This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2482484, IEEE Transactions on Dependable and Secure Computing

Towards Building Forensics Enabled Cloud Through Secure Logging-as-a-Service

Shams Zawoad, Amit Kumar Dutta, and Ragib Hasan

Abstract—Collection and analysis of various logs (e.g., process logs, network logs) are fundamental activities in computer forensics. Ensuring the security of the activity logs is therefore crucial to ensure reliable forensics investigations. However, because of the black-box nature of clouds and the volatility and co-mingling of cloud data, providing the cloud logs to investigators while preserving users' privacy and the integrity of logs is challenging. The current secure logging schemes, which consider the logger as trusted cannot be applied in clouds since there is a chance that cloud providers (logger) collude with malicious users or investigators to alter the logs.

In this paper, we analyze the threats on cloud users' activity logs considering the collusion between cloud users, providers, and investigators. Based on the threat model, we propose Secure-Logging-as-a-Service (*SecLaaS*), which preserves various logs generated for the activity of virtual machines running in clouds and ensures the confidentiality and integrity of such logs. Investigators or the court authority can only access these logs by the RESTful APIs provided by SecLaaS, which ensures confidentiality of logs. The integrity of the logs is ensured by hash-chain scheme and proofs of past logs published periodically by the cloud providers. In prior research, we used two accumulator schemes Bloom filter and RSA accumulator to build the proofs of past logs. In this paper, we propose a new accumulator scheme – *Bloom-Tree*, which performs better than the other two accumulators in terms of time and space requirement.

Index Terms—Information Security, Computer Crime, Data Security, Forensics

1 Introduction

Cloud computing has opened a new horizon of computing for business and IT organizations by offering unlimited infrastructure resources, very convenient pay-as-you-go service, and low cost computing. The rapid adoption of cloud computing has effectively increased the market value of clouds which crossed the $100 billion milestone in 2013 [1] and it will continue to grow in the future [2], [3], [4]. While cloud computing is attractive as a cost-efficient and high-performing model, today's cloud infrastructures often suffer from security issues [5], [6], [7], especially with regards to computer forensics [8], [9], [10], [11], [12]. However, the availability of massive computation power and storage facilities at very low costs can also motivate a malicious individual to launch attacks from machines inside a cloud [13], or use clouds to store contraband documents [14], [15]. It was reported that to launch a Distributed Denial of Service (DDoS) attack, attackers are now placing a new Linux DDoS Trojan – *Backdoor.Linux.Mayday.g* in compromised Amazon EC2 virtual machines (VMs) and launching attacks from those VMs [16]. For these types of attacks, we need to execute digital forensics procedures in the cloud to determine the facts about an incident. Unfortunately, many of the implicit assumptions made in regular forensics analysis (e.g., physical access to hardware) are not valid for cloud computing. Hence, for cloud infrastructures, a special branch of digital forensics has been proposed by researchers – *Cloud Forensics*.

Activity logs of cloud users can reveal the actions taken by a user using cloud infrastructures. Hence, logs are crucial evidence to prosecute a suspect. However, collecting logs from the cloud infrastructure is extremely difficult because cloud users or investigators have very little control over the infrastructure. Currently, there is no way for investigators to collect logs from a terminated VM; they need to depend on the Cloud Service Providers (CSP) to collect logs from the cloud. However, investigators need to believe the CSPs blindly, as they cannot verify whether the CSPs are providing valid logs or not. Very often, the experienced attackers first attack the logging system [17], [18]. While the necessity of logs is indisputable in forensic investigations, the trustworthiness of this evidence will remain questionable if we do not take proper measures to secure them. An adversary can try to host a botnet server, spam email server, or phishing websites in cloud VMs and he can remove all the traces of these malicious activities later by tampering with the logs. Conversely, investigators can also be malicious and they can alter the logs before presenting to the court. The following hypothetical scenario can illustrate the specific problem that we intend to solve:

Bob is a successful businessman who runs a very popular shopping website. Mallory, a competitor of Bob, rented some VMs hosted in a cloud and launched a Distributed Denial of Service (DDoS) attack on Bob's shopping website from those rented VMs. As a result of the DDoS attack, Bob's website was down for an hour, which had quite a negative impact on Bob's business in terms of profit and goodwill. Consequently, Bob asked a forensic investigator to investigate the case. Analyzing the logs of Bob's web server, the investigator found that the

• *Shams Zawoad and Ragib Hasan are affiliated with the Department of Computer and Information Science, University of Alabama at Birmingham, Birmingham, AL 35294-1170 USA. E-mail: Shams Zawoad* zawoad@cis.uab.edu*, Ragib Hasan* ragib@cis.uab.edu*
Amit Kumar Dutta is affiliated with VMware, Palo Alto, CA 94304 USA. Email: amitd@vmware.com

This research was supported by the National Science Foundation CAREER Award CNS-1351038, a Google Faculty Research Award, and the Department of Homeland Security Grant FA8750- 12-2-0254.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2482484, IEEE Transactions on Dependable and Secure Computing

website was flooded by some IP addresses that are owned by a cloud service provider. Eventually, the investigator issued a subpoena to the cloud provider to provide him the network logs for those particular IP addresses.
The possible outcomes in this scenario are:

• *Case 1:* Mallory colluded with the Cloud provider to alter logs. Since the investigator had no way to verify the correctness of the logs, Mallory would remain undetected.

• *Case 2:* Mallory terminated her rented machines and left no traces of the attack. Hence, the cloud provider would fail to provide any useful logs to the investigator.

• *Case 3:* Mallory could claim that investigator colluded with the cloud provider and altered the logs to frame her.

To mitigate the challenges discussed in the above scenario, we propose the notion of Secure-Logging-as-a-Service (SecLaaS). Since data residing in the VM are volatile (cannot be sustained without power), SecLaaS collects logs from the VMs and stores them in a persistent storage to resolve the issue of volatility of logs. While preserving, it encrypts confidential data and maintains a hash-chain of the logs to protect the original sequence of logs. After a certain epoch (which can be configured according to the required security level and be specified in the service level agreement), SeclaaS generates the proofs of past logs (PPL) and makes the proofs publicly available. A CSP can only alter the logs of an active epoch for which the PPL has not been published yet. However, once the PPL for an epoch is made publicly available, CSP, users, or investigators cannot modify/add/remove/reorder the logs of that epoch and of any prior epochs. Hence, with the help of hash-chain and PPL, SecLaaS provides forward security of the logs of all past epochs. Finally, SecLaaS exposes RESTful APIs to ease the process of log collection by investigators or the court authority.

Implementing SecLaaS in clouds will enable investigators to collect trustworthy logs of cloud-based malicious activities and present those to the court authority. The security properties ensured by SecLaaS can help CSPs to establish trust with the cloud users. By preserving activity logs securely, SecLaaS can also make clouds more auditable, which is an essential requirement for various regulatory acts, e.g., Sarbanes-Oxley (SOX) [19] or The Health Insurance Portability and Accountability Act (HIPAA) [20]. Therefore, a CSP can attract more customers by establishing trust and regulatory compliance using SecLaaS. The additional cost for integrating SecLaaS can thus be compensated by increasing the customer base. Additionally, a secure logging service enabled for all users, can prevent a malicious user from launching malicious activities from clouds.

Researchers proposed read-only APIs and web-based management consoles to ease the log acquisition process for better forensics support [8], [21]. However, these works do not preserve the confidentiality and integrity of logs when a CSP is dishonest. Previous schemes that can preserve the privacy and integrity of logs from external attackers are not designed to protect the integrity when the logger (in this case the cloud) itself is malicious [17], [18], [22], [23], [24], [25]. The existing forward secure logging mechanism relies on an initial secret, which an external attacker cannot access. However, in our threat model, the cloud provider is malicious and knows the initial secret; hence, the provider can always tamper with the logs starting from the very first log entry. Moreover, to verify the existing forward secure chain, the investigator needs to collect all the logs of a suspect starting from the very beginning of the chain, even though the investigator may need logs of only few hours or days. The existing secure logging schemes can only protect logs from being altered by an investigator, since the investigator cannot access the initial secret. However, if the investigator colludes with the CSP, the existing schemes fail to detect alteration by the investigator. For a successful forensic scheme based on logs, we resolve these issues in a secure and trustworthy manner.

Contributions: The contributions of this paper are as follows:
• We propose a scheme of revealing cloud users' logs for forensics investigation while preserving the confidentiality of users' logs from malicious cloud employees or external entities.

• We introduce Proof of Past Log (PPL) – a tamper evident scheme to prevent CSPs or investigators from manipulating the logs after-the-fact.

• We present a variation of the Bloom filter scheme – *Bloom-Tree*, which provides better security and performance compared to the traditional Bloom filter approach.

• Our evaluation of the proposed scheme in a test-bed, which is built on top of an open source cloud computing platform – OpenStack suggests that it can be feasible to implement SecLaaS in real clouds.

This article is an extended version of our previous work [26]. In this extended version, we propose a new accumulator scheme – *Bloom-Tree*. We present the performance analysis of the new scheme and our experimental results suggest that the Bloom-Tree outperforms the regular Bloom filter approach in terms of security, storage requirement, and execution time. We also present the design and implementation of the RESTful API for accessing logs, an example of an API request and response, and how we ensure the security of the APIs.

Organization: The rest of this paper is organized as follows. Section 2 provides the background and the challenges of cloud forensics in terms of logging. In Section 3, we present the related research work. Section 4 describes the adversary's capabilities and possible attacks on logging-as-a-service. Section 5 presents our proposed SecLaaS scheme, and Section 6 provides the security analysis of the scheme. Section 7 presents the implementation and performance evaluation of our scheme on OpenStack. Section 8 discusses the usability of our proposed schemes and finally, we conclude in Section 9.

2 Background and Challenges

In this section, we present an overview of digital forensics and cloud forensics and discuss the challenges of logging-as-a-service for cloud forensics.

