting policy violations after they occur, identifying agents to blame for violations, and punishing the violators.

2. BACKGROUND

Accountability in cloud computing emerged due to the security issues in cloud. Cloud stores mass amount of user's data, there is a critical need to be secured that data. The owner of the data does not aware about where their data is stored and they do not have control of where data is placed. Here it explores the security challenges in cloud. Some of the security risks include secure data transfer, secure software interface, secure stored data, user access control, data separation. To promote privacy and security

• *Reshma Sadasivan is currently pursuing post graduation in computer science and ecngineering in Anna university, India, reshmasadasivancse@gmail.com*

concern of end users accountability mechanism is used. Here the basic concept is that user's private data are sent to the cloud in an encrypted form, and then with the encrypted data processing is carried out.

Accountability become a core concept in cloud that helps to increase trust in cloud computing. The term Accountability refers to a narrow and imprecise requirement that met by reporting and auditing mechanisms. Accountability is the agreement to act as a responsible proctor of the personal information of others, to take responsibility for protection and appropriate use of that information beyond legal requirements, and to be accountable for misuse of that information. Prospective accountability use preventive controls. Preventive controls for the cloud include risk analysis and decision support tools, policy enforcement, trust assessment, obfuscation techniques, identity management. Retrospective accountability use detective controls. Detective controls for the cloud include auditing, tracking, reporting and monitoring. Accountability in cloud focuses on keeping the data usage transparent and track able.

In cloud computing technology there are a set of important policy issues, which include issues of privacy,security,anonymity, government surveillance, reliability, and liability, among others . But the most important between them is security and how cloud provider assures it.

3 MODULE DESCRIPTION

3.1 Developing a cloud environment

Initially the basic network model for the cloud data storage is developed in this module. Four different network entities can be identified as follows: Client(Data Owner): an entity, which has large data files to be stored in the cloud and relies on the cloud for data maintenance and computation, can be either individual consumers or organizations; Cloud Storage Server (CSS): an entity, which is managed by Cloud Service Provider (CSP), has significant storage space and computation resource to maintain the clients' data; Certificate Authority: an entity, which has expertise and capabilities that clients do not have, is trusted to assess and expose risk of cloud storage services on behalf of the clients upon request. In the cloud paradigm, by putting the large data files on the remote servers, the clients can be relieved of the burden of storage and computation; Public User: The one who access the cloud data which is the private data of cloud data owners. The public data is stored in the cloud by data owners for business purposes it can be accessed by any user for their needs.

3.2 Proposing a Cloud Information Accountability (CIA) framework

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

correction information combined with the encryption and authentication mechanism provides a robust and reliable recovery mechanism, therefore meeting the third requirement. The 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 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. 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.

3.3 Data Access in Cloud Information Accountability

Develop the JAR file includes a set of simple access control rules specifying whether and how the cloud servers and possibly other data owners 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, they use CA wherein a trusted certificate authority certifies the CSP. In the event that the access is requested by a user, they employ authentication, wherein a trusted identity provider issues certificates verifying the user's identity based on his username.

Once the authentication succeeds, 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.

3.4 Develop Automated Logging Mechanism

In the automated logging mechanism, the main responsibility of the JAR is to handle authentication of entities which want to access the data stored in the JAR file. In this 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. Each 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. They support two options;

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

The most critical part is to log the actions on the users' data. In the current system, 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, here propose a specific method to correctly record or enforce it depending on the type of the logging

module. And finally designed the end to end mechanism for auditing using push or pull mode configure for log files send to the data owner.

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. Here support two options: PureLog: Its main task is to record every access to the data. The log files are used for pure auditing purpose. AccessLog: It has two functions: logging actions and enforcing access control. In case an access request is denied, the JAR will record the time when the request is made. The log files are used for pure auditing purpose. 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.

3.5 Proposing Accountability in Portable Devices

While the specification concludes with claims defining the features of the invention that are regarded as novel, it is believed that the invention will be better understood from a consideration of the following description in conjunction with the drawing figures, in which like reference numerals are carried forward. A brief description of the prior art is also thought to be useful. The present invention solves the problem of downloading portable applications and authenticating their source onto client device with limited computing resources by creating a signed application descriptor file (ADF), and a developer descriptor file (DDF). The ADF is a file that describes the portable application in terms of the computing resources it requires, and can be loaded onto the client device first so that
the client device can determine whether or not it has sufficient resources, or it can let the user of the client device determine if there are sufficient resources. The ADF file is signed by the developer of the corresponding application using a certification authority, which is a well known and trusted signing authority. Attributes in the signed ADF correspond to those of the application so that if the user of the client device decides to load the application, the application can be authenticated against the signed ADF.

