ISSN:0975-9646

An Effective Secured Cloud Based Log Management System Using Homomorphic Encryption.

A. Murugan
Assistant Professor,
Department of Computer Science and Engineering,
SRM University,
Tamil Nadu, India- 603203

Tarun Kumar Kala
Department of Computer Science and Engineering
SRM University,
Tamil Nadu, India- 600029.

Abstract- For any organization which depending on sensitive data processing it is very important to maintain information about every event occurring within the organization's system or network, these event details are called as log files. The log files will able to record user activities, troubleshooting problems and any policy violations. As this log files plays vital role and also contains sensitive information, it should be maintained very securely. The capital expenses will be very immense to maintain log data for many organizations over long period. The alternative economic solution is to maintaining log files over a cloud database. Log files with sensitive information over a cloud environment will leads to challenges about confidentiality and privacy. In this paper, we propose effective secure cloud-based log management and also the use of homomorphic encryption as a solution for dealing the issues to access a cloud based data storage.

1. INTRODUCTION

A Log file is to record the detailed information of every event of a system or application running in an organization[1]. Log files are highly useful to find any operational problems shortly they have occurred and also able to useful information to resolve such problems. Log files are also helpful in identifying the security incidents, fraudulent activities, and policy violations. Logs contains of sensitive information of an organization so there should be some protection from malicious attacker.

1.1 Log Generation and Maintenance

There are some traditional protocols based on syslog to generating logs. Some security extensions proposed such as reliable delivery of syslog [2], forward integrity for audit logs [3], syslog-ng [4], and syslog-sign [4], provides either partial protection, or do not protect the log records from malicious attacks. On other side the count, size, and format of computer security logs have increased rapidly, which needs of computer security log management—the process for generating, transmitting, storing, analyzing, and disposing of computer security log data. Organizations facing a major problem with log management are to effectively balancing a limited quantity of log management resources with a continuous supply of log data. For any organization log generation and maintenance can be complicated by several factors, including a high number of log sources; inconsistent log content, formats, and timestamps among sources; and increasingly large volumes of log data. Log management also need to achieve some properties such as confidentiality, integrity, and availability of logs. Deploying secure logging information to meet all the above challenges cloud storage is best economical alternative.

1.2 Delegating Logs to Cloud storage

It is highly advantage to use cloud as a medium of storage with universality of accessing platforms and mostly of minimal hardware requirements on user end. In cloud environment everything is delivered as a service (Xaas)[6] and there are three main service model :

1. *Software as a Service* (*SaaS*) - This service is to delivering software over the Internet, consisting of software running on the provider's cloud infrastructure, delivered to a single or multiple clients on demand through a thin client (e.g. browser) .
2. *Platform as a Service* {*PaaS*} - This provides the flexibility for a client to build—develop, test and deploy applications on the provider's platform. In this service *PaaS*-hoster provides the infrastructure besides *PaaS*-provider provides the development tools and platform to the end *PaaS* user.
3. *Infrastructure as a Service* {*IaaS*} -This offers on demand access to resources such as networking, servers and storage, can be accessed with a service API.

There are four deployment models of cloud computing based on ownership over infrastructure, this is where the security issues raises.

1. *The Public Cloud* - This is the basic view of cloud computing in general. It is usually owned by a large organization makes its infrastructure available to the general public over the Internet by a multi-tenant model on a self-service basis. For the end user this is the most cost-effective model leading to substantial savings with attendant privacy and security issues since the physical location of the provider's infrastructure usually traverses numerous national boundaries.
2. *The Private Cloud* – In this model cloud infrastructure is under a single tenant environment which may be managed by the tenant organization or by a third party within or outside the tenant premises. This model costs more than the previous public cloud.
3. *The Community Cloud* – This model refers to a cloud infrastructure shared by multiple organizations belongs to a specific community, which may be managed by any one of the organizations or a third party.
4. *The Hybrid Cloud* – This model is a combination of any two (or all) of the three models discussed above

However user can choose any of the model based on his requirements but the major security issues raises in cloud computing are availability, data security, third-party control, Privacy and legal issues based on the model, which can be discussed in detail in coming sections.