2.1 Digital Forensics

Digital forensics is the process of preserving, collecting, confirming, identifying, analyzing, recording, and presenting crime scene information. Figure 1 illustrates the process flow of digital forensics. According to the definition by the National Institute

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2482484, IEEE Transactions on Dependable and Secure Computing

Fig. 1: Process Flow of Digital Forensics [11]

for Standards and Technology (NIST), digital forensics is an applied science to identify an incident, collection, examination, and analysis of evidence data [27]. While executing the above processes, maintaining the integrity of the information and strict chain of custody for the data is mandatory. Wolfe *et al.* defines digital forensics as a methodical series of techniques and procedures for gathering evidence, from computing and storage devices that can be presented in a court of law in a coherent and meaningful format [28].

2.2 Cloud Forensics

NIST defines cloud forensics as "*the application of scientific principles, technological practices and derived and proven methods to reconstruct past cloud computing events through identification, collection, preservation, examination, interpretation and reporting of digital evidence*" [29].

As cloud computing is based on extensive network access, and as network forensics handles forensic investigation in private and public network, Ruan *et al.* defined cloud forensics as a subset of network forensics [30]. Different steps of digital forensics, as shown in Figure 1 vary according to the service and deployment model of cloud computing. For example, the evidence collection procedures in Software-as-a-Service (SaaS) and Infrastructure-as-a-Service (IaaS) are different. For SaaS, we solely depend on CSPs to get application logs, while in IaaS, we can acquire the virtual machine image from customers, and can enter into examination and analysis phase. In the private cloud deployment model, we have physical access to the digital evidence. Unfortunately, we rarely can get physical access to the evidence in the public cloud deployment model.

2.3 Challenges

Reduced Level of Control in Clouds: In traditional computer forensics, investigators have full control over evidences (e.g., router logs, process logs, hard disk). Unfortunately, we extensively depend on CSPs to acquire logs from clouds. Availability of the logs varies depending on the service models. Table 1 shows the control of customers in different layers for the three different service models – IaaS, PaaS, and SaaS. Cloud users have highest control in IaaS and least control in SaaS. This physical inaccessibility of the evidence and lack of control over the system make evidence acquisition challenging in cloud forensics. In SaaS, customers do not get any log of their system, unless the CSP provides the logs. In PaaS, it is only possible to get the application logs from customers. To get the network log, database log, or operating system log, we need to depend on cloud providers.

In IaaS, customers do not have access to network or process logs. Several other problems come along with the control issue. For example, dependency on CSPs brings the honesty issue of CSPs' employees, who are not certified forensic investigators. CSPs can always tamper with the logs as they have the full control over the generated logs. Additionally, CSPs are not always obligated to provide all the necessary logs, when it conflicts with their data protection policies.

Layers	SaaS	PaaS	IaaS
Network	×	×	×
Servers	×	×	×
OS	×	×	✓
Data	×	×	✓
Application	×	✓	✓
Access Control	✓	✓	✓

TABLE 1: Customers' control over different layers in different service model

Volatility of Logs: Volatile data cannot be sustained without power. Volatile data of a VM includes all the logs stored in that VM, e.g., SysLog, registry logs, and network logs. Without preserving the snapshots of the VM, it is not possible to retrieve these logs from a terminated VM. However, it is assumed that an attacker would not choose to preserve snapshots to remain clean. Hence, if a malicious user turns off a VM instance after launching an attack, then these logs will be unavailable.

Multi-tenancy: In cloud infrastructures, multiple VMs can share the same physical infrastructure, i.e., logs for multiple customers may be co-located, which is different from the traditional single owner computing system. Hence, depending on the acquisition mechanism, multiple users data can be mingled logically and physically. An alleged user can claim that the logs contain information of other users. The investigator then needs to prove it to the court that the provided logs indeed belong to the malicious user. Moreover, we need to preserve the privacy of other tenants.

Accessibility of Logs: Logs generated in different layers of cloud infrastructures are required to be accessible to different stakeholders of the system. For example, system administrators need relevant logs to troubleshoot the system; developers need their required logs to fix the bugs of an application; forensic investigators need logs, which can help in their investigation. Hence, there should be some access control mechanism, so that everybody gets what they need exactly – nothing more, nothing less and obviously, in a secure way. We should not expect that a malicious cloud employee, who can violate the privacy of users, can get access to users' log information.

Decentralization: In cloud infrastructures, log information is not located at any single centralized log server; rather logs are decentralized among several servers. Multiple users' log information may be co-located, or spread across multiple servers. Moreover, there are several layers and tiers in the cloud architecture, where logs are generated in each tier. For example, application, network, operating system, and database – all of these layers produce valuable logs for forensic investigation. Collecting logs from these multiple servers and layers, and providing them to the investigators in a secure way is extremely challenging. The decentralized nature of clouds also brings the challenge of clock synchronization. According to Zimmerman *et al.*, if the client-side log files do not match the time stamps on provider-side log files, it will be difficult to defend such evidence in the court [31].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2482484, IEEE Transactions on Dependable and Secure Computing

Absence of a Standard Format of Logs: To analyze logs most effectively, it will be necessary to have a standard format of the logs. Unfortunately, till now, there is no standard format of logs for cloud infrastructures. Logs are available in heterogeneous formats – from different layers of a cloud to different service providers. Moreover, not all the logs provide crucial information for forensic purpose, e.g., by whom, when, where, and why some incidents occurred. This is an important bottleneck to provide a generic solution for all the cloud service providers and for all types of logs.

3 Related Work

Because of the necessity of trustworthy logs in forensics investigation, several researchers have explored this problem across multiple dimensions. As a solution for forensic investigation in clouds, Zafarullah *et al.* proposed logging provided by the OS and the security logs [32]. They set up a cloud computing environment using Eucalyptus an using Snort, Syslog, and Log Analyzer (e.g., Sawmill), they were able to monitor the behavior of Eucalyptus and to log all internal and external interaction of Eucalyptus components. For their experiment, they launched a DDoS attack from two virtual machines and from the logs on the Cloud Controller (CC) machine, they were able to identify the attacking machine IP, browser type, and content requested. From these logs, it is also possible to determine the total number of VMs controlled by a single Eucalyptus user and the VMs communication patterns. To provide logs for cloud forensics, Patrascu *et al.* proposed a cloud forensics module in the management layer of cloud infrastructures. This module communicates with different stacks of the kernel, such as virtual file system, network stack, system call interface, etc. to acquire logs [33]. However, the security and the availability of the logs to forensics investigators have not been ensured by these works.

To make the network, process, and access logs available to customers, Bark *et al.* proposed to expose read-only APIs by CSPs [8]. In the same context, Dykstra *et al.* implemented FROST, a tool for OpenStack to collect virtual disks, API logs, and guest firewall logs [21]. These works mainly focus on making the logs easily available; it has not been shown how to protect users' privacy and integrity of logs from a malicious CSP and investigators. In [34], Marty provided a guideline, which tells us to focus on when to log, what to log, and how to log. The answer of when to log depends on the use-cases, such that business relevant logging, operations based logging, security (forensics) related logging, and regulatory and standards mandates. At minimum, he suggested to log the time-stamps record, application, user, session ID, severity, reason, and categorization, so that we can get the answer of what, when, who, and why (4 W).

Secure logging has been discussed in several research works [22], [23], [24], [25], [35]. In the threat model of these works, researchers considered attacks on privacy and integrity from external entity on a logging server. However, none of these works focused on secure logging in the cloud environment, especially where the logger itself (cloud provider) is dishonest and also did not consider the collusion between different entities. To detect temporal inconsistencies in a VM's timeline, Thorpe *et al.* developed a log auditor by using the 'happened-before' relation [36] in the cloud environment [37]. However, not all of the events occurred inside clouds have the happened-before relation between them. For example, browsing two independent webpages cannot be bound with this relation. Moreover, the proposed systems do not consider the threat where a CSP is dishonest.

Organizations, who find the cost of developing and maintaining a secure logging infrastructure unbearable, can delegate the task of log management to clouds to reduce the cost. Ray *et al.* proposed a framework to serve this purpose [38]. They proposed a cryptographic protocol to address integrity and confidentiality issues with storing, maintaining, and querying log records in a server operating in the cloud. Transmission of log records between log generators, such as computers and cloud servers is made secure through the use of an anonymizing network.

Researchers have also proposed several variations of regular accumulator schemes. The space-code Bloom filter is one of such variations [39], which traces the number of occurrences of an element as well as membership checking. Papamanthou *et al.* proposed a tree-based construction for membership verification based on the RSA accumulator, which was the building block of developing authenticated hash-table [40]. Koloniari *et al.* proposed two variations of Bloom filter: breadth Bloom filters and depth Bloom filters [41]. These schemes are useful to represent hierarchical data like *xml* and support path expression queries, which is used in resource discovery of peer-to-peer networks. Our proposed Bloom-Tree is a variation of the Hierarchical Bloom Filter (HBF) introduced in [42], which is used as a compact hash-based, payload digesting data structure and to find source and destination from a given excerpt of a networks packet's payload. In HBF, the payload of a packet is divided into blocks and inserted into the hierarchy of Bloom filters from bottom-up. However, the way we build the Bloom-Tree and apply it to ensure the integrity of logs is novel.

The solution proposed by Marty provided a guideline for logging standard. Zafarullah *et al.* showed that it is possible to collect necessary logs from the cloud infrastructure, while Bark *et al.* and Dykstra *et al.* proposed public APIs or management console to mitigate the challenges of log acquisition. However, none of them proposed any scheme for storing the logs in a cloud and making it available publicly in a secure way. Dyskstra *et al.* mentioned that the management console requires an extra level of trust and the same should hold for APIs. In this paper, we take the first step towards providing a solution to mitigate these challenges. Combining all the previous solutions and our scheme will help to make clouds more forensics-friendly.

