ISSN 2278-3091

International Journal of Advanced Trends in Computer Science and Engineering, Vol.2 , No.6, Pages : 96-101 (2013)
Special Issue of ICETEM 2013 - Held on 29-30 November, 2013 in Sree Visvesvaraya Institute of Technology and Science, Mahabubnagar – 204, AP, India

Enhancing Seekable Sequential Key Generators along with FSS Scheme for Secure Logging in the Cloud

V.Vijaya Chandra Rao[1], **Dr.A.Suresh Babu** [2]
[1]JNTUACEP, India, vijaychandrarao.v@gmail.com
[2]JNTUACEP, India,asureshjntu@gmail.com

ABSTRACT

The need for secure logging is well-understood by the security professionals, together with each researchers and practitioners. The flexibility to the accuracy verifies all (or some) log entries is very important to any application using secure logging techniques. During this paper, we start by examining progressive in secure logging and determine some issues inherent to systems supported trusty third-party servers in the cloud. We tend to then propose a distinct approach to secure logging primarily based upon recently developed Forward-Secure consecutive aggregate (FssAgg) authentication techniques. Our approach offers each space-efficiency and obvious security. we tend to illustrate two concrete schemes - one private-verifiable and one public-verifiable - that provide sensible secure logging with none reliance on on-line trustworthy third parties or secure hardware. We tend to additionally investigate the thought of immutability within the context of forward secure sequential aggregate authentication to produce finer grained verification. Finally, we tend to report on some expertise with a prototype built upon a preferred code version control system.

Keywords: Secure logging, cloud server, MACs, Signatures, forward secure stream integrity, BBox.

INTRODUCTION

A LOG could be a record of events occurring among an organization's system or network [1]. Logging is vital as a result of log information may be utilized to troubleshoot issues, fine tune system performance, recognize policy violations, investigate malicious actions, and even manages user actions. Log records play a major role in digital forensic analysis of systems. Laws like HIPAA [2], Payment Card trade information Security standard [3], or Sarbanes-Oxley [4] typically need forensically sound preservation of data. To go with these laws, proof produced during a court of law, together with log records, should be unbiased, non-tampered with, and complete before they'll be used.

System and application logs are of nice value for administrators, e. g., for observance, fault management, and forensics. The predominant format for logs is simple text that is the focus here; for instance, the syslog daemon on UNIX system and Linux systems, also as applications just like the Apache web server, store log entries in line-oriented plain text. As an alternative, proprietary binary or XML-based file formats also as relational databases is also used; as an example, this can be employed by the Microsoft Windows event log.

Independent of the storage format, secure logs should fulfill the subsequent basic criteria in practice:

- The log's integrity should be ensured: Neither a malicious administrator nor an attacker, who has compromised a system, could be able to delete or modify existing, or insert fake log entries.
- The log must not violate compliance criteria [5]. For example, European data protection laws regulate the retention of personal data, which includes, among many others, user names and IP addresses. These restrictions also apply to log entries according to several German courts' verdicts that motivated our work.
- The confidentiality of log entries should be safeguarded; i. e., read access to log entries should be confined to an impulsive set of users.

The provision of log entries should be created certain of in a typical information center or network operations setting, the integrity criterion is typically satisfied by employing a trustworthy, central log server: Log entries aren't (solely) stored locally on a device, however sent over the network to a different machine; thus, an attacker would have to be compelled to compromise each these device and also the log server before having the ability to control the log while not being detectable. The compliance criterions are often satisfied by deleting previous log files, e. g., once and the common period of seven days. As of these days, confidentiality is usually handled in a per-file manner; for instance, UNIX file system permissions are usually utilized build a log file readable for a particular user or cluster. Prospective trendy logging facilities additionally enable confidentiality in a per-entry granularity, however aren't nevertheless in such wide use. Finally, the provision demand for log files is that the same as for alternative necessary information and usually ensured by information redundancy, i. e., copies and backups.

