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/TSUSC.2018.2833502, IEEE Transactions on Sustainable Computing

CLASS: Cloud Log Assuring Soundness and Secrecy Scheme for Cloud Forensics

M A Manazir Ahsan, Ainuddin Wahid Bin Abdul Wahab, Mohd Yamani Idna Bin Idris, Suleman Khan, Eric Bachura, Kim-Kwang Raymond Choo, *Senior Member*, IEEE

Abstract—User activity logs can be a valuable source of information in cloud forensic investigations; hence, ensuring the reliability and integrity of such logs is crucial. Most existing solutions for secure logging are designed for conventional systems rather than the complexity of a cloud environment. In this paper, we propose the Cloud Log Assuring Soundness and Secrecy (CLASS) process as an alternative scheme for the securing of logs in a cloud environment. In CLASS, logs are encrypted using the individual user's public key so that only the user is able to decrypt the content. In order to prevent unauthorized modification of the log, we generate proof of past log (PPL) using Rabin's fingerprint and Bloom filter. Such an approach reduces verification time significantly. Findings from our experiments deploying CLASS in OpenStack demonstrate the utility of CLASS in a real-world context.

Index Terms— Cloud forensics, Cloud log, Cloud log assuring soundness and secrecy, Cloud security, Proof of past log, Sustainable computing

1 Introduction

CLOUD storage, security and privacy are fairly established research areas [1-7], which is not surprising considering the widespread adoption of cloud services and the potential for criminal exploitation (e.g. compromising cloud accounts and servers for the stealing of sensitive data). Interestingly though, cloud forensics [8-10] is a relatively less understood topic. In the event that a cloud service, cloud server, or client device has been compromised or involved in malicious cyber activity (e.g. used to host illegal contents such as radicalization materials, or conduct distributed denial of service (DDoS) attacks) [11, 12], investigators need to be able to conduct forensic analysis in order to "answer the six key questions of an incident – what, why, how, who, when, and where" [13].

Due to the inherent nature of cloud technologies, conventional digital forensic procedures and tools need to be updated to retain the same usefulness and applicability in a cloud environment [14]. Unlike a conventional client device, cloud virtual machines (VMs) can be supported by hardware that might be located remotely and thus would not be physically accessible (e.g. out of the jurisdictional territory) to an investigator.

- *M.A.M. Ahsan, A.W.A Wahab, and M.Y. Idris are with Faculty of Computer Science and Information Technology, University of Malaya, Malaysia. E-mail: manazir.ahsan@siswa.um.edu.my, (ainuddin, yamani@um.edu.my)*
- *S.Khan is with the School of Information Technology, Monash University, 47500 Bandar Sunway, Malaysia. E-mail: Suleman.khan@monash.edu.*
- *E. Bachura and K.K.R. Choo are with the Department of Information Systems and Cyber Security, The University of Texas at San Antonio, San Antonio, TX 78249-0631, USA E-mail: (eric.bachura@utsa.edu, raymond.choo@fulbrightmail.org)*

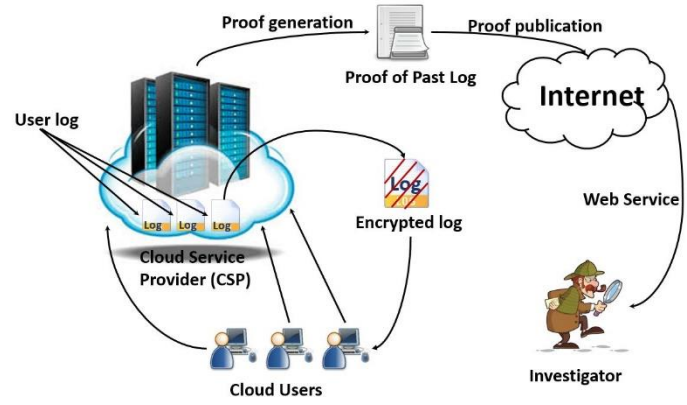


Fig. 1. Overview of CLASS scheme process

In addition, VMs can be distributed across multiple physical devices in a clustered environment or they can exist within a pool of VMs on the same physical components. Therefore, seizing the machine for forensic analysis is not viable in most investigations. Furthermore, data residing in a VM may be volatile and could be lost once the power is off or the VM terminates. Hence, the cloud service provider (CSP) plays a crucial role in the collection of evidential data (e.g. cloud user's activity log from the log). For example, the CSP writes the activity log (cloud log) for each user. Thus, preventing modification of the logs, maintaining a proper chain of custody and ensuring data privacy is crucial [15]. This research considers "activity log data" as any recorded computer event that corresponds to a specific user. Such data must be maintained confidentially to preserver user privacy and to facilitate potential investigative activities.

In 2016, Zawoad et al. proposed a secure logging service called "SecLaaS" [16] that is designed to collect data from one or more log sources, parse the data and then store the parsed data in persistent storage in order to mitigate the risk associated with data volatility. Prior to the storing of

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/TSUSC.2018.2833502, IEEE Transactions on Sustainable Computing

data, it encrypts the log and generates a log chain to achieve confidentiality and integrity respectively. SecLaaS encrypts the log(s) using the investigating agency's public key and stores the encrypted log(s) in a cloud server. This ensures privacy and confidentiality of the cloud user, unless the particular user is subject to an investigation (e.g. via a court order). To facilitate log integrity, SecLaaS generates proof of past log (PPL) with the log chain and publishes it publicly after each predefined epoch. A trust model was also suggested that stores the PPL in other clouds to minimize the risk of a malicious cloud entity altering the log. However, in SecLaaS, it is difficult to ensure or verify that the CSP is writing the correct information to the log, or that any information pertinent to the investigation is not omitted or modified. Specifically, SecLaaS does not provide the user the ability to verify the accuracy of the log (since the log is encrypted with the agency's public key). In other words, SecLaaS has limitations in addressing accountability and transparency enforced, especially from the perspective of the user.

