

SECURE LOGGING FRAMEWORK USING CLOUD

¹MANIKANTA K B, ²RANJITHA V, ³PRIYANKA N

¹Assistant Professor, ²Assistant Professor, ³Assistant Professor

¹Department of CSE,

¹GITAM SCHOOL OF TECHNOLOGY, BENGALURU, Karnataka, India

Abstract : Log files storage in an organization is important because to perform correct function of organization. Log files consists sensitive information and confidentiality records. Hence to provide security to log files is necessary. Store log files to cloud for long term save the cost. And provide security. So, in this paper we propose the cloud based framework to doing the same.

IndexTerms – MAC, Privacy, Cloud

I. INTRODUCTION

CLOUD 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 Amazon, 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 becoming a significant barrier to the wide adoption of cloud services.

Privacy is one of the most concerned issues in cloud computing. Electronic health records and financial transaction records usually contains sensitive information but these data can offer significant human benefits if they are analyzed and mined by various research centers. For instance, Microsoft HealthVault, an online cloud health service, aggregates data from users and shares data with various research organizations. The data can be easily exposed by using traditional privacy protection on cloud. This can bring considerable economic loss or severe social reputation impairment to data owners. Hence, data privacy issues need to be addressed urgently before data sets are analyzed or shared on cloud.

A LOG is a record of events occurring within an organization's system or network [1]. Logging is important because log data can be used to troubleshoot problems, fine-tune system performance, identify policy violations, investigate malicious activities, and even record user activities. Log records play a significant role in digital forensic analysis of systems. Regulations such as HIPAA [2], Payment Card Industry Data Security Standard [3], or Sarbanes-Oxley [4] often require forensically sound preservation of information. To comply with these regulations, evidence produced in a court of law, including log records, must be unbiased, not tampered with, and complete before they can be used. Since log files contain record of most system events including user activities, they become an important target for malicious attackers. An attacker, breaking into a system, typically would try not to leave traces of his or her activities behind. Consequently, the first thing an attacker often does is to damage log files or interrupt the logging services. Furthermore, the sensitive information contained in log files often directly contributes to confidentiality breaches. An example of this is when logs contain database transaction data. Frequently, log information can be helpful to an attacker in gaining unauthorized access to system. One example of this is the case when a user mistakenly enters her password in the username field while logging into a system. Logging programs will store the password as the user-id to record the information that a user has failed to log in. Last, but not least, information in log file can also be used to cause privacy breaches for users in the system since the log file contains record of all events in the system. In light of the above observations, it is very important that logging be provided in a secure manner and that the log records are adequately protected for a predetermined amount of time (maybe even indefinitely). Traditional logging protocols that are based on syslog [5] have not been designed with such security features in mind. Security extensions that have been proposed, such as reliable delivery of syslog [6], forward integrity for audit logs [7], syslog-ng [8], and syslog-sign [9], often provide either partial protection, or do not protect the log records from end point attacks. In addition, log management requires substantial storage and processing capabilities. The log service must be able to store data in an organized manner and provide a fast and useful retrieval facility. Last, but not least, log records may often need to be made available to outside auditors who are not related to the organization. Deploying a secure logging infrastructure to meet all these challenges entails significant infrastructural support and capital expenses that many organizations may find overwhelming. The emerging paradigm of cloud computing promises a low cost opportunity for organizations to store and manage log records in a proper manner. Organizations can outsource the long-term storage requirements of log files to the cloud. The challenges of storing and maintaining the log records become a concern of the cloud provider. Since the cloud provider is providing a single service to many organizations that it will benefit from economies of scale. Pushing log records to the cloud, however, introduces a new challenge in storing and maintaining log records. The cloud provider can be honest but curious. This means that it can try not only to get confidential information directly from log records, but also link log record related activities to their sources. No existing protocol addresses all the challenges that arise when log storage and maintenance is pushed to the cloud.