Our work is intended by the large-scale distributed setting of the SASER-SIEGFRIED project (Safe and Secure European Routing) [6], during which over 50 project partners design and implement network architectures and technologies for secure future networks. The project's goal is to remedy security vulnerabilities of today's ip layer networks within the 2020 timeframe. Thereby, security mechanisms for future networks are going to be designed supported an analysis of the presently predominant security issues within the ip layer, additionally as future problems like vendor backdoors and traffic anomaly detection. The project focuses on inter-domain routing, and routing decisions are based on security metrics that are a part of log entries sent by active network elements to central network management systems; thus, the integrity of this information should be protected, providing a use case that's almost like traditional intra-organizational log file management applications.

In lightweight of the on top of observations, it's vital that logging be provided in a very secure manner which the log records are adequately protected for a predetermined quantity of your time (maybe even indefinitely). Traditional work protocols that are based on syslog [7] haven't been designed with such safety measures in mind. Security extensions that are proposed, like reliable delivery of syslog [8], forward integrity for audit logs [9], syslog-ng [10], and syslog-sign [11], usually offer either partial protection, or don't protect the log records from final point attacks. Additionally, log management needs substantial storage and processing capabilities. The log service should be ready to store information in an organized manner and supply a quick and helpful retrieval facility. Last, however not least, log records typically ought to be made accessible to outside auditors who don't seem to be associated with the organization. Deploying a secure logging infrastructure to meet of these challenges entails important infrastructural support and capital expenses that several organizations might notice enormous.

The rising paradigm of cloud computing guarantees a low price chance for organizations to store and manage log records in an exceedingly correct manner. Organizations will outsource the long-term storage needs of log files to the cloud. The conflicts of storing and managing the log records become a priority of the cloud provider. Since the cloud provider is providing one service to several organizations that it'll have the benefit of economies of scale. Pushing log records to the cloud, but introduces a new challenge in storing and maintaining log records. The cloud provider may be honest however curious. This implies that it will attempt not only to urge confidential information directly from log records, however additionally link log record related activities to their sources. No any other defined protocol directs all the challenges that arise once log storage and maintenance is pushed to the cloud.

We have a tendency to propose new secure logging schemes based on recently proposed FssAgg authentication techniques. Our schemes offer forward-secure stream integrity for audit logs generated and keep on untrusted work machines and avoid the undesirable options of previous schemes. Our schemes inherit the obvious security of the underlying FssAgg schemes. We tend to measure proposed schemes by scrutiny them with previous schemes, in terms of security moreover as communication and computation efficiency. Our evaluation shows that our schemes provide higher security and incur less computation and communication overhead. We tend to propose an algorithm to get outputs correct, complete and compact log views. Accuracy, completeness and compactness of log views follow from tamper proof upon the idea that the attacker model is analogous to it of log information at rest, that the non-public key of the BBox [14] isn't acknowledged by the attacker which the devices communicate each event occurring within the system.

SYSTEM OVERVIEW

The overall design of the cloud primarily based secure log management system is shown in Fig. 1. There are four major useful elements during this system.

Log Generators

These are the computing devices that generate log information. Every organization that adopts the cloud-based log management service includes a variety of log generators. Every of those generators are provided with logging capability. The log files generated by these hosts aren't keeping locally except briefly until such time as they're pushed to the logging client.

Logging client or logging Relay:

The logging client could be a collector that receives clusters of log records generated by one or a lot of log generators, and prepares the log information in order that it may be pushed to the cloud for future storage. The log information is transferred from the generators to the client in batches, either on a schedule, or as and once required reckoning on the quantity of log information waiting to be transferred. The logging client consolidates security protection on batches of accumulated log information and pushes every batch to the logging cloud. Once the logging client pushes log information to the cloud it acts as a logging relay. We have a tendency to use the terms logging client and logging relay interchangeably. The logging client or relay may be enforced as a bunch of collaborating hosts. For simplicity but, we have a tendency to assume that there's one logging client.