4 Threat Model

In this section, we first define the important terms of our proposed system. Then, we describe the attacker's capability, possible attacks on logs, and the security properties that a secure cloud log service should provide.

4.1 Definition of terms

• *Log:* A log can be the network log, process log, operating system log, or any other log generated in the cloud for a VM.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2482484, IEEE Transactions on Dependable and Secure Computing

• *Proof of Past Logs (PPL):* The *PPL* contains the proof of logs to ensure the integrity of logs.

• *Log Chain (LC):* The *LC* maintains the chronological ordering of logs to protect the logs from reordering.

• *CSP:* A Cloud Service Provider (CSP) is the owner of a public cloud infrastructure, who generates the *PPL*, makes it publicly available, and exposes APIs to collect logs.

• *User:* A user is a customer of the CSP, who rents VMs provided by the CSP. A user can be malicious or honest.

• *Investigator:* An investigator is a professional forensic expert, who needs to collect necessary logs from cloud infrastructures in case of any malicious incident.

• *Auditor:* Usually, an auditor will be the court authority that will verify the correctness of the logs using *PPL* and *LC*.

• *Intruder:* An intruder can be any malicious person including insiders from CSP, who wants to reveal user's activity from the *PPL* or from the stored logs.

4.2 Attacker's Capability

In our threat model, users, investigators, and CSPs can collude with each other to provide fake logs to the auditor. We assume that a CSP is honest at the time of publishing the *PPL*, but during an investigation, they can collude with a user or an investigator and provide tampered logs, for which the *PPL* has already been published. A user cannot modify the logs by himself, but he can collude with a CSP to alter the logs. An investigator can also be dishonest and collude with users or a CSP to frame an honest user, or to save a malicious user from an accusation. A malicious investigator can collude with a CSP to alter the logs or can tamper with the logs after collecting the logs from the CSP.

4.3 Possible Attacks

Log modification: A dishonest CSP, while colluding with a user or an investigator can modify the logs, either to save a malicious user or to frame an honest user. If an investigator is not trustworthy, he can also tamper with the logs before presenting the logs to the court. There can be three types of contamination of logs: 1) removal of crucial logs, 2) plantation of false logs, and 3) modification of the order of the logs.

Privacy violation: As the PPLs are publicly available on the web, any malicious person can acquire the published PPL and try to learn about the logs from the proof. A malicious cloud employee, who has access to the log storage, can identify the activity of the user from the stored logs.

Repudiation by user: As data are co-mingled in the cloud, a malicious user can claim that the logs contain another cloud user's data.

Repudiation by CSP: A malicious CSP can deny a published PPL after-the-fact.

4.4 System Property

While designing SecLaaS, we focus on ensuring secure preservation of cloud users' logs in a persistent storage, and to prevent any malicious entity from producing fake proof of past logs. A false *PPL* attests the presence of a log record for a user, which the user does not actually own. Once the proof has been published, the CSP can neither modify the proof nor repudiate any published proof. Additionally, we must prevent false implications by malicious forensic investigators. Hence, a secure log service for clouds should possess the following integrity (I) and confidentiality (C) properties:

I1: A CSP cannot remove a log entry from the storage after publishing the PPL.

I2: A CSP cannot change the order of a log from its actual order of generation.

I3: A CSP cannot plant false log after-the-fact.

I4: An investigator cannot hide or remove a log entry at the time of presenting logs to court.

I5: An investigator cannot change the actual order of a log entry at the time of presenting evidences to court.

I6: An Investigator cannot present fraudulent logs to the court.

I7: CSPs cannot repudiate previously published proofs of logs.

C1: An adversary cannot recover logs from the published proofs of logs.

C2: A malicious cloud employee will not be able to recover confidential information from the log storage.

5 The SecLaaS Scheme

In this section, we discuss how SecLaaS stores logs securely, provides APIs to forensics investigators, and verifies the integrity of logs. First, we present an overview of the mechanism. Then, we discuss the protocol specific description of the system.

5.1 Overview

A malicious cloud user can attack other VMs running inside the cloud or can attack an external computer outside the cloud. An attacker can also attack the *Node Controller* (NC), which hosts the virtual machine instances and manages the virtual network endpoints to launch some side channel attacks [43]. Some of the side channel attacks are: measuring cache usage to identify cryptographic keys, identifying passwords by observing the keystroke timings, and launching denial of service attacks on the shared resources. Figure 2 presents an overview of storing the logs in a secured way and making the logs available to forensic investigators in case of such attacks.

As VMs are running inside the NC, we can trace malicious activities of VMs from various logs generated in the NC. For each running VM, SecLaaS first extracts various kinds of logs from the NC and stores them in a persistent log database. Hence, terminating the VM will not prevent our system to provide useful logs during investigation. While saving logs in the log database, SecLaaS ensures the integrity and confidentiality of the logs. After storing a log entry in the log database, SecLaaS additionally preserves the proof of that log entry in the proof database. An investigator can acquire the necessary logs of a particular IP by calling APIs provided by SecLaaS. In order to prove the logs as admissible evidence, investigators can provide the proofs of past logs along with the logs.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2482484, IEEE Transactions on Dependable and Secure Computing

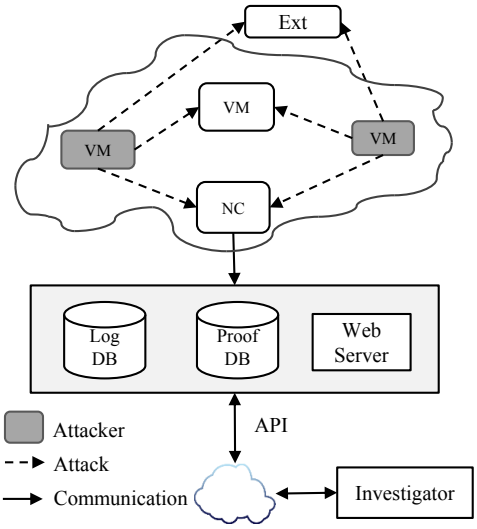


Fig. 2: Overview of SecLaaS

5.2 System Details

Notation and Assumptions: $H(M)$ is a collision-resistant one-way hash function, which produces a hash of a message M. The $E_{P_K}(M)$ function encrypts a message M using the public key P_K. The $Sig_{S_K}(M)$ function generates signature of a message M using the secret key S_K. We assume that the law enforcement agencies (LEA) and the CSP have setup their secret keys and public keys and distribute the public keys. The secret key and public key of the CSP are S_{KC} and P_{KC}, and for the LEA these are S_{KA} and P_{KA}.

5.2.1 Log and Proof Insertion

In Figure 3, we illustrate the detail flow of log retrieval, secured log insertion, and PPL generation. We use the network log as an example to describe the entire system. However, after step (*b*), the system will be identical for any type of logs. Below are the details of the system:

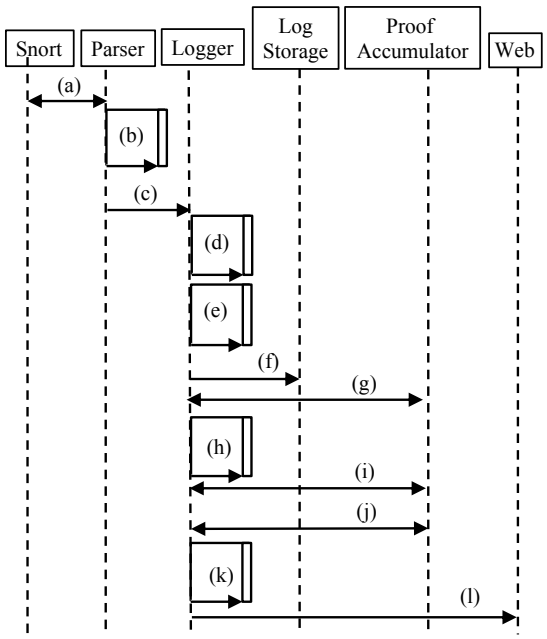


Fig. 3: Process Flow of Retrieving Log and Storing the PPL

(a) The parser module first communicates with the log sources to collect different types of logs. For example, the parser can listen to Snort [1], a free lightweight network intrusion detection system to store network logs.

(b) After acquiring logs from different sources, the parser then parses the collected logs and generates a Log Entry *LE*. For example, a Log Entry *LE* for a network log could be defined as follows:

$$LE =< FromIP, ToIP, T_{\mathrm{L}}, Port, UserId >, \quad (1)$$

where T_L is the time of the network operation in UTC.

(c) The parser module then sends the Log Entry *LE* to the logger module to further process the *LE*.

(d) To ensure the privacy of users, the logger module can encrypt some confidential information of the LE using the public key of LEA P_{KA} and generates the Encrypted Log Entry *ELE* as follows:

$$ELE =< E_{P_{KA}}(ToIP, Port, UserId), FromIP, T_{\mathrm{L}} > \quad (2)$$

As searching on encrypted data is costly, some of the fields of *LE* that often appear in search can be unencrypted. For network logs, some crucial information that we can encrypt includes: destination IP (ToIP), and user information (UserId).

(e) After generating ELE, the logger module then creates the Log Chain LC to preserve the correct order of the log entries. The LC is generated as follows:

$$LC =< H(ELE, LC_{\mathrm{Prev}}) >, \quad (3)$$

where $\mathrm{LC_{Prev}}$ is the Log Chain of the log entry that appears before the current log entry.

(f) The logger module now prepares an entry for the persistent log database, which we denote as $DBLE$. The $DBLE$ is constituted of ELE and LC

$$DBLE =< ELE, LC > \quad (4)$$

(g) After creating the $DBLE$, the logger module communicates with the proof storage to retrieve the latest accumulator entry.