In this paper, we propose a comprehensive solution for storing and maintaining log records in a server operating in a cloud-based environment. We address security and integrity issues not only just during the log generation phase, but also during other stages in the log management process, including log collection, transmission, storage, and retrieval. The major contributions of this paper are as follows. We propose an architecture for the various components of the system and develop cryptographic protocols to address integrity and confidentiality issues with storing, maintaining, and querying log records at the honest but curious cloud provider and in transit. We also develop protocols so that log records can be transmitted and retrieved in an anonymous manner over an existing anonymizing infrastructure such as Tor [16]. This successfully prevents the cloud provider or any other observer from correlating requests for log data with the requester or generator. Finally, we develop a proof-of-concept prototype to demonstrate the feasibility of our approach and discuss some early experiences with it. To the best of our knowledge, ours is the first work to provide a complete solution to the cloud based secure log management problem. The remainder of this paper is organized as follows. Section II identifies the security properties that must be ensured in a cloud-based log management system. Section III discusses some of the existing secure logging protocols in light of these properties. In Section IV, we present the architecture for our cloud-based secure logging facility and describe the services provided by each component. We conclude this paper with a discussion of our future work in Section V.

II. PROPERTIES OF SECURE LOGGING

- 1) **Correctness:** Log data is useful only if it reflects true history of the system at the time of log generation. The stored log data should be correct, that is, it should be exactly the same as the one that was generated.
- 2) **Tamper Resistance:** A secure log must be tamper resistant in such a way that no one other than the creator of the log can introduce valid entries. In addition, once those entries are created they cannot be manipulated without detection. No one can prevent an attacker who has compromised the logging system from altering what that system will put in future log entries. One cannot also prevent an attacker from deleting any log entries that have not already been pushed to another system. The goal of a secure audit log in this case is to make sure that the attacker cannot alter existing log entries (i.e., the precompromise log entries) and that any attempts to delete or alter existing entries (truncation attack [10]) will be detected.
- 3) **Verifiability:** It must be possible to check that all entries in the log are present and have not been altered. Each entry must contain enough information to verify its authenticity independent of others. If some entries are altered or deleted, the ability to individually verify the remaining entries (or blocks of entries) makes it possible to recover some useful information from the damaged log. Moreover, the individual entries must be linked together in a way that makes it possible to determine whether any entries are missing (forward integrity, [7]).
- 4) **Confidentiality:** Log records should not be casually browse able or 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.
- 5) **Privacy:** Log records should not be casually traceable or linkable to their sources during transit and in storage.

III. RELATED WORK

A number of approaches have been proposed for logging information in computing systems. Most of these approaches are based on syslog which is the de facto standard for network wide logging protocol (see RFC 3164 [5]). The syslog protocol uses UDP to transfer log information to the log server. Thus, there is no reliable delivery of log messages. Moreover, syslog does not protect log records during transit or at the end-points. Syslog-ng [8] is a replacement that is backward compatible with syslog. Some of its features include support for IPv6, capability to transfer log messages reliably using TCP, and filtering the content of logs using regular expressions. Syslog-ng prescribes log record encryption using SSL during transmission so as to protect the data from confidentiality and integrity breaches while in transit. However, syslog-ng does not protect against log data modifications when it resides at an end-point. Syslog-sign [9] is another enhancement to syslog that adds origin authentication, message integrity, replay resistance, message sequencing, and detection of missing messages by using two additional messages—"signature blocks" and "certificate blocks." Unfortunately, if signature blocks associated with log records get deleted after authentication, tamper evidence and forward integrity is only partially fulfilled. Syslog-sign also does not provide confidentiality or privacy during the transmission of data or at the end points. Syslog-pseudo [11] proposes a logging architecture to pseudonymize log files. The main idea is that log records are first processed by a pseudonymizer before being archived. The pseudonymizer filters out identifying features from specific fields in the log record and substitutes them with carefully crafted pseudonyms. Thus, strictly speaking, this

protocol does not ensure correctness of logs. That is, the log records that are stored are not the same as the ones that are generated. The other problem with this paper is that while the protocol anonymizes each log record individually it does not protect. Log records from attacks that try to correlate a number of anonymized records. Our objective, on the other hand, is precisely this. Moreover, privacy breaches that can occur from scenarios such as the user erroneously typing the user id in a password field (as discussed earlier) or identifying information available in fields that are not anonymized, are also not addressed in this paper. The Anonymous log file anonymizer [12] performs a similar anonymization of identifying information by substitution with default values or more coarse values. However, the problem with this paper is that if the original values are needed in investigation, they cannot be restored. Neither syslog-pseudo nor anonymous log file anonymizer protects log records from confidentiality and integrity violations and other end-point attacks.