1.3 Secure logging and challenges over a cloud.

In light of the above issues in a cloud environment, to provide a secure logging as service there are some desirable properties to achieve:

1. *Availability* - This defines that the logs over cloud storage must be available whenever it is required. This is a major concern for cloud based database users.
2 *Verifiability*: This property is to check all entries in the log are present and have not been modified. Each entry must be verified its authenticity independent of others, and also the entries must be linked together in a way that makes it possible to determine whether any entries are missing(forward integrity[3]).
3 *Privacy*: Log records over a cloud will be distributed globally which raises issue of data exposure and privacy.
4 *Confidentiality*: Log records should not be casually searchable to gather sensitive information. Legitimate search access to users such as auditors or system administrators should be allowed. In addition, since no one can prevent an attacker who has compromised the logging system from accessing sensitive information that the system will put in future log entries, the goal is to protect the pre-compromised log records from confidentiality breaches.

To meet all the above challenges a secure way log generation--(extracting, transforming, encrypting) should be needed. The typical choice to achieve all the properties is applying Homomorphic encryption in log generation.

2. RELATED WORK.

If the log records are encrypted and stored in the cloud that would effectively solve issues. However, if a user wants to access the log data it needs to decrypting log data first, or shipping it entirely back to the user for computation, So the cloud provider thus has to decrypt the data first perform the computation then send the result to the user which raises the issue of privacy and confidentiality. This is where the homomorphic encryption plays its role. Using homomorphic encryption user able to carry out an arbitrary computation on the hosted log records without exposing log data to the cloud provider - computation can be done on encrypted data without decryption. In homomorphic encryption schemes it allows the transformation of cipher texts *C(m)* of message m, to cipher texts C(*f(m))* of a computation/function of message m, without disclosing the message.

2.1 System architecture

The complete system architecture of a cloud based log management system is shown in fallowing Fig. 1..

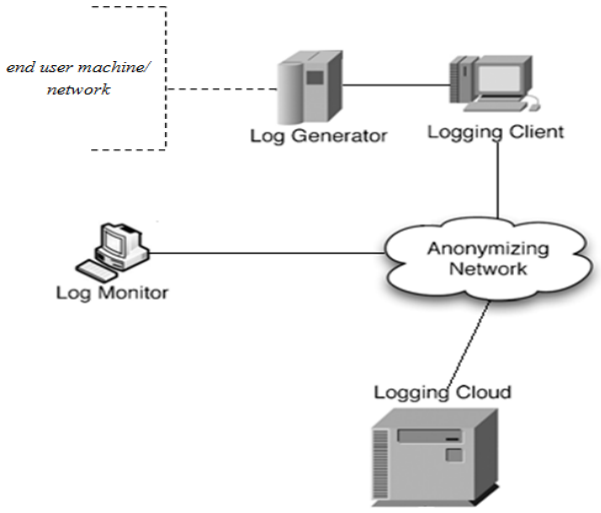

Figure 1. System architecture of a cloud based log management system example.

There are four major functional components in a cloud based logging management system.

1) *Log Generator*: This is an computing device logging capability that can able to generate log data over organizations system or network. These generators stores log data temporarily later they are pushed to the logging client.
2) *Logging Client or Logging Relay*: This is to collect log records generated by one or more log generators, and prepares the log batches which it can be pushed to the cloud for long term storage.
3) *Logging Cloud*: The logging cloud is a remote storage and maintenance service to log data from multiple organizations which are subscribed to service providers. These cloud can perform queries--- delete log data and log rotation only on authorized requests of organizations.
4) *Log Monitor*: The major functionalities are to monitor, review log data, generating queries to retrieve log data and analyse log data retrieved log data from the cloud.

2.2 Basic Process of log preparation.

This log files preparation protocol contains three sets of keys—A_i and X_i for ensuring integrity, and K_i ensuring confidentiality. These keys are derived in sequential manner starting with three master keys A_0, X_0, and K_0. We use a secret key cryptosystem to provide integrity and confidentiality

The protocol begins with three randomly generated master keys—A_0 and X_0, and K_0. The Log records will be extracted from authenticated client machine will be taken as series of messages $L_1, L_2 \ldots L_n$. value n is determined randomly.