Extending SecLaaS, we propose a secure cloud logging scheme, Cloud Log Assuring Soundness and Secrecy (CLASS), designed to ensure CSP accountability (i.e. writing the correct information to the log) and preserve the user's privacy – i.e. our contribution in this paper. Specifically, we include the capability for the user to verify the accuracy of their log. To do this, the log will be encrypted using the user's public key (rather than the agency's public key). To avoid introducing unnecessary delays to the forensic investigation, during user registration with the cloud service, both the CSP and the user will collectively choose a public/private key pair referred to as content concealing key (CC-key) for the user. The corresponding (content concealing) private key will be shared with other CSPs using Shamir's [17] or Blakley's [18] secret sharing schemes. This would allow the private key to be regenerated whenever necessary. We also demonstrate how we can leverage Rabin's fingerprint [19] and bloom filter in PPL generation to establish log veracity. We then implement CLASS in OpenStack and evaluate its performance.

We will review related work in the next section, before discussing the threat model and secure logging system requirements in Section 3. The SecLaaS scheme is discussed and reviewed in Section 4. Our proposed scheme is presented in Section 5. An evaluation of how the proposed scheme meets the security properties a complexity evaluation is covered in Section 6. Performance evaluation is covered in Section 7, to include a discussion on implementation and setup. We provide concluding remarks and discuss future extensions in Section 8.

2 Related Work

Reliable cloud logs play a crucial role in forensic investigations. Log acquisition, maintaining confidentiality, integrity and forward secrecy, validity verification and accessibility by investigators (or another authorized party), are some of the many different dimensions a forensic investigator and researcher should pay attention to. Anwar et al. [20] attempted to address some of these challenges by identifying the possibility of Syslog or snort log to assist in the detection of cloud attacks. The authors conducted cloud forensic investigations based on the logs generated by Eucalyptus, an open-source cloud computing software. Specifically, they generated their own dataset by simulating a DDoS attack on Eucalyptus and identified the attacking machine IP address by analyzing the log. Security, access control, and verification of log were not considered. Patrascu and Patriciu [21] proposed a forensic module to be maintained as part of the cloud's controller module, which is designed to communicate with a different stack of cloud assets (e.g. virtual file system, virtual memory, network stack, and system call interface) in order to collect and store logs. Similarly, security of the collected log (either in transmission or in storage) was not considered.

In an already compromised system (e.g. after an adversary has obtained access to the cloud server's secret key), it is too late to cryptographically ensure that the unsecured log data has not been altered. Bellare and Lee introduced the notion of forward integrity [22] or forward secrecy as a system property to mitigate an attacker's capability to contaminate a logging system without detection. Contamination includes insertion of false logs, modification or deletion of existing logs, and reordering of logs. Forward integrity is established using a cryptographically strong one-way hash function (i.e. HMAC) and a secret key that serves as an initial point of a chain of a pseudo-random function (PRF). In other words, each successive log entry has an associated hash key that is dependent upon the previous entry (similar in functionality to a blockchain).

Schneier and Kelsey proposed a log management scheme [23] based on forward integrity and provided several real-world example applications. Unlike Bellare and Lee, the forward integrity property in the approach of Schneier and Kelsey is ensured using a secret key which is the initial point of a one-way hash chain and message authentication code rather than PRF. The basis of both schemes relies on the fact that, keeping a small and secret piece of information with each log entry which cannot be generated without a secret key, and this secret key changes with each new log. With this secret information, a log entry can be verified later on for its integrity. However, such schemes [22, 23] require the presence of an online trusted server to maintain the secret key and to verify its integrity. To remove the need for an online trusted server, Holt proposed using public key cryptography [24]. Specifically, instead of using the one-way hash function, the author used a digital signature or identity-based signature and in lieu of private keys residing in the trusted server, the author proposed encrypting with public key cryptography and keeping the encrypted keys with the log entries. A known limitation of public key cryptography is the associated computational overheads; thus, it was proposed to use an elliptic curve cryptosystem. However, these

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/TSUSC.2018.2833502, IEEE Transactions on Sustainable Computing

approaches do not consider privacy protection from an honest but curious logger, which is the baseline adversary model typically used in the security literature.

There are other attacks and security properties that need to be considered, such as the truncation attack and delayed detection challenge. In a truncation attack, an attacker truncates some log entries without detection, and the delayed detection problem occurs when the contamination of a log continues until someone detects such contamination. To address both truncation attack and delayed detection challenge, Ma and Tsudik proposed the concept of forwarding secure sequential aggregate (FssAgg) [25]. In this approach, two tags (known as FssAgg tags) are associated with each log entry, namely: one is for the semi-trusted log accumulator and other is for the trusted verifier. Using FssAgg, they were able to ensure forward secure stream integrity instead of forwarding security. Also, an FssAgg tag can ratify any log prior to it in a certain epoch; thus, the last FssAgg tag of an epoch can `testify' the entire chain of log entries up to that epoch. This, however, incurs additional computation costs during the verification phase.

The schemes discussed so far are designed for a conventional system, rather than a cloud environment. In addition, none of the schemes considers a malicious logger.

Ray et al. presented a framework [26], where a logger or log accumulator will send a series of logs to the cloud server via some authenticated channel. Then, the cloud server will take steps to maintain confidentiality, integrity, verifiability, and availability of secure logs. To prevent truncation error from both ends, they used special type of logs on starting and ending points of each block of logs. To protect logs from security breaches or privacy violation, they encrypt log entries with a chain of sequentially generated keys and to preserve integrity they use another set of keys generated in the same way. However, there is no option for public verifiability due to the use of symmetric key encryption to protect confidentiality and privacy. Verification of log results in a privacy violation by those conducting the verification.

Zawoad et al. proposed a novel scheme "SecLaaS" [16], that has taken into consideration many of the CSP's inherent weaknesses, such as volatility, co-mingling of cloud data, and potential malicious loggers (CSP itself). They have provided a complete solution of secure logging for all cloud activities of VMs (or its clients). We evaluate SecLaaS, identifying a number of weaknesses. We propose an alternative approach that incrementally builds upon SecLaaS in an attempt to respond to these identified weaknesses.

Tian et al. [27] recently proposed a scheme for public auditing of operational behavior in the cloud. They introduced the idea of trusted third party to account for log credibility and to alleviate the some of the burdens faced by forensic investigators. For selective verification, they suggested block based logging. They also used widely recognized hash-chain schemes for forwarding security and append-only property. One novel approach they used is in the authentication structure is Merkle Hash Tree (MHT) for tamper resistance. However, MHT is vulnerable to preimage attack as a direct result of how it functions. In the work of Lokhande and Mane [28], the approach differs from SecLaaS in the proof accumulation, where they used bloom filter based R-tree (BR-tree) to accumulate and publish proof of past log.