Reliable-syslog [6] aims to implement reliable delivery of syslog messages. It is built on top of the blocks extensible exchange protocol (BEEP [13]) which runs over TCP to provide the required reliable delivery service. The Reliable syslog protocol allows device authentication and incorporates mechanisms to protect the integrity of log messages and protect against replay attacks of log data; however it does not prevent against confidentiality or privacy breaches at the end-points or during transit. The notion of forward-integrity of log records was proposed by Bellare and Yee [7] to protect precompromise log data from postcompromise insertion, deletion, modification, and reordering. Forward integrity is established by a secret key that becomes the starting point of a hash-chain. The hash-chain is generated by a cryptographically strong one-way function in which the key is changed for every log record. Schneier and Kelsey [14] also proposed a logging scheme that ensures forward integrity. This scheme is based on forward-secure message authentication codes and one-way hash chains similar to that suggested by the Bellare-Yee protocol. However, the major problem of both schemes is that both require online trusted servers to maintain the secret key and verification of log records. If the trusted server is attacked or compromised, it breaks the security of the logging scheme. Holt [15] improves upon the Schneier-Kelsey protocol by incorporating public verifiability of log records. However, this scheme being a public-key based scheme, the overhead is significantly more. None of these three schemes consider the privacy concerns of storing and retrieving log records. In addition, all these three scheme suffer from truncation attacks where an attacker deletes a contiguous subset of log records from the very end. This attack can be launched on log records that have yet to be pushed to the trusted server where they are finally stored. Ma and Tsudik address this problem in [10]. They use the notion of forward-secure sequential aggregate authentication in which individual signatures are folded into one single aggregated signature and all other signatures are deleted. An attacker cannot recreate this signature without knowing all previous signatures. The concern with Ma and Tsudik's scheme is that it is very expensive to verify only a single log record (or block, depending on what is considered a unit of log entries over which the aggregated message authentication code is generated). The whole chain of log records over which the accumulated signature has been generated, needs to be verified to achieve this. Further, Ma and Tsudik's scheme does not address the confidentiality and privacy problems with log file storage and retrieval.

IV. SYSTEM ARCHITECTURE

A. Components

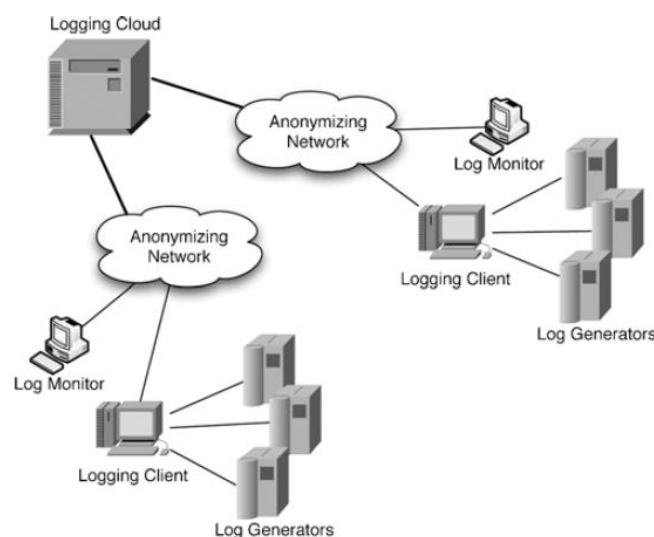


Fig 1. System architecture for cloud-based secure logging

The overall architecture of the cloud based secure log management system is shown in Fig. 1. There are four major functional components in this system.

- 1)Log Generators: These are the computing devices that generate log data. Each organization that adopts the cloud-based log management service has a number of log generators. Each of these generators is equipped with logging capability. The log files generated by these hosts are not stored locally except temporarily till such time as they are pushed to the logging client.
- 2)Logging Client or Logging Relay: The logging client is a collector that receives groups of log records generated by one or more log generators, and prepares the log data so that it can be pushed to the cloud for long term storage. The log data is transferred from the generators to the client in batches, either on a schedule, or as and when needed depending on the amount of log data

waiting to be transferred. The logging client incorporates security protection on batches of accumulated log data and pushes each batch to the logging cloud. When the logging client pushes log data to the cloud it acts as a logging relay. We use the terms logging client and logging relay interchangeably. The logging client or relay can be implemented as a group of collaborating hosts. For simplicity however, we assume that there is a single logging client.