1) Before any log data arrives at the logging client, the logging client creates a special first log entry L_0 = [TS, log-Initialization, n]. It then encrypts this log entry with the key K_0 and computes the message authentication code $MAC_0 = HA_0[EK_0 [L_0]]$ for the encrypted entry with the key A_0. The client adds the

resulting first log entry for the current batch—[$EK0[L_0]$,MAC_0]—to the log file.

2) The logging client then computes new set of keys $A_1 = H[A_0]$, $X_1 = H[X_0]$ and $K_1 = H[K_1]$, securely erases the previous set of keys and waits for the next log message to arrive.
3) When the first log message L_1 arrives, the logging client creates a record $M_1 = L_1 \parallel HA_1\ [EK_0\ [L_0]$. It encrypts M_1 with the key K_1, and creates a message authentication code for the resulting data as $MAC_1 = HA_1\ [EK_1\ [M_1]$. It also computes an aggregated message authentication code $MAC'_1 = H^n_{XI}[MAC_0 \| MAC_1 \parallel n]$. The log batch entry is $[EK_1\ [M_1]$,$MAC_1]$. It then creates the next set of keys $A_2 = H[A_1]$, $X_2 = H[X_1]$ and $K_2 = H[A_1]$ and securely deletes A_1 and K_1.
4) For every new log data L_i that the logging client subsequently receives, it creates log file entries $[EK_i\ [M_i]$,$MAC_i]$, where $M_i = L_i \parallel MAC_i - 1$ and $MAC_i = HA_i\ [EK_i\ [M_i]]$. It also creates the aggregated message authentication code $MAC'_i = H^{n-i+1}{}_{xi}[MAC_{i-1} \parallel MAC_i] \parallel n - i + 1$. Once MAC'_i has been generated, MAC^{i-1} is securely deleted. The client finally creates new keys $A_{i+1} = H[A_i]$, $X_{i+1} = H[X_i]$ and $K_{i+1} = H[K_i]$ and securely erases the keys A_i, X_i and K_i.
5) After the client creates the last log entry M_n for the current batch from the last log data L_n, it creates a special log close entry $LC = [EK_{n=1}[TS, log\text{-}close \parallel MAC_n\ ],HA_{n+1}\ [EK_{n+1}[TS, log\text{-}close \parallel MAC_n]]]$, and an aggregated message authentication code MAC'_{n-1}. It then securely erases the three keys used in this step and uploads the resulting log batch and aggregated message authentication code to the logging cloud as one unit.

2.3 Achieving Anonymity

While the logging client uploads data to the logging cloud, logging client has to be authenticated. Achieving anonymity means the logging client should not be identified over any of its transactions including the authentication process. To maintain privacy and accountability, we are using *k*-times anonymous authorization protocol[7]. There are four different protocols for anonymous upload, retrieval and deletion of log data. Here assume that the logging client and also may be logging cloud or other adversaries knows the public key of the entity.

1) *Anonymous Upload-Tag Generation*: The log records must be indexed by a unique key value to retrieve it later, where the key value should not be exposed. This upload-tag is will be generated by the logging client in cooperation with the log monitor.
2) *Anonymous Upload*: After the logging client authentication to the logging cloud in an anonymous manner, the logging client sends a message contains the upload-tag, a delete tag and a batch of previously prepared log data. The delete-tag is used later to delete or rotate the log data by any entity authorized by the logging client if they need to do so.
3) *Anonymous Retrieve*: This protocol is to download log data needs to send a retrieve request anonymously together with the upload-tag respective to the desired log data. The logging cloud process request gets and sends data to the requester over the anonymous channel. There is no need of authenticating the requester as log batches being encrypted that can be used by clients having valid decryption keys.
4) *Anonymous Delete*: To delete log data over cloud, the requester needs to send a delete message to the logging cloud. Then the logging cloud throws a challenge for authentication to delete by presenting a correct delete tag, which contains information that is used by the logging cloud to index the message.