3 Threat model & Security Properties

In this section, we will describe some definitions required to understand our scheme, the threat model, an attacker's capability, possible attacks [29] , and the standard security properties that a secure cloud logging system must possess. A summary of notations used in this paper is presented in Table 1.

Table 1. Summary of Notations

Log	Log can be network log, process log, registry log, application log or any customized text that meets the requirement of being stored for investigation purpose.
Log Chain (LC)	LC is a small piece of information that co-exists with its corresponding log in order to maintain the integrity and to prevent any modification of the log (such as addition, modification, deletion, and reordering).
Proof of Past Log (PPL)	PPL is a signature or information about the actual log that will be available publicly for forwarding secrecy [22]. That means if the system is compromised, an attacker cannot change the log without detection. PPL can be used to establish log veracity.
Cloud Service Provider (CSP)	CSP is a cloud service provider in which a user can rent and use computing and storage resources. We assume that a CSP is honest but curious [30]. That means it will serve according to contract agreement but has a curiosity about client activity. We design our (CLASS) scheme to include features to prevent a dishonest CSP.
User	User is a CSP client.
Investigator	An investigator is an individual or entity with legal authority to conduct investigative activities in response to some event. These activities include accessing and assessing the contents of log files supplied by a CSP. It is possible for an investigator to collude with a malicious user or CSP to manipulate the perception of an event.
Auditor	An auditor is an individual or entity who is authorized to verify the integrity of log entries, typically through techniques such

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/TSUSC.2018.2833502, IEEE Transactions on Sustainable Computing

	as PPL. It is assumed that the auditor is always fully trusted.
Content Concealing (CC)	CC is a strategy that helps to withstand against privacy breaches that are the result of collusion between a malicious cloud employee and an investigator.
Content Concealing Key (CC key)	CC key is a pair of the private-public key that is used for concealing log content.CC key (the private key not the public key) should be shared by Shamir's [17] or Blakley's [18] secret sharing approach among some trusted entities.

3.1 Threat Model

Our scheme is designed based on the "trust no one" policy. Any party among the CSP, investigator, and user, should be capable of protecting its own security and privacy against another party or collusion between other parties. Potential challenges to designing forensic enabled cloud logging have been discussed in a number of previous studies [16, 26, 31, 32]. For example, in an insecure cloud logging model, only the CSP can write to a log. An investigator or user can collude with the CSP to modify a log before or after publishing PPL. Thus, if a CSP falsely alters a log, whether in collusion with a malicious user or investigator or not, it can hinder the investigative process and conceal the truth of an event. This could result in an attacker failing to be identified or, more dangerously, attributing the attack to the wrong entity.

Conversely, as the cloud is the host of multiple users, a malicious user can repudiate the log under investigation as his/her own log which can lead the criminal lawsuit to be dismissed. On the other hand, the log contains secretive data of the user and the user's privacy may be vulnerable due to this fact. A malicious investigator can alter the log before presenting to court authorities. Moreover, in collaboration with dishonest CSP or CSP employee(s), the investigator can violate the privacy of the user. Based on the above discussion, possible attacks on secure cloud log are given below:

Modification of Log: A dishonest CSP can modify the log before or after publishing its proof (PPL) upon or beyond collusion with the user or investigator. A malicious investigator may alter the log before presenting to court to save a dishonest user or to frame an honest user. Modification of a log can be of many forms, such as insertion of invalid entries, removal of the crucial entries, changing existing entries, reordering log entries to mislead the investigation and to hide malicious activities.

Privacy Violation: Leakage of a log file can reveal information that is able to be directly linked to a users' identity or is able to aggregate in such a way as to create such a link. Even with cryptographic security, cloud employees can transfer the log to an entity that has the key to decrypt (i.e. an investigator) and thus privacy violation may take place.

Repudiation of Ownership of Log: Cloud servers host many users. This presents the possibility for a malicious cloud user to repudiate that the log files under investigation represent the activity of another user. On the other hand, a CSP can repudiate that it did not write the log under investigation. Likewise in SecLaaS, the CSP writes a log for every user and the user has no visibility regarding log entries. This may raise user suspicions regarding log veracity and credibility.

3.2 Security Properties

CLASS seeks to achieve three properties of cryptography, namely: confidentiality, integrity, and authenticity, in terms of the following criteria.

Correctness: Cloud logs should reflect the correct history of a system's event with the occurring time. Any distortion to it is considered a violation of the correctness property.

Tamper Resistance: No one except real logger can introduce an invalid log entry as a valid one. Any sort of contamination such as the addition of new log entries, modification or deletion of existing log entries or even reordering of log entries requires prevention. At a minimum a tamper resistant scheme prevents an attacker from modification of logs without detection.

Verifiability: Verification should be possible by both the user whose activity is represented in the log and the investigating entity. The auditor or any other party involved in the related litigation need to be able to establish log veracity.

Confidentiality: Log data contains sensitive user information and requires privacy protection. For example, if a user mistakenly puts their password into the username field, the system will record this as a failed sign-in attempt and store the password as username in the log. This illustrates the need for confidentiality for all logged data in addition to data more traditionally viewed as requiring privacy protections.

Admissibility: A secure cloud log should be maintained in a way that allows it to be admissible in a court of law for criminal prosecution. The features of log integrity (correctness and tamper resistance), a chain of custody, and forward secrecy all help to achieve such admissibility.

4 SecLaaS

4.1 Specification

The SecLaaS scheme presents an efficient and robust procedure for cloud log management, from collecting logs to making their proof of veracity publicly available. At the same time, it preserves their integrity, privacy, and forward secrecy. SecLaaS preserves the encrypted log in storage then generates and publishes its proof so that no party can modify it.

Fig. 2 describes SecLaaS steps. The following are notations used in this scheme:

- H(m) = Collision resistance one-way hash function of message m.

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/TSUSC.2018.2833502, IEEE Transactions on Sustainable Computing

- E_{Pa}(m) = Encryption of m with the public key of law enforcement authority (LEA).
- $Signature_{Scsp}$(m) = Signature of message m with private key of CSP.