3) Logging Cloud: The logging cloud provides long term storage and maintenance service to log data received from different logging clients belonging to different organizations. The logging cloud is maintained by a cloud service provider. Only those organizations that have subscribed to the logging cloud's services can upload data to the cloud. The cloud, on request from an organization can also delete log data and perform log rotation. Before the logging cloud will delete or rotate log data it needs a proof from the requester that the latter is authorized to make such a request. The logging client generates such a proof. However, the proof can be given by the logging client to any entity that it wants to authorize.

4) Log Monitor: These are hosts that are used to monitor and review log data. They can generate queries to retrieve log data from the cloud. Based on the log data retrieved, these monitors will perform further analysis as needed. They can also ask the log cloud to delete log data permanently, or rotate logs.

We assume that the organization maintains the log generators and the logging client. The log monitor can be maintained by the same organization or can be a separate entity. The logging client can also play the role of a log monitor. We develop our model assuming that the log monitor is a separate entity that is trusted by the logging client. Since the logging client and log monitor operate independent of each other, they can communicate only in an asynchronous manner. This means that if a logging client wants to send some data to the log monitor (or vice versa), the sender cannot expect the receiver to be online to receive the data. As a result the sender has to publish the data in some location and the receiver needs to retrieve the data from there when needed. The logging cloud facilitates this communication by receiving and servicing appropriate requests.

The logging client and the log monitor communicates with the external world, over an unencrypted network that provides anonymous full-duplex communication. Our proof-of-concept prototype gets this service from the Tor network [16],[17]. Attacks on the Tor network that breach anonymity of communicating parties, have been well studied and solutions proposed. Protecting the anonymous communication channel is beyond the scope of this paper.

B. Log File Preparation

Central to our log file preparation protocol is the creation and management of three sets of keys— A_i and X_i which are keys for ensuring log record integrity, and K_i which is a key for ensuring log record 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. We do not rely on a single trusted entity to store and manage keys. Instead, we propose using a secret sharing scheme (such as the ones by Shamir [19] or Blakley [20]) to distribute the keys A_0 , X_0 , and K_0 across several hosts.

The idea is that given a secret S , and n and q two nonnegative integers such that $0 < q \leq n$, we would like n entities to share the secret S such that: 1) no single entity holds the complete secret; 2) any subgroup of entities of size $\geq q$ can collectively recreate or recover the secret S ; and 3) no subgroup of entities of size $t < q$ can re-create or recover the secret. One of the problems of the secret sharing schemes proposed by Shamir or Blakley is that an attacker can successively compromise each host till it has compromised q hosts. In such a case, the attacker obtains the secret keys. To protect against such a possibility we use a proactive secret-sharing scheme [21], [22]. The idea behind these schemes is that at the end of a fixed period of time, the shares stored at each host change.

C. Secrecy for Log Records

Symbol Used	Interpretation
$E_p[M]$	Encryption of message M with some secret key p
$M_1 M_2$	Concatenation of messages M_1 and M_2
$H_k^i[M]$	Cryptographic hash of message M with key k and a hard to invert hash function H with the hash performed i number of times
$H^n[M]$	Message M hashed n number of times
TS	A global timestamp

Table 1

Table I summarizes the symbols used in the discussion on log preparation protocol.

The protocol starts with the logging client randomly generating three master keys— A_0 and X_0 for ensuring log integrity, and K_0 for ensuring log confidentiality. These keys need to satisfy the requirements of the chosen proactive secret-sharing scheme. It then embarks upon preparing the log records. Log data arrives at the logging client as a series of messages L_1, L_2, \dots, L_n . Each L_i contains a group of log records generated by a log generator. We assume that these messages are transmitted to the logging client over an authenticated network. The logging client uploads prepared log records in batches of n . The value n is determined randomly at the beginning of each log batch preparation.

1) Before any log data arrives at the logging client, the

logging client creates a special first log entry $L_0 = \text{TS, log-Initialization, } n$. It then encrypts this log entry with the key K_0 and computes the message authentication code $MAC_0 = H_{A_0} [E_{K_0} [L_0]]$ for the encrypted entry with the key A_0 . The client adds the resulting first log entry for the current batch— $E_{K_0} [L_0], MAC_0$ —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 $K_1 = H[K_0]$, 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 $M_1 = L_1 \parallel H_{A_0} [E_{K_0} [L_0]]$. It encrypts M_1 with the key K_1 , and creates a message authentication code for the resulting data as $MAC_1 = H_{A_1} [E_{K_1} [M_1]]$. It also computes an aggregated message authentication code