The DDF is associated with a particular application software developer and specifies the general access control related information assigned to the developer. For example, a DDF may restrict the kind of application libraries that applications developed by the developer can use, or the security domain to which the developer belongs.

In particular, a network client device, such as a mobile communication device, connects with a distribution server over one or more bearer networks. Typically the bearer network includes a TCP/IP network, and for public distribution of software, it includes the Internet. However, numerous private networks are connected to the Internet through various gateways and portals, including many wireless mobile communication networks. Indeed, the present invention is suited particularly well to use on mobile communication devices such as Internet capable mobile or cellular radio telephones.

These devices may use what is referred to as a "micro browser" to view information, or "content", placed on the Internet and other networks accessible by the device, as well as execute portable code. As with general purpose computers, there is a desire to load portable applications onto these devices. Developers of portable applications provide the application on a database of the distribution server. Client devices access the distribution server over the network and receive the desired portable code or portable application over the connection. This is one way which JAVA code sections, such as applets, are distributed.

Client network device and its computing resources. In this instance, the client device is receiving an application file, which includes a signed ADF and the application code. The application code may be, for example, a JAVA archive (JAR) file. These two parts maybe transferred separately or together. The signed ADF prescribes the security domain of the application when loaded into the client device. Essentially, the security domain determines which of the client device's resources the application will be allowed to access when running in the virtual machine environment. In the preferred

embodiment, the application is received in the form of byte code which the virtual machine executes. The virtual machine only allows the application to access the resources permitted, as dictated by the signed ADF. The resources include other processes, classes, and methods, as well as certain hardware components such as volatile and non-volatile memory space. The signed ADF is substantially smaller than the presently used signed JAR file format because a signed JAR file includes all applications-related files, and is more desirable for use with computing devices with relatively limited resources.

. The signed ADF includes an application descriptor file, a file hash of the JAR file, a developer descriptor file (DDF), a developer certificate, a time stamp, and a developer signature. ADF describes the resources which are required by the client device, and may include a security policy file or a license policy file, or both. The ADF contains a pointer to the network location of the application in the file hash, an indication of the amount of memory space required to execute the application, and the environment necessary for execution. The security policy contains the information regarding which resources the application needs permission to use, as well as the names of files the application may create, and the network addresses it may need to access.

The license policy may be used to set how the application may be used, such as whether it as a finite number of uses, or a finite period of time, whether it may be transferred to other users, and so on. The file hash is a hash of the JAR file , and is the result of a cryptographic method which produces a small digest that can be used to authenticate a larger file (in this case a JAR file), as is known in the art. For additional security, more than one hash may be used and included in the signed ADF using different hash algorithms, such as SHAT, MD5, or others.

The DDF also contains information about the JAR file, and may also include information regarding the developer. The developer's certificate is a certificate issued to the developer and includes the developer’s public key so that the certificate may be authenticated by the client device. It also contains information about the identity of the developer, the validity period, the issuing certificate authority, the issuing date, and so on, to be used in the authentication process. The time stamp is a signed time stamp. It is provided by a trusted source, such as a certificate authority or perhaps the client device's subscriber network operator. By providing a signed timestamp, the client device can determine when the application was signed, and if that time stamp is authentic. Finally, the developer's signature is concatenated onto the other data structures. The developer's signature is allows the client device to authenticate the ADF.

. The developer sends a request, containing the hash for the ADF and the developer certificate, to a code signing authority , and requests a signed timestamp. The code signing authority is a trusted entity, such as, for example, VeriSign, Inc., or the client's network operator. It should be noted that the certificate authority and the code signing authority are not necessarily the same entity. A certificate authority manages certificates, while a code signing authority verifies developer certificates and signs ADF hash files and timestamps it. The developer generates and sends a hash of the ADF to the code signing authority and receives a signed timestamp back from the certificate authority.