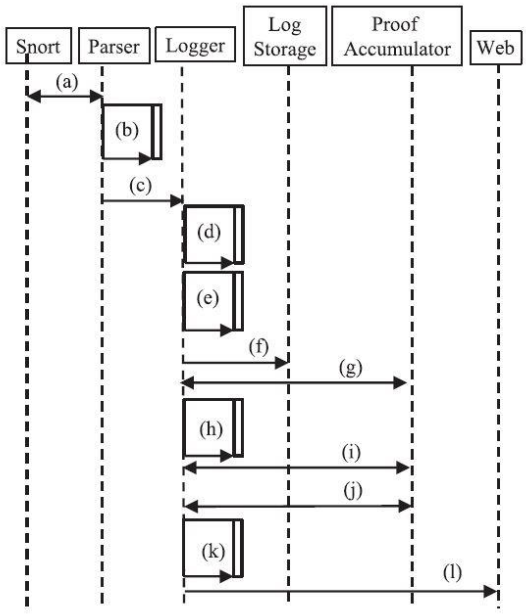


Fig. 2. Detailed SecLaaS scheme process

Now, we present SecLaaS scheme:

a. Parser module communicates with log source to get activity log.
b. The parser parses the log and generates Log Entry, LE. LE can be any useful information required for proper investigation. SecLaaS considers, for sake of demonstration, originating IP address (FromIP), destination IP address (ToIP), time of log (TL), port (Port) and user ID (UserID).
   LE = <FromIP, ToIP, T_L, UserID>
c. Parser sends the LE to logger module for further processing.
d. To preserve the privacy of the user, logger module encrypts some part of LE with law enforcement authority's public key, Pa. Because searching through encrypted data is computationally expensive [33-35], another part of LE is kept unencrypted. Thus, encrypted log entry is generated in this phase.
   ELE = <E_{Pa}(ToIP, Port, UserID), FromID, T_L>
e. After generating ELE, logger module creates log chain (LC) to preserve the integrity of log and to avoid reordering of logs.
   LC = <H(ELE, $LC_{Previous}$)>
f. The logger module prepares database log entry (DBLE) with ELE and LC in order to store it into the persistent database.
   DBLE = <ELE, LC >
g. At this stage, logger module communicates with proof storage to obtain latest accumulator entry
h. The logger module generates membership information of DBLE for accumulator, which depends on chosen accumulator i.e. bit array, bloom filter, one-way accumulator or their own accumulator based on bloom filter named bloom tree.
i. Logger module updates the proof database based on updated accumulator.
j. For each day and for each IP address, logger retrieves the last accumulator entry and denotes it as AE_D.
k. Logger creates proof of past log (PPL) using AE_D.
   PPL = <AE_D, T_L, $Signature_{Scsp}$(AE_D, T_L)>
l. Finally, PPL is published through some web services or RSS feed. The authors proposed a trust model that PPL will be shared among several CSPs so that log can remain intact as long as at least one CSP is honest.

4.2 Analysis

SecLaaS scheme is highly sophisticated and able to solve many problems regarding security, privacy, and admissibility of cloud logs. However, a thorough review reveals some limitations of SecLaaS. The following subsections describe the identified weaknesses of SecLaaS.

4.2.1 Privacy Violation of User by Collusion between Cloud-Employee & Investigator

SecLaaS scheme preserves the log in the persistent database after encrypting it with law enforcement agency's public key so that no one (not even a malicious cloud employee) apart from the correct authority can decrypt it, regardless of access to the log database. Thus, cloud user privacy is ensured and no unauthorized access can take place. However, if someone who knows the private key of the law enforcement agency colludes with a malicious cloud employee and the cloud employee provides some portion of or the entire log database, then user privacy will be compromised. Thus, in the SecLaaS scheme, user privacy can be violated as a result of collusion between malicious cloud employees and law enforcement agencies.

4.2.2 Adulterating Log before Publishing

It is assumed that a CSP will generate a correct log and publish its proof of past log (PPL) honestly. However, this does not account for log alteration prior to PPL publication. SecLaaS can only account for modification or contamination after publishing PPL. Thus, SecLaaS cannot mitigate log modification if the CSP writes false entries prior to publishing its PPL.

4.2.3 Repudiation of Log by Cloud User

Because the CSP acts as a logger it has the sole responsibility of writing and publishing the log. The user has no knowledge of what the CSP is writing as his activity. This may lead to repudiation by a malicious user as the CSP writes log without user's knowledge. Moreover, because of the co-mingling nature of cloud environments, a malicious cloud user can claim that logs (under

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/TSUSC.2018.2833502, IEEE Transactions on Sustainable Computing

investigation) belong to other users and the current scheme has no way to mitigate such claims. However, allowing the user to check his log (immediately after its generation) would enhance CSP accountability and diminish user repudiation.

5 Proposed Scheme: CLASS

In this section, we improve on SecLaaS and present CLASS scheme. We are assuming that in a cloud infrastructure, no party is trusted, that means an attack can come from any party: a CSP, user, or investigator. We are also assuming that cryptographic primitives work properly (i.e. if someone encrypts a message, then nobody can decrypt it without knowing the secret key).

5.1 System Overview

A dishonest cloud user can attack a system outside the cloud. They can also attack any application deployed in the same cloud or an attack can be launched against a node controller which controls all the cloud activities. For a virtual machine (VM), CLASS scheme (Fig. 1) takes the log from the node controller (NC), hides its content, and stores it in a database. This allows logs to become available for further investigation despite VM shutdown. Moreover, CLASS publishes its proof so that log integrity can be protected and admissibility ensured.

5.2 System Details

Before a detailed examination of the proposed scheme, the following definitions and notations are provided:

- $M_1||M_2$: concatenation between two messages M_1 and M_2.
- H(m): collision-resistance one-way hash function of message m.
- $E_{PK}(m)$: encryption of message m with the public key (PK).
- $Signature_{SK}(m)$: signature of message m with private key SK.
- $E_K(m)$: encryption of message m with symmetric key K.