(h) In this step, the logger generates the proof of *DBLE*, i.e., the logger creates the membership information of the $DBLE$ for the accumulator. The membership information for $DBLE$ depends on the chosen accumulator scheme (e.g., bit array for the Bloom filter), which we describe in Section 5.3. The logger then updates the last retrieved accumulator entry with this membership information.

(i) The logger module sends the updated accumulator entry to the accumulator storage to store the proof.

(j) At the end of each day, the logger retrieves the last accumulator entry of each static IP, denoted as AE_D.

(k) Using the AE_D, the logger now generates the Proof of Past Log *PPL* as follows:

$$PPL =< AE_D, T_P, Sig_{S_{KC}}(AE_D, T_P) >, \quad (5)$$

where T_P represents the proof generation time, and $Sig_{S_{KC}}(AE_D, T_P)$ is the signature over (AE_D, T_P) using the secret key of the CSP, S_{KC}.

1. http://www.snort.org

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2482484, IEEE Transactions on Dependable and Secure Computing

(l) After computing the *PPL*, the logger module will publish the *PPL* on the web. The *PPL* can be available by RSS feed to protect it from manipulation by the CSP after publishing. We can also build a trust model by engaging other CSPs in the proof publication process. Whenever one CSP publishes a PPL, that PPL will also be shared among other CSPs. Therefore, we can get a valid proof as long as one CSP is honest.

5.2.2 Verification

An investigator first gathers the required logs from the CSP and presents the logs along with the PPL to the court. The verification process, which will be executed by the auditor constitutes of three steps: 1) authenticity verification of the published PPL, 2) integrity verification of individual log entries, and 3) verification of the chronological order of the logs. Among these three steps, only the detail of the second step (integrity verification process) depends on the chosen accumulator scheme.

1) PPL Verification: The verification process, presented in Figure 4 starts from checking the validity of the published Proof of Past Log PPL. The auditor extracts the published AE_D and T_P from the PPL. Then using the signature $Sig_{S_{KC}}(AE_D, T_P)$ and public key of the CSP P_{KC}, the auditor verifies the integrity of the AE_D and T_P. A PPL is valid if the digital signature is valid. The verification process then proceeds to the next step.

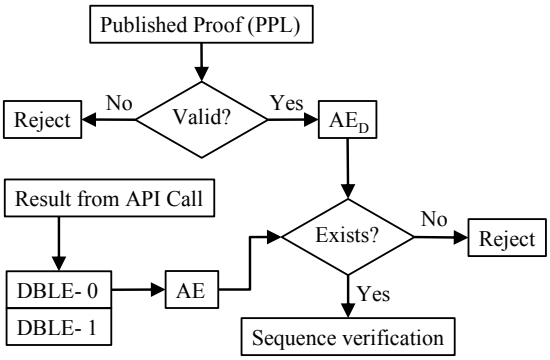


Fig. 4: Log Verification Process Flow

2) Integrity Verification of Log: To verify the integrity of a log entry $DBLE$, the auditor generates the membership information for the $DBLE$, AE and checks whether the $DBLE$ exists in the AE_D using the AE. If exists, then the auditor proceeds towards the log order verification process, otherwise the log entry $DBLE$ is rejected. If the $DBLE$ is the last log entry of an epoch, the AE of the $DBLE$ needs to be equal to the AE_D, otherwise it is rejected.

3) Sequence Verification: Figure 5 illustrates the log order verification process of two consecutive log entries – $DBLE0$ and $DBLE1$, where $DBLE0$ appears immediately before $DBLE1$ in the original sequence of log generation. In Figure 5, $ELE0$ denotes the Encrypted Log Entry of the first log and $ELE1$ represents the same for the second log entry. To verify the correct order, the auditor calculates the Log Chain LCa from the first Log Chain $LC0$ and the second Encrypted Log Entry $ELE1$ according to the following equation.

$$LCa =< H(ELE1, LC0) > \quad (6)$$

If LCa matches with the 2nd Log Chain $LC1$, the auditor accepts the logs, otherwise he rejects it. The auditor executes this process for all $\{DBLE_i, DBLE_j\}$, where $i \epsilon \{0, n-1\}$, $j = i+1$, and n is the number of logs provided for verification.

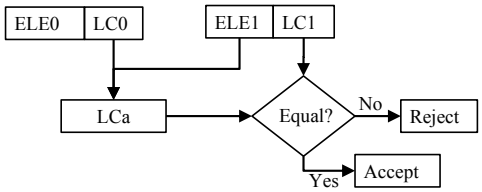


Fig. 5: Log Order Verification Process Flow

5.3 Accumulator Design

We used three accumulator schemes – Bloom filter [44], One-Way Accumulator [45], and our variation of the Bloom filter based accumulator *Bloom-Tree*. The steps from (g) to (k) of log and proof insertion (Figure 3), and the integrity verification of logs work differently for the different accumulator schemes.

5.3.1 Bloom filter:

A Bloom filter is used to check whether an element is a member of a set or not [44]. It stores the membership information in a bit array. Bloom filters decrease the element insertion time and membership checking time. However, it is a probabilistic data structure and suffers from false positives. We can decrease the false positive rate by increasing the size of the bit array.

Proof Creation: To use the Bloom filter as a proof, we use one Bloom filter for one static IP for each day. That means, one Bloom filter stores the proof of all the logs of one static IP for a particular day. In step (g) of Figure 3, the logger module retrieves the latest Bloom filter AE from the proof storage, which holds the bit positions for the previously inserted logs of the day. In step (h), the logger generates k bit positions for the database entry $DBLE$ by hashing the log entry with k different hash functions. The logger then sets the calculated k bit positions of the AE and sends the updated AE to the proof storage. At the end of each day, the CSP retrieves the Bloom filter entry of each static IP AE_D and creates the proof of past log PPL for that day using equation 5.

Integrity Verification of Log: To verify the integrity of a log entry $DBLE$ using the Bloom filter accumulator, the auditor first calculates the k bit positions of the Bloom filter by hashing the $DBLE$ with the k different hash functions. These bit positions will be then compared with the published AE_D. If all the calculated bit positions are set in the published Bloom filter AE_D, then the $DBLE$ is valid. One single false bit position means the log entry is not valid.

5.3.2 One-Way Accumulator:

One-Way accumulator is a cryptographic accumulator, which is based on RSA assumption and provides the functionality of checking the membership of an element in a set [45]. This scheme works with zero false negative and false positive probability. Initially, we need to create the public and private values for the accumulator. The private values are two large prime numbers P, and Q. The public values are N and X,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2482484, IEEE Transactions on Dependable and Secure Computing

where $N = P * Q$ and X is the initial seed, which is a large prime number.

Proof Creation: In step (g) of Figure 3, the logger retrieves the latest accumulator entry AE. If the AE is empty, i.e., no $DBLE$ has been inserted yet on that day, then the AE is generated using the following equation:

$$AE = X^{H_n(DBLE)} \bmod N, \tag{7}$$

where $H_n(DBLE)$ is a numeric hash value of $DBLE$. If the retrieved AE is not empty, the new AE will be generated using the following equation:

$$AE = AE^{H_n(DBLE)} \bmod N \tag{8}$$

The logger module then sends the calculated AE to the proof storage. At the end of the day, the logger retrieves the last accumulator entry AE_D and creates the proof of past log PPL for the day using Equation 5. Additionally, the logger needs to generate an identity for each $DBLE$ and tag it with the $DBLE$. For m number of $DBLE$s on a day, the identity AID_i of the i^{th} $DBLE$ is calculated using the following equation:

$$AID_i = X^{H_n(DBLE_1)..H_n(DBLE_{i-1})H_n(DBLE_{i+1})..H_n(DBLE_m)} \bmod N \tag{9}$$

Integrity Verification of Log: Along with the $DBLE$s, the auditor will be provided with the AID of the $DBLE$. While verifying the validity of the $DBLE_i$, the auditor first computes $(AID_i^{H_n(DBLE_i)} \ mod \ N)$ and compares it with AE_D. If $AE_D = (AID_i^{H_n(DBLE_i)} \ mod \ N)$, the log entry is valid otherwise not.

5.3.3 Bloom-Tree

As a probabilistic data structure, the Bloom filter suffers from false positive (FP) rate. The only way to decrease the percentage of FP is to increase the size of bit array. However, a bit array with larger size introduces space overhead and performance degradation for log insertion and verification. Inspired by [42], we design a data structure – *Bloom-Tree*, which requires a significantly smaller amount of space while ensuring a very low percentage of FP compared to the regular Bloom filter. An example of a *Bloom-Tree* is presented in Figure 6.

Proof Creation: To build a Bloom-Tree, we create a new Bloom filter after every m number of logs. Hence, for n number of logs in a day, there will be $NB = \lceil (n/m) \rceil$ number of Bloom filters. To insert the proof of the i^{th} log entry $DBLE_i$, in step (g) of Figure 3, the logger module first checks whether $i \bmod m = 0$. If $i \bmod m = 0$, a new Bloom filter AE will be created, otherwise the logger retrieves the latest bloom filter AE from the proof storage. Later, using the same approach as described previously for the regular Bloom filter, the logger updates the k bit positions of the AE and sends the updated AE to the proof storage.

To generate the PPL, the logger retrieves the NB Bloom filters for a static IP at the end of each day. At the time of PPL generation, these Bloom filters will then be cumulatively added to a higher order Bloom filter. The accumulators that hold the membership information of the logs, will be purged from the proof storage after the PPL is generated (nodes inside the dotted red region in Figure 6). An intermediate node (nodes inside the solid green region) holds the membership information of its child Bloom filter and is stored in the proof database. The branching factor b of the accumulator can vary according to the volume of logs. The root of this tree will be considered as the AE_D and the logger creates the PPL using this AE_D. To maintain this data structure and use it for integrity verification, each Bloom filter will have its parent's identity and a boolean attribute to denote whether a node is a leaf node.