The developer then concatenates the ADF, the hash of the JAR file, the DDF, the developer's certificate, and the signed time stamp together. The developer's certificate contains a hash of the DDF. The developer signs the concatenated file , by adding the developer's digital signature , and the ADF is fully signed. The signed, concatenated file is the signed ADF, and is then placed on a distribution server , along with the application code or JAR file. The network address of the signed ADF is then made available so that client devices can download it to begin the installation and authentication procedure for the JAR file. In the preferred embodiment, the developer uses a software developer's kit (SDK), which is a set of software creation tools distributed by, for example, the creator of the virtual machine environment that runs on the client device. The SDK automates this whole procedure.

The developer's browser, or similar client application, first sends a certificate request to a code signing certificate authority server, which may be operated by the same certificate authority referred to above. The certificate authority sends the developer information to a

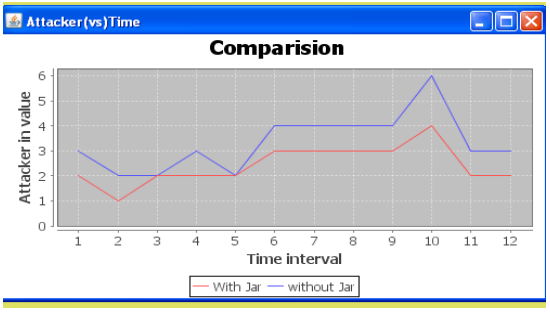
developer administration server. This entity may be, for example, the operator of the client's home network or network service provider. For example, in the case of the client device being a wireless mobile communication device, the developer administrator may be the wireless service provider that provides the wireless communication service. The developer administrator returns a developer descriptor file (DDF) to the code signing certificate authority , which becomes part of the signed ADF, as described above in reference to FIG. 3. In the preferred embodiment, the developer's certificate will contain a hash of the DDF. The preferred format for the certificate is a wireless transport layer security (WTLS) certificate because it is smaller than, for example, an X.509 certificate. The certificate authority then forwards a developer certificate and the DDF, preferably both text encoded, to the developer's computer, such as, for example, by email. The text encoded information is easy to transfer as an email enclosure.

After the signed ADF has been created and placed on a distribution server, client devices can download the signed ADF and the application or JAR file.The client device has had a code signing certificate authority public key and a time stamping root key placed in the client device. The time stamping root key is a public key used for authenticating the signed time stamp. To begin the method, the client device transmits a request to a distribution server for the application. The distribution server transmits the signed ADF for the desired application, which is received by the client device. Preferably the signed ADF contains an application descriptor file, file hash of the application code, developer descriptor file, developer certificate, signed time stamp, and the developer certificate.

These transactions take place using known network protocols, such as TCP/IP. Upon receiving the signed ADF, the client device verifies the developer certificate with the code signing certificate authority's public key. The client may also authenticate the signed time stamp with the time stamping root key. The verified timestamp is used to check whether the ADF file is signed within the valid period of the developer certificate. Both of these must be verified. If it was not already received, the client device obtains the network location of the application code or JAR file, and transmits a request to the server, specifying the particular application desired . Although shown here a being on the same distribution server as the signed ADF, the application may be located on a different server. The server transmits the application code to the client device . Upon receiving the application code, the client device compares it to the parameters in the signed ADF. Specifically, it compares the hash received in the signed ADF with the hash of the application, and may also verify attributes such as file size. The hash of the JAR file may be produced by the client device, and compared to the hash received in the signed ADF. If the hash of the application received in the signed ADF matches the hash of the received application file, the client device loads the application into the virtual machine environment for execution according to the security and license policies, if any were present in the ADF.

Thus, the invention avoids the use of relatively large files and certificates for authenticating the application code, as is used with more powerful, general purpose computers. A typical certificate will be 100-500 bytes, the file hash about 20 bytes, the DDF about 100-200 bytes, and of course the JAR file can be very small to very large, depending on the application. The use of a signed ADF allows devices with relatively limited resources to easily authenticate the trustworthiness of an application, and to set appropriate permissions for the various resources. The security is accomplished by providing the client device with a set of keys initially, such as the code signing certificate authority key and a time stamping root key. Furthermore, a developer provides their public key in the signed ADF so that client devices can use them to further establish a trusted chain. A hash of the application, and preferably a signed hash, is used so that it may be compared to the application once it is received at the client device. A security policy file and a license policy file may be provided to describe what resources the application will need, what it will create, and the limitation on the use of the application as well as the transferability of the application.