We presume that the cloud service provider (CSP), law enforcement agency (LEA), and user set up their necessary public/private key pairs and publish public keys. PK_C and SK_C are the public and private keys of the CSP respectively and PK_A and SK_A are the public and private keys of the LEA (or auditor). PK_U and SK_U are the public and private keys of a particular cloud user. PK_U and SK_U are titled together as CC-key because they are used to conceal log content of a particular user. During a user's subscription time, the CSP and user mutually generate a CC-key for the user and the CSP only keeps the public portion of the CC-key. The private key (SK_U) of the CC-key is shared among multiple clouds using Rabin's or Blakley's secret sharing scheme.

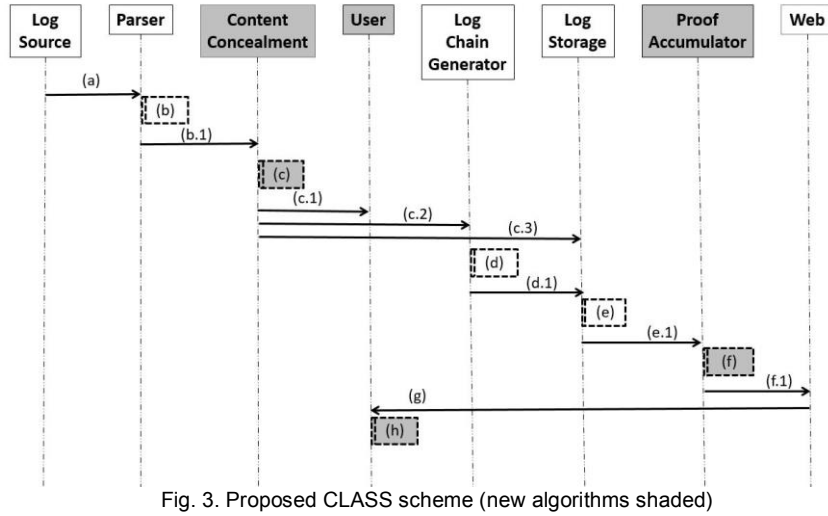


Fig. 3. Proposed CLASS scheme (new algorithms shaded)

5.3 Preservation of Log & Its Proof

Fig. 3 depicts details of CLASS from log retrieval to proof of past log (PPL) publication. Modified portions are shaded in grey. For illustration, we have chosen the network log but it can be any log of the system. Details of our scheme are given in the following step-by-step list:

a. Parser collects the log from log source. For example, we collect log from log file stored in a specific directory. When a log file changes (i.e. new lines append) it triggers the parser to check the change and to start processing new log.

b. Retrieving log from log source, the parser parses the log and gets the necessary information. For the time being there is no standard format of a log and this is another challenge of cloud forensics [26]. The how, when, where, and who, aspects of logs are not yet standardized. Our goal is to keep log content secure given a parser that will provide the system a log message in string format, regardless of content. The format of the log is out of the scope of this work. For our example, we choose originating IP (FromIP), destination IP (ToIP),

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/TSUSC.2018.2833502, IEEE Transactions on Sustainable Computing

user id (UserID) and log time (T_L). With this information, the parser generates log entry (LE).

LE = <FromIP, ToIP, UserID, T_L >

c. Asymmetric encryption of log with individual user's public key PK_U is computed to conceal user's content. This helps to resolve the problem mentioned in section 4.2.1. If we use public key encryption with investigator's public key PK_A and store the log database in CSP's custody then neither cloud employee nor investigator can violate the privacy of user alone. But if a malicious cloud employee provides a portion or full copy of the log database to an investigator then the user's privacy is in jeopardy. Asymmetric encryption with (individual) user's private key addresses this problem, though in turn leads to a potential concern of recovery of the log in the case of an investigation because privacy is provided with user's secret key. This problem can be solved using Shamir or Blackley's secret key sharing scheme and is discussed in the following "secret key sharing" section (section 5.6). After content concealment, the CLASS scheme will generate an encrypted log entry (ELE) and this ELE will be available to the user so that user can cross check if CSP is writing a correct log entry. Because searching through encrypted data is expensive, we keep some data in plain text format for filtering purposes [33, 34, 36].

ELE = <EP_{Ku}(ToIP || UserID), FromIP, T_L>

d. After that, log chain (LC) is created in order to protect the integrity of the log and prevent potential manipulation. CLASS creates LC using a hash function with current ELE and previous LC:

LC = <H(ELE || $LC_{Previous}$) >

e. At this stage, the payload is ready to be stored in the database. CLASS generates database log entry (DBLE) with ELE and LC and stores it in the log database.

DBLE = < ELE, LC >

f. For each DBLE, the CLASS scheme requires the generation of proof of past log (PPL) which is then made publicly available. CLASS generates the PPL in a manner that is designed to minimize the usage of memory space. In CLASS, we propose to generate PPL in a batch of the logs of a certain period or a certain amount of logs (e.g. n number of logs). At the end of each epoch, for each DBLE in a batch, CLASS concatenates each of the logs in a chain of logs in chronological order, derives the fingerprint, FP using Rabin's fingerprint [19]. Then the CLASS scheme constructs an accumulator entry (AE) which is bloom filter membership information of the fingerprint FP. For each static IP, CLASS retrieves accumulator entry (AE), epoch time T_E, a signature using CSP's private key and concatenates AE, T_E, with its signature, generating PPL.

g. Finally, the CLASS scheme publishes PPL to the web via RESTful API or RSS feed:

FP = FP (LC_1 || LC_2 || … … … || LC_N)

AE = BloomFilter(FP)

PPL = <AE, T_E, $Singature_{SKc}$(AE, T_E)>

After PPL is publicly available, it can be shared among multiple CSPs to build a trust model. This mitigates the potential for forgery so long as a single CSP is honest.

h. After the publishing of PPL, the parser can verify their log, which is readily available in each epoch. If the user finds any discrepancy then the user can take steps accordingly. In this case, the user has to check two tests: backward check and forward check as described in Section 5.5. A backward check is to verify if logs accurately depict user activity and a forward check is to verify if the cloud server publishes the correct PPL for the corresponding logs.