In a cloud environment the customer is unaware of where his request is physically executed. The customer has to trust the resource provider, because request is executed entirely under the control of the resource owner and thus cannot rely on the security and confidentiality of the remote resource. There is a need for a mechanism to operate on encrypted data, which can be achieved by applying Homomorphic Encryption (HE)[8].

2.4 Applying Homomorphic Encryption (HE).

A homomorphism is a structure-preserving transformation between two sets, where an operation on two members in the first set is preserved in the second set on the corresponding members[9].

Let P and C be sets denoting the plain-text space and the cipher-text space, respectively where p1, p2 ϵ P, t a transformation between the two sets with its reverse function t' and an operation $\oplus$. The system is said to be an homomorphism, if $\forall p1, p2\ \epsilon P, (p1 \oplus p2) = t'(t(p1) \oplus t(p2))$. If there are two functions $\oplus$ and $\otimes$, such that $\forall p1, p2\ \epsilon P, (p1 \oplus p2) = t'(t(p1) \oplus t(p2))$, $(p1 \otimes p2) = t'(t(p1) \otimes t(p2))$ this is called an algebraic homomorphism [6]. The range of transformation of the two member's p1 and p2 will be C, the result of decryption back into the range of P.

To understand the homomorphic scheme lets fallows an example, where k $\in$ N be a large prime integer as a secret key and *x* and *y* be two arbitrary integers with $(x, y) < k \in N$. Then encryption of x can be done as $x' = x + (r_1 * k)$ with $r \in N$ being a large random integer. To get original text *x, x'* can be the decrypted as *x'* mod *k*. here we can then perform an encrypted addition as $(x'+y')$ which implies $(x'+y') = (x+(r_1*k))+(y+(r_2*k))$ and $x+y+(r_1+r_2)*k$ and when decrypted mod *k* yields $(x+y)$. The multiplication is performed as $(x'* y') = (x + (r_1 * k)) * (y + (r_2 * k))$ and $x*y+x*(r_2*k)+y*(r_1*k)+(r_1*r_2)*k*k$ mod $k = (x*y)$.

Example: $x = 5; y = 4; k = 23; r_1 = 6; r_2 = 3;$
$x' = 5 + (6 * 23) = 143;$
$y' = 4 + (3 * 23) = 73;$
$x' + y' = 216; 216 \bmod 23 = 9;$
$x' * y' = 10439; 10439 \bmod 23 = 20$

Though above satisfies the homomorphic encryption scheme but there is a drawback, with every operation the intermediate result grows towards the modulus and the result exceeds the prime modulus k, then the decryption fails. To overcome this we need to do normalization (would be any function) that can minimize the remainder mod k of the result while preserving the parity mod k, which can be achieved by using advanced homomorphic schemes, such as [10] and [11], base on Gentry's approach of bootstrapping a fully homomorphic.

3. System Analysis

The test system consists of two identical machines with a 2.13 GHz Intel Xeon E7 CPU and 128GB memory. Both machines were running Fedora 15 64-bit Linux with 2.6.42.7 kernel and were stationed on the same LAN. The first machine was running the logging client application, while the second machine was running the syslog-ng application that was configured to create fixed-size (100 bytes) log records

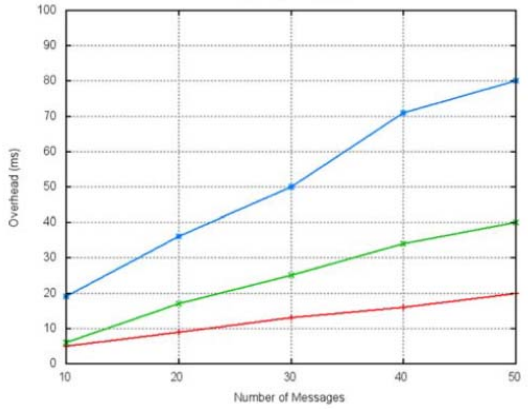

Figure 2. analysis graph for log management process.