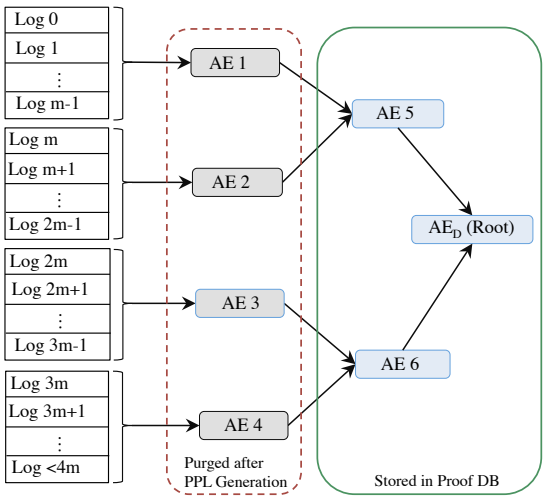


Fig. 6: An example of a Bloom-Tree where the total number of logs is at most $4 * m$.

Integrity Verification of Log: An example of the verification process using the *Bloom-Tree* is illustrated in Figure 7. Besides the AE_D, the auditor needs to be provided with the nodes of the *Bloom-Tree*. To verify the integrity of logs, the logs need to be provided with sets of m logs. Hence, in this approach, the auditor needs to collect some extra logs, though he does not need those. If the auditor actually needs logs in the range of $[s, e]$, he needs to collect logs from $\lfloor s/m \rfloor * m$ to $(\lfloor e/m \rfloor + 1) * m - 1$. For example, to verify the integrity of 114^{th} to 126^{th} logs, the auditor needs to collect logs from 110^{th} to 129^{th} for $m = 10$. Accordingly, the auditor needs to collect $2(m-1)$ extra log entries in the worst case.

The collected logs will then be grouped according to their order of generation, where each group contains m logs. For the m number of logs of the i^{th} set, a new Bloom filter AE_{G_i} is created and all the logs of the i^{th} set are inserted to the AE_{G_i}. Now, the verifier finds the leaf node of the *Bloom-Tree* that contains the AE_{G_i}.

If all of the m logs are valid, we will find a node AE_L that contains the AE_{G_i}. In Figure 7, node $AE_1 1$ (marked as green) is one such leaf node. Consequently, the parent of the AE_L is identified and the auditor verifies whether AE_L exists in its parent accumulator. The process continues until it reaches the AE_D. A positive result while checking with AE_D, confirms that all of the logs of the i^{th} set are valid. The same procedure will be applied to other sets of logs.

5.4 API Design

We designed and exposed the API using RESTful (Representational State Transfer) web services [46]. REST-style web services are stateless, which enables multiple servers to handle

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2482484, IEEE Transactions on Dependable and Secure Computing

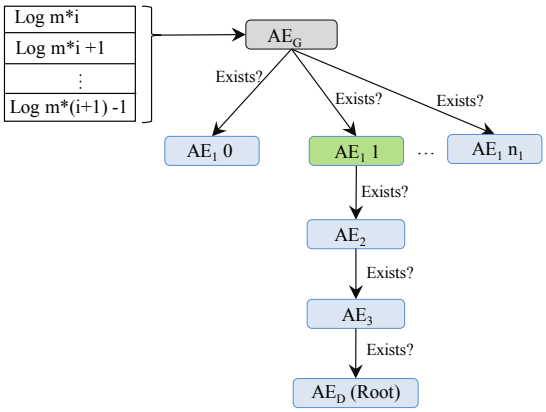


Fig. 7: Integrity Verification using the Bloom-Tree

multiple requests to improve the scalability of servers. By using REST, we can design web services that focus on a system's resources. For logging as a service, the resource is log and proofs of logs. RESTful web services are implemented by using HTTP standard methods (e.g. GET, PUT, POST, DELETE). According to the REST principle, to retrieve a resource, GET operation is used on that resource. Caller of a REST service can pass different parameter to retrieve his or her desired result. We utilize the same approach in SecLaaS. For example, in the network log scenario, a sample GET request for our scheme can be,

```
GET /log?fromIP=10.13.155.4&date=2015-03-04
&start=10:45:00&end=12:45:00&tz=UTC
```

This GET operation requests the logs where *" from ip"* is 10.13.155.4 and the time is between 10:45:00 to 12:45:00 UTC on March 4th, 2015. From this GET request, SecLaaS first finds the logs that match the search criteria. For the *Bloom-Tree*, we also need to provide all the nodes of the tree that match the search criteria.

An example of a response message while using the *Bloom-Tree* is presented in Figure 8. The response message produces the ELE and LC of each log entry separately, which are encapsulated in $DBLELIST$. The $PROOF$ tag represents one node of the *Bloom-Tree*, which contains the bit array, identity of the parent node, and a boolean value to indicate whether a node is a leaf or not. For the Bloom filter and the RSA accumulator, the $PROOFLIST$ will be empty. After receiving the above response from SecLaaS API, the caller will acquire the Proof of Past Logs (PPL) of that day, which was made publicly available by the CSP earlier. Any client-side application should be able to parse this response message and run the verification processes discussed earlier.

Security of REST web services is mostly ensured by using HTTPS protocol (HTTP over transmission security protocol SSL / TLS). All REST API calls must take place over HTTPS with a certificate signed by a trusted CA. The client application first verifies the certificate of CSPs to ensure that it is indeed communicating with a valid CSP. To authenticate a valid caller, we use an API key. API callers (law enforcement agencies and CSPs) share a secret value and the signature of the API caller on that secret value is treated as the API key. This signing should occur before encoding the URL query string. To authenticate an API caller, CSPs verify the signature on the shared secret. It provides a two-way authentication: compromising only the secret value or only the private key of the LEA cannot break this security. To spoof a valid caller, an attacker needs to compromise both.

```
<? xml version ="1. 0" encoding="UTF-8" ?>
<SECLAASRESPONSE>
    <DBLELIST>
        <DBLE>
            <ELE>Log entry 1</ ELE>
            <LC>Log chain of 1st entry </ LC>
        </DBLE>
        . . .
        <DBLE>
            <ELE>Log entry N</ ELE>
            <LC>Log chain of Nth entry </ LC>
        </DBLE>
    </ DBLELIST>
    <PROOFLIST>
        <PROOF>
            <BITS>Bit array</ BITS>
            <PARENT>Parent Node ID</ PARENT>
            <ISLEAF>1/ 0</ ISLEAF>
        </ PROOF>
        . . .
        <PROOF>
            <BITS>Bit array </ BITS>
            <PARENT>Parent Node ID</ PARENT>
            <ISLEAF>1/ 0</ ISLEAF>
        </ PROOF>
    </ PROOFLIST>
</ SECLAASRESPONSE>
```

Fig. 8: Response Message of SecLaaS

6 Security Analysis

In our collusion model, there are three entities involved – CSP, user, and investigator. All of them can be malicious individually or can collude with each other. CSPs have full control over the stored logs and the proofs of logs. Hence, they can always tamper with the logs. After acquiring logs through API, an investigator can also alter the logs before presenting it to the court. Therefore, we propose a tamper evident scheme in this paper. Any violation of the integrity properties, as mentioned in Section 4, can be detected during the verification process. By using our proposed scheme, we can also preserve the privacy of cloud users from external or insider attackers. In this section, we discuss how our proposed system can ensure all the security properties that are required to protect collusion between CSP, user, and investigator.

Table 2 presents all the possible combinations of the collusion, possible attacks for each collusion, and required security properties to defend against that collusion. We denote an honest CSP as C, a dishonest CSP as $\bar{C}$, an honest user as U, a dishonest user as $\bar{U}$, an honest investigator as I, and a dishonest investigator as $\bar{I}$.

I1, I2, I4, I5. The integrity properties I1, I2, I4, and I5 prevent removal and reordering of log entries by a dishonest CSP and investigator. At the verification stage, our system can detect any such removal and reordering of log entries.

Let us assume that there are three log entries DBLE0, DBLE1, and DBLE2 and their proof has already been published. Now, if the CSP removes DBLE1 and provides only DBLE0 and DBLE2 to the investigator, then this removal can be easily detected at the sequence verification stage. In this case, the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2482484, IEEE Transactions on Dependable and Secure Computing

Is Honest?			Notation	Attack	Required Security Properties
CSP	User	Investigator			
✓	✓	✓	C U I	No attack	None
×	✓	✓	$\bar{C}$ U I	Reveal user activity from logs	C2
✓	×	✓	C $\bar{U}$ I	Recover cloud users' log from published proof	C1
✓	✓	×	C U $\bar{I}$	Remove, reorder, and plant fake logs	I4, I5, I6
✓	×	×	C $\bar{U}\bar{I}$	Remove, reorder, plant fake logs, and recover other cloud users' log	I4, I5, I6, C1
×	✓	×	$\bar{C}$ U $\bar{I}$	Remove, reorder, plant fake logs, repudiate published PPL, and reveal user activity	I1, I2, I3, I4, I5, I6, I7, C2
×	×	✓	$\bar{C}\bar{U}$ I	Remove, reorder, plant fake logs, repudiate published PPL, recover cloud users' logs and activity	I1, I2, I3, I7, C1, C2
×	×	×	$\bar{C}\bar{U}\bar{I}$	Remove, reorder, plant fake logs, repudiate published PPL, recover cloud users' logs and activity	I1, I2, I3, I4, I5, I6, I7, C1, C2

TABLE 2: Collusion model, possible attacks and required security properties

hash of LC0 and ELE2 will not match with LC2, because the original LC2 was calculated by hashing LC1 and ELE2.

Since the published PPL is generated using the accumulator entry of the last log of an epoch AE_D, removal of the last log DBLE2 can also be detected. After removing DBLE2, DBLE1 will be the current last log. However, the AE1 of DBLE1 will not match with the AE_D of PPL, which was actually AE2 of DBLE2.