3.6 Performance evaluation



First examine the time taken to create a log file and then measure the overhead in the system. With respect to time, the overhead can occur at three points: during the authentication, during encryption of a log record, and during the merging of the logs. Also, with respect to storage overhead, we notice that our architecture is very lightweight, in that the only data to be stored are given by the actual files and the associated logs. Further, JAR act as a compressor of the files that it handles and can evaluate our performance by the following parameters they are Log Creation Time, Time Taken to Perform Logging, Log Merging Time Size of the Data JAR Files.

While the preferred embodiments of the invention have been illustrated and described, it will be clear that the invention is not so limited. Numerous modifications, changes, variations, substitutions and equivalents will occur to those skilled in the art without departing from the spirit and scope of the present invention as defined by the claims.

4 CONCLUSION

In cloud computing, we can ensure the trustiness of cloud by using accountability. CIA can provide access and usage control with authentication. Accountability is used for tacking the data that means tracing the control of data. We can use JAR files that contain user's data and their policies. JAR file can be authenticated; so that it allows the developer to develop more powerful applications even modify the code and audit the code of the copied code by the attacker. It includes advantages are can able to distribute applications to many different mobile devices, information gathering capabilities is high and portability. This paper discussed the accountability in cloud computing and that can be applicable in portable devices

REFERENCES

[1] Andreas Haeberlen,"A case for accountable cloud", Max Planck Institute for Software Systems (MPI-SWS) [[2] Cloud Security Alliance. (2010). CloudAudit The Automated Audit, Assertion, Assessment, and Assurance API) Available: http://cloudaudit.org/

[2] Sonam Chugh and Sateesh Kumar Peddoju, "AccessControlBased Data Security in Cloud Computing", Vol. 2, Issue 3, May-Jun 2012, pp.2589-2593

[3] Eric Keller, Ruby B. Lee and Jennifer Rexford," Accountability in Hosted Virtual Networks",

[4] Jinhui Yao and Chen Wang,"Accountability as a service for the cloud"

[5] Veerraju Gampala, Srilakshmi Inuganti, Satish Muppidi ", Data Security in Cloud Computing with Elliptic Curve Cryptography ", International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-2, Issue-3, July 2012

[6] V.Sathya Preiya, 2 R.Pavithra 3 Dr. Joshi,"SECURE ROLE BASED DATA ACCESS CONTROL IN CLOUD COMPUTING",International Journal of Computer Trends and Technology- May to June Issue 2011

[7] Ms.P.M.kiruthika, Ms.T.Amirtha and Mrs.R.Deepa,"A framework for accountability and trust in cloud computing", International Journal of Communications and EngineeringVolume 01– No.1, Issue: 03 March2012

[8] Kyriacos E. Pavlou and Richard T. Snodgrass,"Achieving database information accountability in the cloud",Department of Computer Science, The University of Arizon

[9] Ryan K L Ko , Peter Jagadpramana , Miranda Mowbray Siani Pearson ," TrustCloud: A Framework for Accountability and Trust in Cloud Computing"

[10] S.Sajithabanu and Dr.E.George Prakash Raj," Data Storage Security in Cloud", IJCST Vol. 2, Iss ue 4, Oct . Dec. 2011.

[11] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling Public Verifiability and Data Dynamics for Storage Security in CloudComputing," Proc. 14th European Symp. Research in Computer Security (ESORICS '09), pp. 355-370, 2009

[12] D. J. Weitzner, H. Abelson, T. Berners-Lee, J. Feigenbaum, J. Hendler, and G. J. Sussman, "Information accountability," Communications of the ACM, vol. 51, no. 6, pp. 82–87, June 2008.

[13] A. R.Yumerefendi, J. S. Chase. Strong accountability for network storage. ACM Transactions on Storage, volume 3, issue 3, article No. 11, 2007

[14] Y. Zhang, K. J. Lin, T. Yu. Accountability in service-oriented architecture: computing with reasoning and reputation. In proc. IEEE International Conference on e-Business Engineering, pp. 123-131,2006.

[15] 104th United States Congress, "Health Insurance Portability and Accountability