$MAC_{-1} = H_{nX_1} [MAC_0 \parallel MAC_1 \parallel n]$. The log batch entry is

$E_{K_1} [M_1], MAC_{-1}$. It then creates the next set of keys $A_2 = H[A_1]$, $X_2 = H[X_1]$ and $K_2 = H[K_1]$ 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 $E_{K_i} [M_i], MAC_{-i}$, where $M_i = L_i \parallel MAC_{-i-1}$ and $MAC_i = H_{A_i} [E_{K_i} [M_i]]$. It also creates the aggregated message authentication code $MAC_{-i} = H_{n-i+1X_i} [MAC_{-i-1} \parallel MAC_i \parallel n - i + 1]$. 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 $LC = E_{K_{n+1}} [\text{TS, log-close} \parallel MAC_n], H_{A_{n+1}} [E_{K_{n+1}} [\text{TS, log-close} \parallel MAC_n]]$, 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.

D. Anonymous Upload, Retrieval and Deletion of Log Data

The logging client uploads data in batches where each batch is delimited by a start-of-log record and an end-of-log record. The cloud provider will accept log records only from its authorized clients. Thus, during upload a logging client has to authenticate to the logging cloud to prove that the client had obtained prior authorization from the logging cloud to use the latter's services. However, we do not want the identity of the logging client to be linked to any of its transactions including the authentication process. To balance privacy and accountability, we propose using the k-times anonymous authorization protocol [23] for authentication by the logging client to the logging cloud.

We develop four different protocols for anonymous upload, retrieval and deletion of log data. We assume the existence of an anonymizing network such as Tor to support the anonymous message exchanges in these protocols. We also assume that the logging client knows the public key of the entity that has valid reason to access the log files created by the logging client (the log monitor in our case). This entity also knows the public key of the client. These public keys may be known to the logging cloud or other adversaries.

1) Anonymous Upload-Tag Generation: To later retrieve it, an uploaded log batch of log records needs to be indexed by a unique key value. However, we need to ensure that this key value cannot be traced back to the logging client that uploaded the data nor the log monitor that seeks the data. For this purpose, the log data is stored at the cloud indexed by an anonymously generated upload-tag. This upload-tag is created by the logging client in cooperation with the log monitor. It has the property that it is created by publicly available information. However, a given upload-tag cannot be linked either to the corresponding logging client or a log monitor. To retrieve log data from the cloud, the log monitor sends a retrieve request to the logging cloud using an upload tag. The upload-tag is not sent in an encrypted manner. Thus, any adversary can use the upload-tag to retrieve the corresponding log data. However, the log data can be deciphered if and only if the corresponding decryption key is available.

2) Anonymous Upload: The logging client authenticates itself to the logging cloud in an anonymous manner. Once successfully authenticated, the logging client sends a formatted message containing 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 (if the logging client, or another entity authorized the logging client needs to do so). Since all messages to the logging cloud are over anonymous channels, any entity can potentially ask the logging cloud to delete a log data. To prevent this from happening, the logging cloud challenges the requester of such a delete operation to prove that it has the necessary authorization. Of course, this can be achieved trivially by the logging client since it can be anonymously authenticated by the logging cloud. However, we allow the deletion to be performed by any entity authorized by the logging client. For this reason, we use the notion of a delete-tag.

3) Anonymous Retrieve: This protocol is straightforward. The entity that needs to download log data (most of the time the log monitor), sends a retrieve request (anonymously) together with the upload-tag corresponding to the desired log data. The logging cloud gets the data from its storage and sends it over the anonymous channel to the requester. The cloud provider does not have to authenticate the requester. This is because, by virtue of the log batches being encrypted, the retrieved data is useful only to those who have the valid decryption keys.

4) Anonymous Delete: To delete log data, the delete requester

sends an appropriate delete message to the logging cloud. In response, the logging cloud throws a challenge to the requester. The requester proves authorization to delete by presenting a correct delete tag.

E. Analysis:

The major objective of these protocols is to protect the anonymity of the logging client by preventing the linking of some log record stored on the cloud with the logging client that generated it. (The real objective is to maintain the anonymity of the organization whose log records are being stored on the cloud. We assume that this is equivalent to protecting the anonymity of the logging client.) To get assurance of the anonymity we must ensure that:

- 1) no single party has enough information to link log records uploaded by the logging client to it, and
- 2) it is not possible for all parties to collude and get this information. The adversaries in our protocol are then the honest but curious cloud provider and any attacker that can attack the network and the cloud provider. We assume that all encryption is resistant to cryptanalysis and that they do not leak information.
- 3) The logging cloud by itself does not have sufficient information to breach anonymity of a logging client. First of all since all messages to the logging cloud from the logging client or log monitor are over the Tor network, these messages cannot directly be tied to their sources.