An auditor can detect the re-ordering of logs using the verification procedure. For example, while providing the logs to an auditor, if the CSP or investigator provides logs in the order of DBLE0, DBLE2, DBLE1, then using the same technique, the auditor can identify that DBLE1 does not come after DBLE2 in actual generation order.

The CSP can further try to change the DBLE2 by replacing the original LC2 with a new Log Chain value. Hence, reordering or removing of logs will not be detected in the sequence verification process. However, an attempt of changing the DBLE2 will be detected during the individual log entry verification phase which is ensured by I3 and I6 properties.

A malicious investigator can also claim the unavailability of logs for a certain period of time. However, the existence of a published PPL for that particular epoch indicates the presence of logs, which can prevent an investigator to establish the claim of unavailability of logs.

I3, I6. The integrity properties I3 and I6 prevent producing fake logs by CSPs and investigators. A colluding CSP can plant false log information while providing the logs to investigators. A dishonest investigator can also try to frame an honest user by presenting fake logs to the court. However, if the CSP or investigator provides fake logs after publishing the proof, our system can detect these fake logs. If $DBLE_f$ is a fake log entry, then using any type accumulator scheme we can detect that the fake log does not exist in the published AE_D of the Proof of Past Log *PPL*, and the auditor can reject that incorrect log. However, the Bloom filter is a probabilistic data structure and suffers from false positive (FP) rate. Hence, there are still some chances of planting false log information by CSPs or investigators if the FP is not small enough to be negligible.

I7. This integrity property ensures non repudiation of Proof of Past Log by CSPs. After publishing the PPL, CSPs cannot repudiate the published proof, as the Accumulator Entry AE_D is signed by their private key. Nobody other than a CSP can use the private key to sign the AE_D. Hence if a PPL passed the signature verification step using the public key of the CSP P_{KC}, the CSP cannot deny the published value.

C1, C2. The confidentiality properties C1 and C2 ensure cloud users' privacy from an external attacker or a malicious cloud employee. In all of the accumulator schemes, an accumulator entry for a log entry is generated by hashing the log. As the hash function provides the one-way property, the proposed scheme ensures the C1 property, i.e., from the proof of logs, adversaries cannot recover any log. While storing the log data in persistent storage, we propose to encrypt some crucial information e.g., user id, destination IP, etc. by using a common public key of the law enforcement agencies. Hence, a malicious cloud employee cannot retrieve crucial confidential information of users from logs stored in persistent storage. For example, a malicious cloud employee cannot identify the visiting IPs of a particular user. In this way, our scheme can ensure the C2 property.

7 Implementation and Evaluation

In this section, we present the implementation of SecLaaS on OpenStack and performance analysis of the scheme using different types of accumulators.

7.1 Implementation

System Setup: To build a private cloud, we used Openstack[1] – an open source cloud computing software. We built our prototype for network logs and used Snort as a source of network logs. We created the virtual environments with VirtualBox[3] (a free virtualization software) running on a single Ubuntu machine. Figure 9 illustrates the system setup and below is the description of the system:

• Host machine's hardware configuration: Intel Core I7 quad core CPU, 16 GB ram, and 750 GB hard drive. Ubuntu 12.04 LTS 64-bit is used as Host Operating System.

• VirtualBox 4.1.22 r80657 for Ubuntu 12.04 LTS.

• Openstack (Essex) installation in VirtualBox; for simplicity, the system had one node controller. Configuration of the virtual

1. http://www.openstack.org
3. https://www.virtualbox.org

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2482484, IEEE Transactions on Dependable and Secure Computing

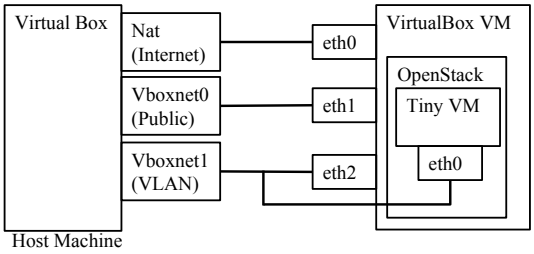


Fig. 9: Prototype Environment Configuration [47]

node controller: Intel 2.5Ghz Dual Core cpu, 8 GB ram, and 20 GB hard drive. Ubuntu 12.04 64-bit Server edition is used as the Operating system for Openstack setup. We hosted maximum ten *m1.tiny* VMs on the node controller for performance evaluation.

• In the virtualized environment, the Cloud Controller required following network adapter configuration in VirtualBox to work properly:

- Adapter 1: Attached to NAT- eth0 of the Cloud Controller is connected here.
- Adapter 2: Host-only network for Public interface- connected with eth1 (IP was set to 172.16.0.254, mask 255. 255.0.0, DHCP disabled)
- Adapter 3: Host-only network for Private (VLAN) interface connected with eth2 (IP to 11.0.0.1, mask 255. 0.0.0, DHCP disabled)

• We used RSA (2048 bit) for signature generation, and SHA-2(SHA-256) hash function for hashing.

• Snort was set up in the node controller to track the network activity of the virtual machines. Here is a sample Snort log:

```
''11/19-13:43:43.222391 11.1.0.5:51215 −>fg
74.125.130.106:80  TCP TTL:64 TOS:0x0 ID:22101
IpLen:20 DgmLen:40 DF  ***A***F Seq: 0x3EA405D9 Ack:
0x89DE7D  Win: 0x7210 TcpLen: 20''
```

This log tells that the virtual machine with private IP 11.1.0.5 performed a http request to machine 74.125.130.160. Logs generated by Snort were first streamed to the Parser module, which creates a Log Entry (LE) and sends to a message queue (MQ). The Logger module always listens to this MQ and whenever there is a new LE in the MQ, the logger starts executing steps (d) to (i) of Figure 3. To publish PPL periodically, steps (j) to (l) are encapsulated as a Cron job. Specific time and frequency of publishing PPL can be controlled by the Cron job parameters.

By reverse engineering Openstack's "Nova" mysql database, it is also possible to find out the static private IP and user information from a public IP. We used the references among FloatingIps, FixedIps, and Instances tables to resolve the user id for a particular log entry. Figure 10 shows the relation between these three tables.

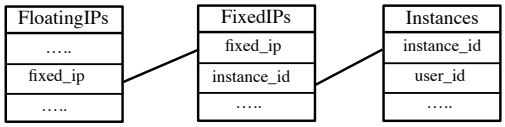


Fig. 10: Resolving User ID from Public IP

Exposing APIs: To implement the RESTful APIs, we used Glassfish as the web server and used JAX-RS for creating the REST APIs. Figure 11 shows the process flow of handling an API call. The web server listens on port 8443 and whenever it receives a request from an API caller, it invokes a java servlet. The servlet then communicates with the verification and authentication module. The verification and authentication module is responsible for checking the validity of the API request and the authentication information provided by the caller. If the call is valid and the caller is authenticated, then the servlet communicate with the persistent storage to retrieve the requested data. Finally, the servlet prepares an XML response and returns to the calling entity. The response can be JSON or XML, but we used XML as it is a well known format and any client application should be able to handle it.

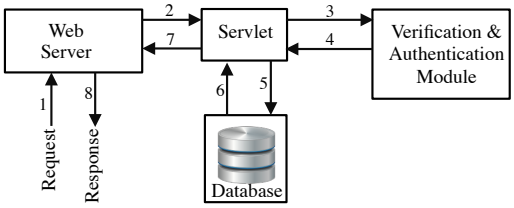


Fig. 11: Process Flow for Handling API Calls

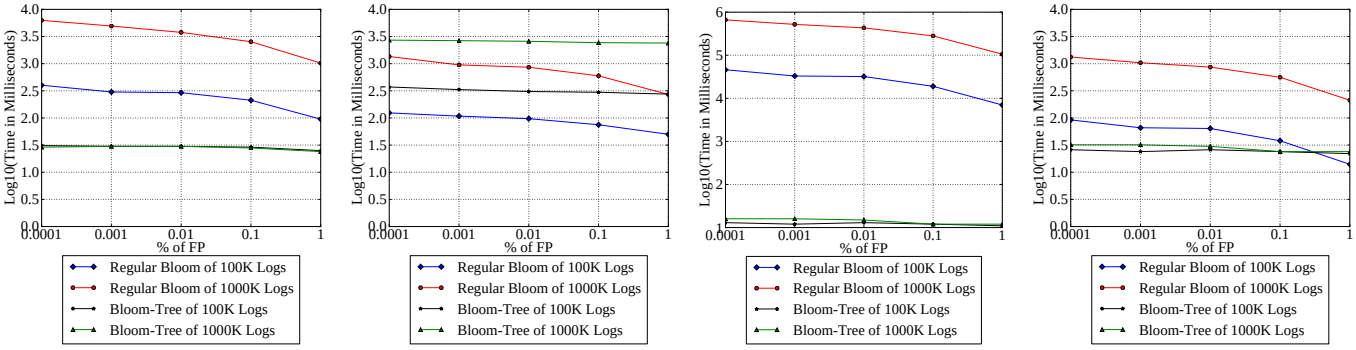
7.2 Comparison Among Different Accumulators

In [26], we have shown that the Bloom filter outperforms the RSA accumulator in terms of log insertion, PPL generation, integrity verification, and space requirement. In this section, we compare the performance between the Bloom filter and the Bloom-Tree for false positive probability of 0.0001%, 0.001%, 0.01%, 0.1%, and 1%. We used $m = 1000$ and $b = 10$ for the Bloom-Tree. For each of the experiments of Figure 12, we stored maximum 1 million logs generated by Snort.