Logging Cloud

The logging cloud provides future storage and maintenance service to log information received from completely different logging clients belonging to different organizations. The logging cloud is managed by a cloud service provider. Only those organizations that have signed to the logging cloud's services will transfer information to the cloud. The cloud, for the asking from an organization may also delete log information and perform log rotation. Before the logging cloud can delete or rotate log information it needs an indication from the requester that the latter is allowed to create such a request. The logging client generates such an indication. However, the proof may be given by the logging client to any entity that it needs to authorize.

Log Monitor

These are hosts that are accustomed monitor and review log information. They will generate queries to retrieve log information from the cloud. Supported the log information retrieved, these monitors can perform additional analysis as required. They will conjointly raise the log cloud to delete log information permanently, or rotate logs.

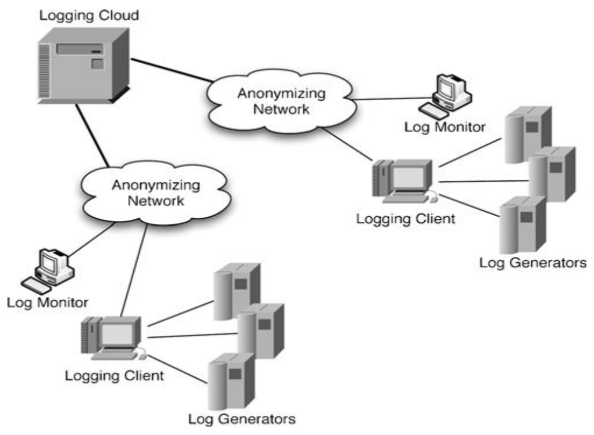

Fig 1 shows System architecture for cloud-based secure logging.

We assume that the organization maintains the log generators and therefore the logging client. The log monitor will be maintained by identical organization or will be a separate entity. The logging client may play the role of a log monitor. We tend to develop our model assumptive that the log monitor may be a separate entity that's trusted by the logging client. After all the logging client and log monitor operate independent of every different, they'll communicate only in an asynchronous manner. This implies that if a logging client needs to send some information to the log monitor (or vice versa); the sender cannot expect the receiver to be online to receive the information. As a result the sender needs to publish the information in some location and therefore the receiver must retrieve the information from there once required. The logging cloud facilitates this communication by receiving and servicing appropriate request.

FORWARD-Secure-Sequential-Aggregate-Authentication

In this section, we have a tendency to brief introduce the elements of FssAgg scheme as they'll be utilized in our secure logging system. We tend to confer with [12, 13] for a lot of formal definition of an FssAgg scheme. We tend to next show however an FssAgg scheme will offer forward-secure stream integrity.

An FssAgg scheme includes the subsequent components:
[FssAgg.kg]-key generation algorithm utilized to generate public/private key-pairs. It additionally takes as input T –the utmost number of your time periods (key environments).
[FssAgg-Assign]-sign-and-aggregate algorithm that takes as input a personal key, a message to be signed and a signature-so-far(an aggregated signature computed up to the current point).It computes a new signature on the input message and combines it with the signature-so-far to provide a new combination signature. The ultimate step in FssAgg.Asig may be a key update procedure FssAgg.Upd that takes as input the signing key for the present period and returns the new sighing key for subsequent period (not exceeding T).We have a tendency to build key update a part of the sign-and-aggregate algorithm so as to get stronger security guarantees (see below).
[FssAgg.Aver]-verification algorithm, which, on input of a supposed combination signature, a group of presumptively signed distinct messages, and a public key , outputs a binary value indicating whether or not the signature is valid.
A secure FssAgg scheme should satisfy the subsequent properties:

- **Correctness:** Any aggregated signature created with FssAgg.Asig Should be selected by FssAgg.Aver.
- **Forward secure combination unforgeability**: nobody, even knowing the present signing key, will build a legitimate FssAgg forgery.