CLASS algorithms can be categorized into two major groups: One for LogPreservation (see Algorithm 1) and one for ProofAccumulation (see Algorithm 2). The LogPreservation algorithm can take log entries individually or in a batch and performs processing prior to storage in a log database. This algorithm encrypts for secrecy and generates hash digest for consistency. The ProofAccumulator algorithm performs daily processing of all log entries corresponding to an IP address to prepare and publish proof of past log (PPL). Pseudocode for both of these algorithms is provided below.

```
LogPreservation( log entries LEs)
    for i ← 1 to size( LEs )
        encrypted_log_i = encrypt( log_entry_i )
        log_chain_i = hash( encrypted_log_i || log_chain_{i-1} );
        Database_log_entry_i = < encrypted_log_i, log_chain_i >;
        store database_log_entry_i into log database;
    end for;
```

Algorithm 1. LogPreservation pseudocode for processing log entries

```
ProofAccumulation( log entries LEs)
    chronological_concatinate_LEs = LE_1 || LE_2 || … || LE_n;
    finger_print = FingerPrint( chronological_concatinate_LEs );
    accumulator_entry = BloomFilter( finger_print );
    signature = Signature( acuumulator_entry, time);
    Publish < accumulator_entry, time, signature >;
end;
```

Algorithm 2. ProofAccumulation pseudocode to generate and publish proof of past log (PPL)

5.4 Accumulator Design

Zawoad et al. [37] used Bloom filter as a proof of past data possession, which is fails to account for Bloom filter's inherent potential for false positives. When using a Bloom filter technique, there is a trade-off between the number of false positives and the size of the filter. To mitigate this problem, a cryptographic one-way accumulator could be

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/TSUSC.2018.2833502, IEEE Transactions on Sustainable Computing

used. However, this requires significant computational overhead. In SecLaaS, they used their own data structure Bloom Tree that reduced the number of false positive incidents but requires an increased number of instances of logs and significant computational resources at verification time. This is true regardless of the number of entries being verified. In addition, it still remains vulnerable to false positives (albeit reduced).

CLASS proposes a bloom filter based PPL that computes membership information of (Rabin's) fingerprint [19] of chronologically ordered log chain of an epoch. It requires one single bloom filter for an entire batch of logs. The fingerprint has a total reflection of the entire log chain in a particular epoch. For example, if there are L logs in an epoch and log chains of these logs in chronological order are LC_1, LC_2, LC_3, ..., LC_L. Fingerprint FP of these log chains is,

$$\mathbf{FP = FingerPrint(LC_1 \,||\, LC_2 \,||\, LC_3 \,||\, ... \;\; ... \;\; ... \,||\, LC_L)}$$

Then a bloom filter with membership information of FP is accumulator entry (AE) for this epoch of log. If the hash function for generating bloom filter information are: $hash_1$, $hash_2$, $hash_3$, ..., $hash_n$.

$$\mathbf{v_1 = hash_1(FP)}$$
$$\mathbf{v_2 = hash_2(FP)}$$
$$\mathbf{v_3 = hash_3(FP)}$$
$$\mathbf{... \;\; ... \;\; ...}$$
$$\mathbf{v_n = hash_n(FP)}$$

A bloom filter with 1 at v_1th, v_2th, v_3th, ..., v_nth positions and 0 at rest of the positions represents accumulator entry, AE. Consequently, PPL for this epoch of log is a combination of accumulator entry (AE), epoch time (TE) and signature of this information by CSP.

$$\mathbf{PPL = < AE, T_E, Signature_{CSP}(AE, T_E) >}$$

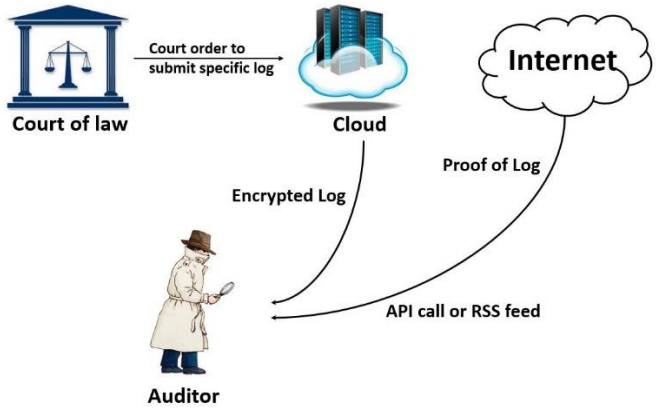


Fig 4. Overview of verification process

If the size of the bloom filter is m and number of hash functions for bloom filter (membership calculation) are n then there are m^n distinct options for a fingerprint. Thus the false positive rate, or the probability of two different fingerprints having the same bloom filter membership information is m^{-2n}. Adjusting the size of bloom filter and a number of hash functions we can optimize the false positive rate of bloom filter in the CLASS scheme. CLASS has a very low false positive rate in PPL since it holds only one entry (i.e. fingerprint) per bloom filter.

5.5 Verification

Only a verification process that establishes authenticity will be able to determine the presence of log contamination. There are two types of verifications in our approach. First is verification where the user checks if the CSP is writing correct log entries or not. Second is verification by any party: user, investigator, law enforcement authority (LEA) or court of law to verify PPL to check for log modification. In both cases, the CSP can provide a small utility verification software or the user/investigator can buy it from individual software vendor (ISV) to verify.

Correct log: In the CLASS scheme, after symmetric encryption, encrypted log (ELE) gets available to the user via some secure channels. Then the user can decrypt and determine the authenticity of log entries. Though it should not be presumed that a user will have an understanding of low-level information within a log, the user may be able to verify high-level information such as sites visited, operations launched and so on. In this way, a user is able to establish log veracity.

PPL verification: This verification is required for forwarding secrecy so that the CSP can't modify logs after publishing proof of log to the public (Fig. 4). Any party with sufficient credentials, e.g. user or investigator or auditor, can verify PPL. Once the log is available, the user (or auditor) collects encrypted log (ELE) for a specific epoch, computes its PPL exactly in the same way the system generated the PPL and cross-matches with published PPL to verify.

5.6 Secret Key Sharing

We propose, in CLASS, to encrypt the log with the user's private key (CC-key). In recognition that this might lead to permanent loss of log data (even for investigative entities), as the private key of a user's CC-key is known only to the user, we propose to share individual user's private key according to Shamir's or Blackley's secret key sharing strategy among multiple CSPs.

This sharing scheme requires standardization. We can build sharing clouds for such a purpose when a user subscribes to a cloud service. The CSP and user together choose a pair of public/private key that is called the content concealing key (or CC-key) because it is used to hide user's log content. For usage of CC-key following rules are followed.