The logging cloud can hold of the following messages-- identity of the logging cloud and log monitor when the partial halves of the anchor upload-tag were being generated and the public parameters used to generate these halves. It cannot, however, tie a single half of the anchor to either the logging client or log monitor that generated it. It can also get hold of an upload-tag.

However, the upload-tag cannot be tied to the anchor that generated it because of the hash function used. Moreover, by virtue of the upload-tag being derived from a Diffie-Hellman key, an upload-tag cannot be tied to the parameters that were used to generate it.

- 4) Any adversary that eavesdrop up on the Tor network cannot link any observed message to either a logging client or a log monitor. By attacking the logging cloud it can at most get the same information as that the logging cloud has. Thus, by previous analysis the attacker cannot reach privacy of the logging client or the log monitor.

- 5) Similar reasoning indicates that by collusion among an attacker and the logging cloud privacy cannot be breached.

Assuming that the anonymous authentication protocol used by the logging cloud is sufficiently robust, an attacker cannot successfully masquerade as a valid logging client. If an attacker replays messages the logging cloud can detect that because every log upload is associated with a unique time stamp. If the cloud colludes with the attacker and does not flag such messages, there will just be duplicate log data in the cloud. In short, by combination of the log preparation protocol and log upload, retrieval and delete protocols, all the desirable security properties for a cloud based log management protocol are satisfied,

V. CONCLUSIONS AND FUTUREWORKS:

Logging plays a very important role in the proper operation of an organization's information processing system. However, maintaining logs securely over long periods of time is difficult and expensive in terms of the resources needed. The emerging paradigm of cloud computing promises a more economical alternative. In this paper, we proposed a complete system to securely outsource log records to a cloud provider. We reviewed existing solutions and identified problems in the current operating system based logging services such as syslog and practical difficulties in some of the existing secure logging techniques. We then proposed a comprehensive scheme that addresses security and integrity issues not just during the log generation phase, but also during other stages in the log management process, including log collection, transmission, storage and retrieval. One of the unique challenges is the problem of log privacy that arises when we outsourced log management to the cloud. Log information in this case should not be casually linkable or traceable to their sources during storage, retrieval and deletion.

We provided anonymous upload, retrieve and delete protocols on log records in the cloud using the Tor network. The protocols that we developed for this purpose have potential for usage in many different areas including anonymous publish-subscribe. Current implementation of the logging client is loosely coupled with the operating system based logging. In the future, we plan to refine the log client implementation so that it is tightly integrated with the OS to replace current log process. In addition, to address privacy concerns current implementation allows access to log records that are indirectly identified by upload-tag values. We plan to investigate practical homomorphic encryption schemes that will allow encryption of log records in such a way that the logging cloud can execute some queries on the encrypted logs without breaching confidentiality or privacy. This will greatly reduce the communication overhead between a log monitor and the logging cloud needed to answer queries on logs. There are some alternative approaches to Tor for providing non linkability and non traceability of network communications, such as JAP [27], Ultrasurf [28], and FreeGate [29]. Among these, Tor is a more mature implementation and one of the most popular. More importantly, Tor can be easily integrated with any TCP based protocol that can be SOCKS-ified. This was one of our requirements and why we evaluated the performance over Tor. It remains to be seen whether the other anonymizing protocols can be so adapted and, if so, how they perform compared to Tor. This is part of our future work.

REFERENCES

- [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] U.S. Department of Health and Human Services. (2011, Sep.). HIPAA—General Information [Online]. Available: <https://www.cms.gov/Hipaageninfo>
- [3] PCI Security Standards Council. (2006, Sep.) Payment Card Industry (PCI) Data Security Standard—Security Audit Procedures Version 1.1 [Online]. Available: <https://www.pcisecuritystandards.org/pdfs/pci-audit-procedures-v1-1.pdf>
- [4] Sarbanes-Oxley Act 2002. (2002, Sep.). A Guide to the Sarbanes-Oxley Act [Online]. Available: <http://www.soxlaw.com/>
- [5] C. Lonvick, The BSD Syslog Protocol, Request for Comment RFC 3164, Internet Engineering Task Force, Network Working Group, Aug. 2001.
- [6] D. New and M. Rose, Reliable Delivery for Syslog, Request for Comment RFC 3195, Internet Engineering Task Force, Network Working Group, Nov. 2001.
- [7] M. Bellare and B. S. Yee, "Forward integrity for secure audit logs," Dept. Comput. Sci., Univ. California, San Diego, Tech. Rep., Nov. 1997.