Log Insertion: Figure 12a shows the performance analysis of log insertion, which includes the time required to complete the steps from (d) to (i) of Figure 3. The graph indicates that for both of the accumulators, the log insertion time increases when we reduce the percentage of FP and increase the total number of logs in a day. However, the log insertion time for the Bloom-Tree is nearly constant with the increase in number of logs. The reason is for the Bloom-Tree, we are always using a bit array that can hold membership information of m items. We notice that the Bloom-Tree provides significantly better performance than the Bloom filter and the difference is greater for lower percentage of FP. The reason is that for the Bloom filter, the size of bit array and number of hash functions increase significantly when lowering the percentage of FP for a large number of items. As m is very small compared to total number of logs, the increment in the size of bit array and number of hash function is reduced using the Bloom-Tree.

PPL Generation: Figure 12b illustrates the performance analysis of generating the Proof of Past Log of a day for different configurations of the Bloom filter and the Bloom-Tree. We notice that for both of the accumulator schemes, the PPL generation time increases while increasing the total number of logs and decreasing the FP probability. Figure 12b illustrates that the Bloom filter outperforms the Bloom-Tree

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2482484, IEEE Transactions on Dependable and Secure Computing



(a) Log Insertion (b) PPL Generation (c) Best Case Integrity Verification (d) Worst Case Integrity Verification

Fig. 12: Comparison between the Bloom filter and the Bloom-Tree

when using same configuration. The cost of building the Bloom-Tree is the reason behind its lower performance, since this is an additional work and is not required for the Bloom filter.

Since the number of nodes and depth of the Bloom-Tree increases with the total number of logs, the cost of building the root (AE_D) increases and so does the PPL generation for greater number of logs. The size of the bit array increases while increasing the number of logs and decreasing the percentage of FP. Since the cost of hashing increases with the size of message, the PPL generation time for the Bloom filter also increases with the size of the bit array.

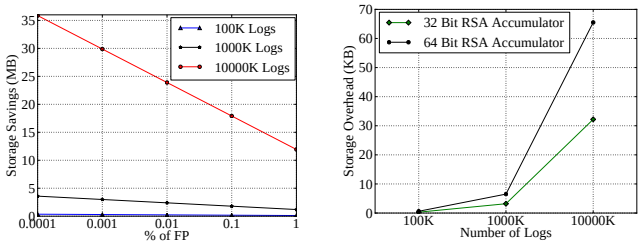
Integrity Verification: For the Bloom-Tree, the auditor may need to collect some extra logs. Hence, to compare the performance for integrity verification, we considered two cases: best case and worst case.

In the *best case* scenario of the Bloom-Tree, the auditor collects the m number of logs, where the start index is SI and SI mod $m = 0$. We compare the performance of verifying the integrity of such m number of logs using the Bloom-Tree and the Bloom filter for different FP probability and total number of logs. Figure 12c depicts that the performance of the Bloom-Tree is significantly better than the Bloom filter in the best case scenario.

In the *worst case* scenario of the Bloom-Tree, the auditor collects $2 * (m - 1)$ additional logs and needs to verify the integrity of those logs in order to verify the integrity of only 2 log entries. To compare the performance in this situation, we measure the integrity verification time for $2m$ logs using the Bloom-Tree and integrity verification time for 2 logs using the Bloom-filter. From Figure 12d, we notice that even in the worst case scenario, the Bloom-Tree performs better when the percentage of FP is very low and number of total logs is high.

The reason for a better performance of the Bloom-Tree is that the size of bit array is very small compared to the Bloom filter and the cost of checking the bits increases linearly with the size of the bit array. Moreover, we need to generate a smaller number of bit positions for the Bloom-Tree, which requires less number of hashing.

Storage Overhead: The storage overhead for Bloom filter only depends on the storage requirement of the proofs. In the Bloom filter, the size of the bit array depends on the expected number of elements and FP probability. To ensure very low FP probability for high number of expected elements, the size of bit array needs to very high. For example, to ensure 1% FP for 100,000 elements, the size of bit array needs to be 958,506 and to ensure 0.0001% FP for the same number of elements, the size of bit array should be 2,875,518.



(a) Storage savings using the Bloom-Tree compared to regular the Bloom filter (b) Overhead of using tree structure for the RSA Accumulator

Fig. 13: Analysis of Space Requirement

For the Bloom-Tree, we store the intermediate nodes of the tree, which hold the membership information of a child accumulator. The size of the bit array for the intermediate nodes depends on the branching factor b. If $b = 10$, the intermediate nodes only need to ensure the desired FP probability for 10 elements. For $m = 1000$ and $b = 10$, the Bloom-Tree scheme needs to store only 11 nodes for 100,000 elements and the size of these nodes depends on the FP probability. To ensure 1% FP for 100,000 elements using the Bloom-Tree, we require 3960 bits including 32 bytes for storing parent identity and 1 byte for a boolean attribute. This is 119.31 KBytes less compared to the regular Bloom filter scheme. Figure 13a presents an analysis of storage savings by the Bloom-Tree compared to the Bloom filter for different configurations of FP and total number of logs. Figure 13a depicts that storage savings by the Bloom-Tree increases with the increase in number of logs and decrease in percentage of FP. We notice that if the total number of logs for a user in a day is 10,000,000 and we want to ensure 0.0001% FP, the Bloom-Tree can save approximately 35 MBytes of storage. Considering SecLaaS is a continuous process, the storage saving can reach several terabytes over the years.

Performance Overhead. To identify the performance degradation of NC for running SecLaaS, we measured the system overhead from the CPU performance information of SysBench

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2482484, IEEE Transactions on Dependable and Secure Computing

[48]. We first measured the CPU performance of NC while it hosted different numbers of VMs and each VM was executing the ping command to an external machine. Later, we kept running Snort, Parser and Logger module on NC and measured the CPU performance to identify the system overhead. Figure 14 represents the overhead for different accumulator schemes. The lowest and highest overhead found was 1.67% and 3.28% and there was no single accumulator scheme which was best or worst in terms of performance overhead. Moreover, the overhead did not increase with the number of VMs; rather we noticed a downward trend of overhead with the increase in number of VMs. The reason behind this behavior is that the overhead of running VMs is higher than the overhead of running SecLaaS. Therefore, when the number of VMs increases, the overall overhead for running SecLaaS decreases.

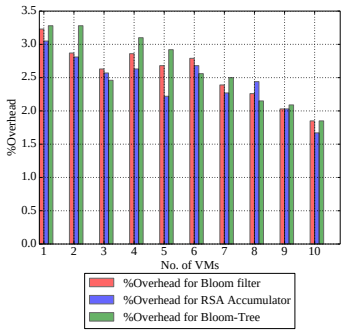


Fig. 14: Performance Overhead for different accumulator schemes

8 Discussion

8.1 Selection of the Accumulator

Our previous work [26] reveals that the Bloom filter outperforms the RSA accumulator in all aspects, excepts the false positive (FP) rate. In this paper, we propose the Bloom-Tree that can ensure negligible FP while providing better performance than the regular Bloom filter approach in terms of log insertion, integrity verification, and space requirement. The Bloom-Tree performs worse than the Bloom filter in PPL generation and in the worst case scenario of integrity verification when the FP is high. However, the PPL will be generated after a certain epoch and that can be done by a background process without affecting the regular log insertion process. Moreover, as our goal is to provide better security by ensuring low FP rate, the worst case scenario for high FP rate should not be considered while selecting a better accumulator.

We considered to build a tree structure for the RSA accumulator similar to the Bloom filter but our theoretical analysis suggest that it will not be better than the regular RSA accumulator. One of the reasons is that the performance (time and space) of the RSA accumulator does not vary with the total amount of expected log entries. It only varies with size of the security parameters P, Q, and X. Figure 13b, shows the overhead of storage requirement if we use tree-based structure similar to the Bloom-Tree for the RSA accumulator. In a tree-based accumulator, number of intermediate nodes increases with the number of logs. For the RSA accumulator, each intermediate node requires the space of the regular RSA accumulator including the identity. Therefore, the storage requirement increases with the number of logs. Introducing the tree-based approach for the RSA accumulator will also introduce higher time for PPL generation and log verification because of the additional intermediate nodes. Considering all of the criteria, we posit that the Bloom-Tree can be the best choice to securely store large amount of logs.

8.2 Regulatory Compliant Cloud

Auditability is a vital issue to make the cloud compliant with the regulatory acts, e.g., SOX [19] or HIPAA [20]. The SOX act mandates that the financial information must reside in an auditable storage, which the CSPs cannot provide currently. Business organizations cannot move their financial information to the cloud infrastructure, as it does not comply with the act. As clouds do not comply with HIPAA's forensic investigation requirements yet, hospitals cannot move their patients' medical information to a cloud storage. Preserving the logs and the proofs of the logs securely will definitely increase the auditability of the cloud environment. Using our scheme, it is possible to store and provide any types of logs from which we can get all the activities of cloud users. Business and healthcare organizations are the two most data consuming sectors; cloud computing cannot achieve the ultimate success without including these two sectors. These sectors are spending extensively to make their own regulatory-compliant infrastructure. A regulatory-compliant cloud can save this huge investment. Hence, we need to solve the audit compliance issue to bring more customers in the cloud world. Implementing SecLaaS can help to make the cloud more compliant with such regulations, leading to widespread adoption of clouds by major businesses and healthcare organizations.