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/TSUSC.2018.2833502, IEEE Transactions on Sustainable Computing

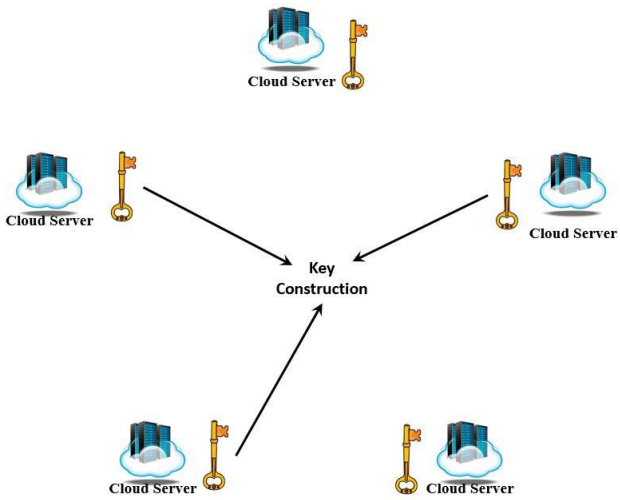


Fig. 5. All cloud servers extract information from private key and three yield their key information to construct the key

- The user owns this CC-key.
- CSP keeps only the public portion of the CC-key for encrypting the user's log.
- Small information will be extracted from private key following Shamir's secret sharing algorithm and each portion will be shared to one CSP. In Fig. 5, the keys represent secret portions of the private key that is shared between five different cloud servers.
- Sharing cloud servers should have no communication/collusion between them.
- Sharing cloud servers should provide secret information of a particular user only when the court of law orders them to provide.
- When it requires investigating a cloud user's log, court-of-law will order a specific number of cloud servers to provide their secret portions for a particular user (as shown in the key construction portion of Fig. 5).
- After getting secret portions of a particular user, the host cloud can reconstruct the private key to decrypt the log of that user using Shamir's secret sharing algorithm.

6 Security and Complexity Evaluation

In this section, we evaluate the security of the CLASS scheme with respect to the properties described in section 3.2 and then present a complexity analysis contrasting CLASS and SecLaas.

6.1 Security Evaluation

Correctness: In a cloud environment, the CSP acts as a logger and writes log entries for user activity without user knowledge. CLASS ensures correctness of log entries by allowing the user to verify those written by the CSP. In this way, CLASS mitigates the effects of CSP activity that would violate the correctness property of a log.

Tamper Resistance: In CLASS and also in SecLaaS, only the CSP can modify the log. Any of collusion among the CSP, investigator, or malicious user, will result in log contamination after publishing its proof. Thus, the victim party can find evidence of this in the log verification phase.

Verifiability: CLASS facilitates verification of log by any parties with the appropriate credentials. It publishes the proof of log (i.e. PPL) publicly and anyone can collect it with the proper credentials. For verification, an authorized party can collect logs from the CSP and cross check with published proof.

Confidentiality & Privacy: The log is encrypted with individual user's public key, hence it requires a private key of that user to unlock his/her log. In this sense, CLASS is a privacy-preserving scheme. Furthermore, it is difficult to trace or link back to an actual log entry from only the published proof of past log (PPL) because PPL is generated using one-way hash functions.

Admissibility: Because the CLASS scheme is resilient to any sort of log modification happening without the possibility of being detected, log files from this scheme will be admissible in a court of law and no malicious party can repudiate its involvement in a misdeed.

In addition to fulfilling the security properties, CLASS is able to overcome some of the limitations inherent to SecLaaS. Each of the following three paragraphs presents such a limitation and describes how CLASS can overcome it.

Privacy violation of user by collusion between cloud-employee & investigator: CLASS suggests encrypting logs using the public key of the individual user before storing them in cloud storage. Thus, if a malicious cloud employee and an investigator collude and the cloud employee provides some portion of or the entire log database to the investigator, privacy will be maintained because logs are encrypted by the user's public key.

Modifying logs before publishing: Because the user can access their encrypted log even before publishing its proof (i.e. PPL), they are able to verify log entries. Upon obtaining an encrypted log, the user can decrypt the log and check if it contains any activities they did not perform. If the CSP performs any illicit writing, then the user can identify it shortly after (and prior to publishing PPL) and take appropriate action, such as challenging CSP activity or unsubscribing from the CSP's service. Thus, the CLASS scheme inhibits CSP modification of the log even before publishing PPL.

Repudiation of Log by Cloud User: In cloud computing environment, CSP writes log entries for all users and users have no knowledge of what CSP is writing as their activities. This opens up a justification for a cloud user to repudiate their logs. In CLASS, the user has the privilege to access his log just after CSP writing it. Consequently, the user can raise an objection if any discrepancy found in the log. Thus, CLASS minimizes the possibility of any repudiation of the log by a cloud user. Table 2 summarizes the comparison between SecLaaS and CLASS.

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/TSUSC.2018.2833502, IEEE Transactions on Sustainable Computing

6.2 Complexity Evaluation

Both SecLaaS and CLASS schemes compute cryptographic operations that are highly dependent on key size. Because both schemes have a variety of configurable settings, a comparison of their performance in terms of time and memory space is needs to be conducted according to their workflow. As described in subsection 5.3, the work flow of CLASS (and also SecLaaS) can be divided into two major groups: log preservation and proof accumulation.

Log preservation encrypts and generates hash digest of user log for its confidentiality and integrity respectively. This process starts with getting the log from log source and ends with storing it into log storage. Both schemes compute a public key encryption and a hash digest computation for each log entry. The only difference is the management/ownership of the cryptographic key. If C_{PK} and C_H is the cost (either execution time or storage size) of public key encryption and hash digest generation respectively then the cost for processing n_L number of logs is $n_L(C_{PK} + C_H)$ for both SecLaaS and CLASS. Hence, there is only a qualitative difference between the two schemes in this phase (no quantitative difference).