The forward secure mixture unforgeability implies two things:
First, it's append-only –nobody will modify any message generated before the compromise that more implies an FssAgg signature will offer integrity protection for the entire message body. An attacker who compromises a singer has two choices: either it includes the intact aggregate-so-far signature in future aggregated signatures or it ignores the aggregate-so-far signature utterly and begins a latest aggregated signature. What it cannot do is by selection deleting elements of an already-generated mixture signature. This appends-only property resembles the property resembles the property of special write-only disk utilized in traditional log systems. Second it's exhausting to get rid of a component signature while not knowing it - therefore it's immune to deletion (including truncation) attack. They're two very helpful properties and that we can exploit them in our applications.
We claim that FssAgg authentication implies forward-secure stream virtue, i.e.:

Forward Security: During an FssAgg scheme, secret singing key updated through a one-approach function. An attacker is therefore unable to recover previous keys from the present (compromised) key and so unable to forge signatures from previous intervals.
Stream Security: The consecutive aggregation method in an FssAgg scheme preserves the order of messages in order that it provides stream security; so, re-ordering of messages is not possible.
Integrity: any insertion of latest messages additonally as modification and deletion of existing messages can render the ultimate combination unchecked.
Armed with this implication, we are able to currently construct a secure logging system from any FssAgg authentication scheme.

A. Algorithms for Accurating Log Views and Records:
As log information at rest, the log views should be correct, complete and compact. These properties are necessary, as if log views don't fulfill them, the results of auditing such log views cannot be thought of correct.
Definition 1 (Accuracy, Completeness and Compactness of Log Views). Let Log File be a log file and i an identifier. A log view L = T, M for I obtained from Log File is correct iff T contains the precise payloads of the log messages sent to the BBox;
T is Complete iff T encompasses the payloads of all the log messages received by the BBox associated with I. L is compact iff T contains only the entries associated with I.
The generation of log views is completed by algorithm1 that selects the entries from the log file consistent with the index I.
Theorem 1. Algorithm 1 outputs correct, complete and compact log views.
Accuracy, completeness and compactness of log views follow from tamper proof upon the idea that the attacker model is analogous to it of log information at rest that the personal key of the BBox isn't notable by the attacker which the devices communicate each event occurring within the system.

Proof. Let LogFile be a log file consisting of a sequence of entries E0. . . en constructed consistent with Section 3 and T be the audit-trail of the log read generated for I. T is made by linearly looking for entries E such Hash(I) = E.HI.Since the entry authentication check detects tampering makes an attempt (modification, insertion and deletion), log views are only generated if the source log file passes the authentication test. Specifically, this ensures that:
• No payload was modified, in order that accuracy of log views is given.
• No index was modified or replaced, and no entry was deleted, in order that all the entries for I are thought-about for selection (Line 8), providing for completeness.
• No entry was affixed to Log File, offering for compactness.
Hence, algorithm 1 produces correct, complete and compact log views

Description of the scheme

We utilize the subsequent notation from here onwords:
-Li: i-th message, i.e., the i-th log entry. (We assume that log entries are time –stamped and generally have a well-defined format).
-k: k-bit full-domain hash function with strong collision resistance :$\{ 0, 1\}^{k} \rightarrow \{0, 1\}^{k}$.
-*H*: one-way hash function with strong collision resistance and arbitrarily long in-put: $\{0, 1\}^{*} \rightarrow \{0, 1\}^{k}$.
- *MAC*: secure MAC function mac:$\{0,1\}k\{0,1\}\}^{*} \rightarrow \{0, 1\}^{t}$ that, on input of a *k*-bit key *x* and an arbitrary message *m*, outputs a *t*-bit *macx*(*m*).
– UPD: key update frequency

Seekable sequential key generators:

Consider a host that uses SKG's keys Ki to authenticate endlessly incurring log messages. A second copy of the same SKG instance would be run by the log auditor. From time to time the latter might want to check the integrity of an arbitrary selection of those messages6. Observe that this situation doesn't extremely correspond to the setting : While the higher SKG copy would possibly represent the host that evolves keys within the expected linear order Ki ! Ki+1, the auditor (running the independent second copy) would really want non-sequential access to SKG's keys.