- [8] BalaBit IT Security (2011, Sep.). Syslog-ng—Multiplatform Syslog Server and Logging Daemon [Online]. Available: <http://www.balabit.com/network-security/syslog-ng>
- [9] J. Kelsey, J. Callas, and A. Clemm, Signed Syslog Messages, Request for Comment RFC 5848, Internet Engineering Task Force, Network Working Group, May 2010.
- [10] D. Ma and G. Tsudik, “A new approach to secure logging,” ACM Trans. Storage, vol. 5, no. 1, pp. 2:1–2:21, Mar. 2009.
- [11] U. Flegel, “Pseudonymizing unix log file,” in Proc. Int. Conf. Infrastructure Security, LNCS 2437, Oct. 2002, pp. 162–179.
- [12] C. Eckert and A. Pircher, “Internet anonymity: Problems and solutions,” in Proc. 16th IFIP TC-11 Int. Conf. Inform. Security, 2001, pp. 35–50.
- [13] M. Rose, The Blocks Extensible Exchange Protocol Core, Request for Comment RFC 3080, Internet Engineering Task Force, Network Working Group, Mar. 2001.
- [14] B. Schneier and J. Kelsey, “Security audit logs to support computer forensics,” ACM Trans. Inform. Syst. Security, vol. 2, no. 2, pp. 159–176, May 1999.
- [15] J. E. Holt, “Logcrypt: Forward security and public verification for secure audit logs,” in Proc. 4th Australasian Inform. Security Workshop, 2006, pp. 203–211.
- [16] R. Dingledine, N. Mathewson, and P. Syverson, “Tor: The secondgeneration onion router,” in Proc. 12th Ann. USENIX Security Symp., Aug. 2004, pp. 21–21.
- [17] The Tor Project, Inc. (2011, Sep.) Tor: Anonymity Online [Online]. Available: <http://www.torproject.org>
- [18] D. Dolev and A. Yao, “On the security of public key protocols,” IEEE Trans. Inform. Theory, vol. 29, no. 2, pp. 198–208, Mar. 1983.
- [19] A. Shamir, “How to share a secret,” Commun. ACM, vol. 22, no. 11, pp. 612–613, Nov. 1979.
- [20] G. R. Blakley, “Safeguarding cryptographic keys,” in Proc. Nat. Comput. Conf., Jun. 1979, p. 313.
- [21] R. Ostrovsky and M. Yung, “How to withstand mobile virus attack,” in Proc. 10th Ann. ACM Symp. Principles Distributed Comput., Aug. 1991, pp. 51–59.
- [22] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, “Proactive secret sharing or: How to cope with perpetual leakage,” in Proc. 15th Ann. Int. Cryptology Conf., Aug. 1995, pp. 339–352.
- [23] 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.
- [24] D. L. Wells, J. A. Blakeley, and C. W. Thompson, “Architecture of an open object-oriented database management system,” IEEE Comput., vol. 25, no. 10, pp. 74–82, Oct. 1992.
- [25] K. Nørveg, O. Sandstaa, and K. Bratbergsengen, “Concurrency control in distributed object oriented database systems,” in Proc. 1st East-Eur Symp. Adv. Databases Inform. Syst., Sep. 1997, pp. 32–32.
- [26] R. Droms, Dynamic Host Configuration Protocol, Request for Comment RFC 2131, Internet Engineering Task Force, Network Working Group, Mar. 1991.
- [27] Project: AN.ON—Anonymity Online. (2012, Mar.). JAP Anonymity and Privacy [Online]. Available: <http://anon.inf.tu-dresden.de/index-en.html>
- [28] Ultrareach Internet Corporation. (2012, Mar.). Ultrasurf—Privacy Security, Freedom [Online]. Available: <http://ultrasurf.us/index.html>
- [29] Global Internet Freedom Consortium. (2012, Mar.). FreeGate [Online].