For the log batches of sequential incremental in size, in first upload and retrieve will be achieved with best performance but further experiments it will gradually increases the overhead computation of overall process which is shown in Fig 2.

4. Summary

Log management system is very important for proper functioning of any organization's information processing systems. It is also very expensive to maintaining log data of many organizations over long periods of time. Delegating log data to cloud based data storage is an economic alternative for this.

The ability to delegating log data to a remote resource provider is a key feature in cloud based data storage. The major problem in cloud environment is the fact that the security and confidentiality of a remote resource cannot be technically validated by the end user. In our case, we are delegating log data which contains record of most system events including user activities, which will be an important target for malicious attackers. This becomes a high risk factor to maintain the log data over a cloud environment.

In this paper, we propose an effective secured cloud base log management system with required protocols and methods that strengthen the security of query execution over a logging cloud database. We used homomorphic encryption scheme which enables a customer to generate a query that can be executed by a third party, without revealing the underlying algorithm or the processed log data, which helps securing log data in a cloud based log management system.

5. Conclusion

Current implementation of log management system with a a method to perform the execution of encrypted query operating on encrypted data. We discussed a simple homomorphic encryption scheme as a reference model. The basic homomorphic scheme will not achieve a effective processing of queries over encrypted log data and also have drawbacks such as to reduce the noise in cipher text space , which can be compensate by implement advanced schemes, such as [10] and [11], base on entry's approach of bootstrapping a fully homomorphic from a somewhat homomorphic system and provide addition and multiplication plus a normalization procedure that is supposed to allow unlimited chaining of operations in cipher-text space. This technique of reducing noise in the cipher-text space requires. So the future work for this paper will be implementation of advanced homomorphic encryption scheme which able to reduce the noise in cipher-text space and computation overhead.

Reference

[1] K. Kent and M. Souppaya. (1992). *Guide to Computer Security Log Management, NIST Special Publication 800-92*[Online].Available: http://csrc.nist.gov/publications/nistpubs/800-92/SP800-92.pdf

[2] D. New and M. Rose, *Reliable Delivery for Syslog*, Request for Comment RFC 3195, Internet Engineering Task Force, Network Working Group, Nov. 2001.

[3] M. Bellare and B. S. Yee, "Forward integrity for secure audit logs," Dept. Comput. Sci., Univ. California, San Diego, Tech. Rep., Nov. 1997.

[4] BalaBit IT Security (2011, Sep.). *Syslog-ng—Multiplatform Syslog Server and Logging Daemon* [Online]. Available: http://www.balabit.com/network-security/syslog-ng

[5] J. Kelsey, J. Callas, and A. Clemm, *Signed Syslog Messages*, Request for Comment RFC 5848, Internet Engineering Task Force, Network Working Group, May 2010.

[6] Aderemi A. Atayero*, Oluwaseyi Feyisetan "Security Issues in Cloud Computing: *The Potentials of Homomorphic Encryption"* vol. 2, no. 10, October 2011

[7] I. Teranishi, J. Furukawa, and K. Sako, "*k*-times anonymous authentication (extended abstract)," in *Proc. 10th Int. Conf. Theor. Appl. Cryptology Inform. Security*, LNCS 3329. 2004, pp. 308–322.

[8] Craig Gentry, *Fully Homomorphic Encryption Using Ideal Lattices*, STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing, DOI:10.1145/1536414.1536440

[9] Michael Brenner, Jan Wiebelitz, Gabriele von Voigt and Matthew Smith Research Center L3S, Hannover, Germany"*Secret Program Execution in the Cloud Applying Homomorphic Encryption"* ieee ,31 May -3 June 2011. [10] Marten van Dijk, Craig Gentry, Shai Halevi and Vinod Vaikuntanathan, *Fully Homomorphic Encryption over the Integers, Advances in Cryptology* EUROCRYPT 2010, Springer DOI:10.1007/978-3-642-13190-5

[11] Nigel P. Smart and Frederik Vercauteren, *Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes*, Public Key Cryptography PKC 2010, Springer 10.1007/978-3-642-13013-7