Functionality and syntax
When comparison to regular SKGs, the distinctive property of seekable sequential key generators (SSKG) is that keys Ki are often computed directly from initial state st0 and index i, i.e., while not executing the Evolve procedure i times. The corresponding new algorithm, Seek, and its relevance the opposite SKG algorithms is visualized in Figure 2. For reasons that may become clear later, once extending SKG's syntax towards SSKG, additionally to introducing the seek algorithm we tend to additionally had to slightly adapt the signature of the GenSKG algorithm:
A seekable sequential key generator is a tuple SSKG = {GenSSKG, GenState0, Evolve, Seek, GetKeyg} of efficient algorithms as follows:
--GenSSKG(1). On input of security parameter 1_, this algorithm outputs a set par of public parameters and a seeking key sk.
--GenState0, Evolve, GetKey as for SKGs.
-- Seek(sk; st0;m). On input of seeking key sk, initial state st0, and m 2 N, this deterministic algorithm returns a state stm.

ISSN 2278-3091

International Journal of Advanced Trends in Computer Science and Engineering, Vol.2 , No.6, Pages : 96-101 (2013)
Special Issue of ICETEM 2013 - Held on 29-30 November, 2013 in Sree Visvesvaraya Institute of Technology and Science, Mahabubnagar – 204, AP, India

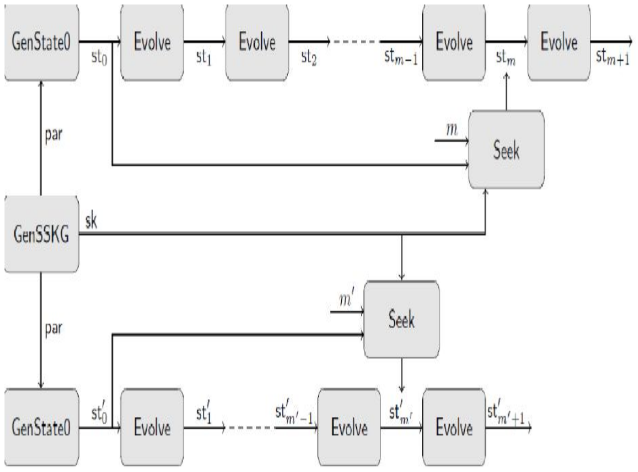

Fig. 2 shows Interplay of the different SSKG algorithms. The figure shows two independent SSKG instances running in parallel. Given seeking key sk and respective instance's initial state st0, one can seek directly to any arbitrary state stm. As in SKGs, GetKey algorithm can be applied to any intermediate state sti to derive key Ki.

Experimental Settings

The actual cloud server is employed as a multithreaded server that accepts at a time connecting log clients and log monitors. Our implementation is such it is simply deployed on any existing cloud system. We've got used a MySQL database engine to store log information. The selection of MySQL is justified by the small batch table schema (only some columns) that takes advantage of the extremely quick MySQL tuple fetch capabilities. We tend to store information within the remote MySQL database as byte representation of Java objects via JDBC connection. The batches are keeping within the cloud database indexed by the upload-tag. we tend to assume that in actual deployment scenario the database back-ends can reside on separate physical machines (regardless of virtualization) divide from the client serving cloud application thus as to alleviate the file I/O load on the cloud application machine (which can be additionally clustered on multiple physical nodes).