9 Conclusion

To execute a successful forensics investigation in clouds, the necessity of logs from different sources, e.g., network, process, databases, is undeniable. Since today's clouds offer very little control compared to traditional computing systems, investigators need to depend on CSPs to collect logs from different sources. Unfortunately, there is no way to verify whether the CSP is providing correct logs to the investigators, or the investigators are presenting valid logs to the court. Moreover, while providing the logs to the investigators, CSPs need to preserve the privacy of the cloud users. Besides protecting the cloud, we should also focus on these issues. Unfortunately, there has been no solution that can make the logs available to investigators, and at the same time, can preserve the confidentiality and integrity of the logs. In this paper, we addressed this problem, which can have significant real-life implications in law enforcement investigating cyber-crime and terrorism. We proposed *SecLaaS*, which can be a solution to store and provide logs for forensics purpose securely. This scheme will allow CSPs to store logs while preserving the confidentiality of cloud users. Additionally, an auditor can check the integrity of the logs using the Proof of Past Log *PPL* and the Log Chain *LC*. We ran our proposed solution on OpenStack and found it practically feasible to integrate with the cloud infrastructure. In future, we will implement SecLaaS as a module of OpenStack.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2482484, IEEE Transactions on Dependable and Secure Computing

REFERENCES

[1] B. Deeter and K. Shen, "BVP Cloud Computing Index Crosses the $100 Billion Market Milestone," http://goo.gl/mEuEi4, 2013, [Accessed March 20, 2015].

[2] Market Research Media, "Global cloud computing market forecast 2015-2020," http://goo.gl/AR3FBD, [Accessed March 20, 2015].

[3] INPUT, "Evolution of the cloud: The future of cloud computing in government," http://goo.gl/Ksuc5i, 2009, [Accessed March 20, 2015].

[4] Borja,Florence de, "IDC Report: IT Cloud Services Market to Reach $43.2 Billion by 2016," http://goo.gl/Csnpy, 2012, [Accessed March 20, 2015].

[5] M. Balduzzi, J. Zaddach, D. Balzarotti, E. Kirda, and S. Loureiro, "A security analysis of amazon's elastic compute cloud service," in *Symposium on Applied Computing*. ACM, 2012, pp. 1427–1434.

[6] S. Subashini and V. Kavitha, "A survey on security issues in service delivery models of cloud computing," *Journal of Network and Computer Applications*, vol. 34, no. 1, pp. 1–11, 2011.

[7] D. Zissis and D. Lekkas, "Addressing cloud computing security issues," *Future Generation Computer Systems*, vol. 28, no. 3, pp. 583–592, 2012.

[8] D. Birk and C. Wegener, "Technical issues of forensic investigations in cloud computing environments," in *SADFE*. IEEE, 2011, pp. 1–10.

[9] G. Grispos, T. Storer, and W. B. Glisson, "Calm before the storm: The challenges of cloud computing in digital forensics," *International Journal of Digital Crime and Forensics*, vol. 4, no. 2, pp. 28–48, 2013.

[10] D. Reilly, C. Wren, and T. Berry, "Cloud computing: Forensic challenges for law enforcement," in *ICITST*. IEEE, 2010, pp. 1–7.

[11] S. Zawoad and R. Hasan, "Digital forensics in the cloud," *The Journal of Defense Software Engineering*, vol. 26, no. 5, pp. 17–20, 2013.

[12] ——, "Towards building proofs of past data possession in cloud forensics," *ASE Science Journal*, vol. 1, no. 4, pp. 195–207, 2012.

[13] The Register, "Amazon cloud hosts nasty banking trojan," http://goo.gl/xGNkNO, 2011, [Accessed March 20, 2015].

[14] www.bbc.com, "Lostprophets' Ian Watkins: 'Tech savvy' web haul," http://goo.gl/C8FVnC, December 2013, [Accessed March 20, 2015].

[15] Dist. Court, SD Texas, "Quantlab technologies ltd. v. godlevsky," Civil Action No. 4: 09-cv-4039, 2014.

[16] Infosecurity-magazine, "Ddos-ers launch attacks from amazon ec2," http://goo.gl/vrXrHE, July 2014, [Accessed March 20, 2015].

[17] M. Bellare and B. Yee, "Forward-security in private-key cryptography," *Topics in Cryptology, CT-RSA 2003*, pp. 1–18, 2003.

[18] ——, "Forward integrity for secure audit logs," Technical report, Computer Science and Engineering Department, University of California at San Diego, Tech. Rep., 1997.

[19] Congress of the United States, "Sarbanes-Oxley Act," http://goo.gl/YHwujG, 2002, [Accessed March 20, 2015].

[20] www.hhs.gov, "Health Information Privacy," http://goo.gl/NxgkMi, [Accessed March 20, 2015].

[21] J. Dykstra and A. T. Sherman, "Design and implementation of frost: Digital forensic tools for the OpenStack cloud computing platform," *Digital Investigation*, vol. 10, no. 0, pp. S87–S95, 2013.

[22] J. E. Holt, "Logcrypt: forward security and public verification for secure audit logs," in *Australasian workshops on Grid computing and e-research-Volume 54*. Australian Computer Society, Inc., 2006, pp. 203–211.

[23] B. Schneier and J. Kelsey, "Secure audit logs to support computer forensics," *ACM TISSEC*, vol. 2, no. 2, pp. 159–176, 1999.

[24] D. Ma and G. Tsudik, "A new approach to secure logging," *ACM Transaction on Storage (TOS)*, vol. 5, no. 1, pp. 2:1–2:21, Mar. 2009.

[25] R. Accorsi, "On the relationship of privacy and secure remote logging in dynamic systems," in *Security and Privacy in Dynamic Environments*. Springer US, 2006, vol. 201, pp. 329–339.

[26] S. Zawoad, A. K. Dutta, and R. Hasan, "SecLaaS: Secure logging-as-a-service for cloud forensics," in *ASIACCS*. ACM, 2013, pp. 219–230.

[27] K. Kent, S. Chevalier, T. Grance, and H. Dang, "Guide to integrating forensic techniques into incident response," *NIST Special Publication*, pp. 800–86, 2006.

[28] J. Wiles, K. Cardwell, and A. Reyes, *The best damn cybercrime and digital forensics book period*. Syngress Media Inc, 2007.

[29] P. Mell and T. Grance, "Nist cloud computing forensic science challenges," *Draft NISTIR 8006*, June 2014.

[30] K. Ruan, J. Carthy, T. Kechadi, and M. Crosbie, "Cloud forensics," in *Advances in digital forensics VII*. Springer, 2011, pp. 35–46.

[31] S. Zimmerman and D. Glavach, "Cyber forensics in the cloud," *IA Newsletter*, vol. 14, no. 1, pp. 4–7, 2011.

[32] Z. Zafarullah, F. Anwar, and Z. Anwar, "Digital forensics for eucalyptus," in *FIT*. IEEE, 2011, pp. 110–116.

[33] A. Patrascu and V.-V. Patriciu, "Logging system for cloud computing forensic environments," *Journal of Control Engineering and Applied Informatics*, vol. 16, no. 1, pp. 80–88, 2014.

[34] R. Marty, "Cloud application logging for forensics," in *Symposium on Applied Computing*. ACM, 2011, pp. 178–184.

[35] A. Yavuz and P. Ning, "BAF: An efficient publicly verifiable secure audit logging scheme for distributed systems," in *ACSAC*, 2009, pp. 219–228.

[36] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[37] S. Thorpe and I. Ray, "Detecting temporal inconsistency in virtual machine activity timelines." *Journal of Information Assurance & Security*, vol. 7, no. 1, pp. 24–31, 2012.

[38] I. Ray, K. Belyaev, M. Strizhov, D. Mulamba, and M. Rajaram, "Secure logging as a service—delegating log management to the cloud," *IEEE systems journal*, vol. 7, pp. 323–334, 2013.

[39] A. Kumar, J. Xu, and J. Wang, "Space-code bloom filter for efficient per-flow traffic measurement," *Journal on Selected Areas in Communications*, vol. 24, no. 12, pp. 2327–2339, 2006.

[40] C. Papamanthou, R. Tamassia, and N. Triandopoulos, "Authenticated hash tables," in *CCS*. ACM, 2008, pp. 437–448.

[41] G. Koloniari and E. Pitoura, "Bloom-based filters for hierarchical data," in *WDAS*, 2003, pp. 13–31.

[42] K. Shanmugasundaram, H. Brönnimann, and N. Memon, "Payload attribution via hierarchical bloom filters," in *CCS*. ACM, 2004, pp. 31–41.

[43] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *CCS*, 2009, pp. 199–212.

[44] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[45] J. Benaloh and M. De Mare, "One-way accumulators: A decentralized alternative to digital signatures," in *EUROCRYPT*, 1994, pp. 274–285.

[46] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures Chapter 5: Representational State Transfer (REST)," http://goo.gl/NAXd, 2000, [Accessed March 20, 2015].

[47] Tikal, "Experimenting with OpenStack Essex on Ubuntu 12.04 LTS under VirtualBox," http://goo.gl/CZ54VE, 2012, [Accessed March 20, 2015].

[48] SysBench, "Sysbench: a system performance benchmark," http://sysbench.sourceforge.net/, [Accessed March 20, 2014].

Shams Zawoad is a graduate research assistant in SECuRE and Trustworthy Computing Lab (SECRETLab) and a Ph.D. student at the University of Alabama at Birmingham (UAB). His research interest is in cloud forensics, anti-phishing, and secure provenance. He received his B.Sc. in Computer Science and Engineering from Bangladesh University of Engineering and Technology (BUET) in January 2008.

Amit Kumar Dutta is working as a Member of Technical Staff at VMware. He was a member of SECuRE and Trustworthy Computing Lab and a MS student at the University of Alabama at Birmingham (UAB). His research interest is in cloud security and in digital data waste. He received his B.Sc. in Computer Science and Engineering from Bangladesh University of Engineering and Technology in October 2009.

Dr. Ragib Hasan is a tenure-track Assistant Professor at the Department of Computer and Information Sciences at the University of Alabama at Birmingham. Prior to joining UAB, He received his Ph.D. and M.S. in Computer Science from the University of Illinois at Urbana Champaign in October, 2009, and December, 2005, respectively, and was an NSF/CRA Computing Innovation Fellow post-doc at the Department of Computer Science, Johns Hopkins University. Hasan has received multiple awards in his career, including the 2014 NSF CAREER Award, 2013 Google RISE Award, and 2009 NSF Computing Innovation Fellowship.