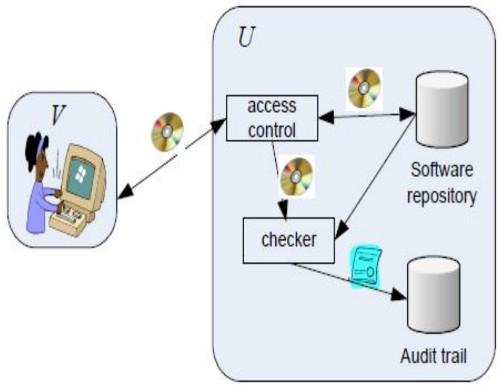

Fig 3 shows Secure Version Control System

Our experiments include two identical machines with a 2.13 gigahertz Intel Core 2 Duo CPU and 200GB memory. Both machines were running Ubuntu 12.04 64-bit Linux with 3.8.8 kernel and were stationed on a similar LAN. The primary machine was running the logging client application, while the second machine was running the syslog-ng application that was configured to make fixed-size (100 bytes) log records

Conclusion

In this paper, we tend to identify some problems in current secure logging techniques. We tend to then proposed two concrete schemes to produce forward-secure stream integrity for logs generated on untrusted machines. Our method assistance forward security. Each of our proposed schemes provides sensible secure logging while not reliance on trusty third parties or secures hardware. Our schemes are supported the recent proposed FssAgg authentication schemes wherever a singular authentication tag is utilized to guard the integrity of underlying message body. We have a tendency to then thought-about the notion of immutability that's required to facilitate quicker verification of individual log entries. We have a tendency to evaluate the performance of our schemes and report on experience with a prototype implementation among a public domain versioning control system.

Although the protection of proposed schemes rests entirely on recently proposed techniques (i.e., [12], [13]), we'd like to construct separate security proofs for every scheme. Moreover, we've got to conduct in depth experiments, and maybe trace-driven simulations, to higher perceive the performance of our schemes. Finally, we will investigate different signature schemes which may be used for constructing additional efficient public-verifiable techniques.

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. W. Hommel, S. Metzger, H. Reiser, and F. von Eye, "Log file management compliance and insider threat detection at higher education institutions," in Proceedings of the EUNIS'12 congress, Oct. 2012, pp. 33-42

6. The SASER-SIEGFRIED Project Website. [retrieved: 03.04.13]. [Online]. Available: http://www.celtic-initiative.org/Projects/Celtic-Plus-Projects/2011/SASER/SASER-b-Siegfried/saser-b-default.asp
7. C. Lonvick, *The BSD Syslog Protocol*, Request for Comment RFC 3164, Internet Engineering Task Force, Network Working Group, Aug. 2001.
8. D. New and M. Rose, *Reliable Delivery for Syslog*, Request for Comment RFC 3195, Internet Engineering Task Force, Network Working Group, Nov. 2001.
9. M. Bellare and B. S. Yee, "Forward integrity for secure audit logs," Dept. Comput. Sci., Univ. California, San Diego, Tech. Rep., Nov. 1997.
10. BalaBit IT Security (2011, Sep.). *Syslog-ng—Multiplatform Syslog Server and Logging Daemon* [Online]. Available: http://www.balabit.com/network-security/syslog-ng
11. J. Kelsey, J. Callas, and A. Clemm, *Signed Syslog Messages*, Request for Comment RFC 5848, Internet Engineering Task Force, Network Working Group, May 2010.
12. Ma, D., Tsudik, G.: Forward-secure sequential aggregate authentication. In: Proceedings of IEEE Symposium on Security and Privacy 2007. (May 2007).
13. Ma, D.: Practical forward secure sequential aggregate signatures. In: ACM Symposium on Information, Computer and Communications Security (ASIACCS'08). (March 2008).
14. Rafael Accorsi "A Secure Log Architecture to Support to Remote Auditing "; 57(2013): pp 1578-1591.

[1]V.Vijayachandra Rao received the bachelor's degree in 2011 from JNTU Anantapur. He is currently pursuing the Master's degree in CSE in the college of JNTUACEP.

[2]Dr.A. Suresh Babu received the PhD degree in Information Extraction Systems in Data Mining from the University of JNTU Anantapur in 2013. He is an assistant professor at the Jntu college of Engineering, Pulivendula, Kadapa, Andhra Pradesh, India. His research interests include Data Mining and Cloud Computing.