Unlike the log preservation process, the *proof accumulation* process differs significantly between SecLaaS and CLASS with CLASS having a clear upper hand over its predecessor. Proof accumulation is a process of computing bloom filter membership information of different number of logs either at single or multiple levels. SecLaaS and CLASS compute proof using bloom tree and bloom filter respectively where bloom tree is a multi-level bloom filter and its branching factor is configurable. Let, b_{BF_size} is size of bloom filter, $k_{BF\_hash\_count}$ is number hash functions of bloom filter for both schemes. If each bloom filter contains m number of logs and we consider 2 levels in bloom tree for SecLaaS then to accumulate proof of n_L number of logs, SecLaaS requires $(n_L + \left\lceil \frac{n_L}{m} \right\rceil) \times k_{BF\_hash\_count}$ number of hash operations. In contrast, CLASS requires only $k_{BF\_hash\_count}$ number of hash operations preceded by faster executable Rabin's Signature computation of n_L number of concatenated logs. On the other hand, if there are n_L number of logs for a static IP in a single day then it requires $\frac{b_{BF\_size}}{8}$ bytes of storage to store proof of past log in either scheme. However, similar to proof generation time, SecLaaS requires more dynamic memory (i.e. RAM) for proof generation when compared to CLASS.

TABLE 2. COMPARISON BETWEEN SECLAAS SCHEME AND OUR PROPOSED SCHEME

Challenge / Threat	SecLaaS Scheme	CLASS Scheme
Modification of cloud-log after publishing proof of past log (PPL)	Detected	Detected
Modification of cloud-log before publishing PPL	Remains undetected	User avails a way to detect it beforehand
User's privacy violation either by cloud server or by investigator	Privacy preserved	Privacy preserved
User's privacy violation by collusion between cloud-employee and investigator	Privacy deteriorated	Privacy preserved
Repudiation of cloud-log by cloud-server	Detected	Detected
Repudiation of cloud-log by cloud-user	Undetected	Detected
Log recovery for investigation	Recovered	Recovered even when the log files were encrypted with user's private key.

7 PERFORMANCE EVALUATION

In this section, we discuss the details for the implementation of our proposed CLASS scheme and follow that with an analysis of its performance.

7.1 Setup

We deployed CLASS scheme on an open source cloud computing platform OpenStack2 (juno) [38]. We conduct our experiment with the minimal setup of two nodes to evaluate the performance of CLASS. We implement OpenStack legacy networking (nova-network) that is composed of two machines: one for the controller and the other for the computing node. We used Ubuntu 14.04 LTS 64 bit version as host operating system for both machines.

1. **Hardware for Controller node:** Minimum hardware requirement for a controller node: CPU - 1~2, RAM - 8 GB, Hard disk drive - 100 GB, Network interface card – 1[38]. In our system, we used Intel core 2 duos, 8 GB of RAM and 320 GB of hard disk.
2. **Hardware for Compute node:** Minimum hardware requirement for a compute node: CPU - 2~4, RAM - 8+ GB, Hard disk drive - 100+ GB, Network interface card – 2 [38]. In our system, we

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/TSUSC.2018.2833502, IEEE Transactions on Sustainable Computing

used Intel i5 processor, 16 GB of RAM and 1 TB of hard disk.

3. **Network Configuration:** The controller node consists of one network interface i.e. management network. Its IP is 10.0.011/24, the default gateway is 10.0.0.1 and host name is "controller". The compute node consists of two network interfaces, one is management network and other is to the external network. Management network's IP address is 10.0.0.31/24 and the default gateway is 10.0.0.1. The external network used special configuration without IP address. Its hostname is "compute1".

The class scheme is implemented with Java 1.7 update 79 and to expose web service we use servlet-JSP technology. As cryptographic tools, we used AES for symmetric key encryption, 2048 bit RSA for public key encryption, MD5 for standard hash function. We define our bloom filter with size 1024 bits and 64 different hash functions of our own. For each step, we used a separate java class to process incoming log messages and to hand over to the next phase such as LogSource, LogParser, LogChainGeneration, ProofAccumulator, WebPublication etc.

To expose RESTful API for log and its proof for a particular day we used apache tomcat as a web server and servlet technology. When someone requests for log (or proof) with a date (with format YYMMDD), our API returns log (or proof) to that particular date in JSON format after authentication [39]. We could return these in XML format as well. However, since JSON format takes less space, is easy to parse, and is widely used in this area, we used JSON using JAX-RS. After retrieving a log and its proof, one can verify the authenticity of the log. In the proposed scheme, one can obtain the log and the proof separately. We developed a lightweight and standalone program for this purpose.

7.2 Performance Analysis

CLASS differs from SecLaaS scheme mainly in two different ways. Firstly, during processing of log, CLASS incorporates asymmetric encryption with individual user's public key (instead of investigator's public key like in SecLaaS) to avoid data leakage. Secondly, during proof accumulation, CLASS incorporates fingerprinting of all log entries of a single IP address in an epoch to minimize false positive in bloom filter. Our experiment setup suggested that it is feasible to implement the proposed secure cloud logging mechanism. We implemented both schemes and collected some metrics for comparison.

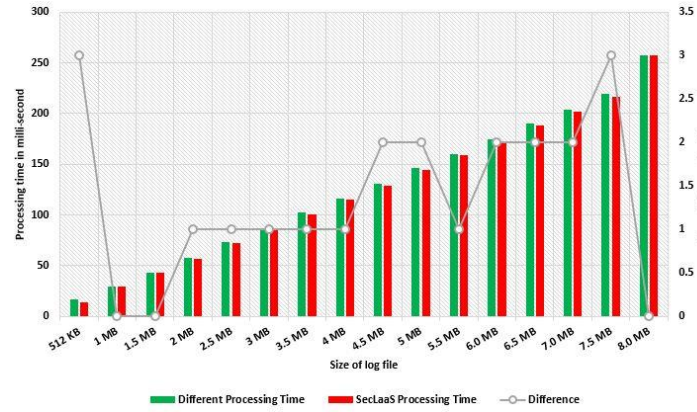


Fig. 6. Log processing time comparison

7.2.1 Log Processing Time

We have taken 512KB of the log file with almost 100,000 log entries and have executed both schemes. Similarly, we run on files of 1MB, 1.5MB, 2MB, 2.5MB, 3MB, 3.5MB, 4MB, 4.5MB, 5MB and so on. Our system is performing the same number of operations as the SecLaaS scheme does. Thus, in the execution phase, there is no difference between the CLASS and SecLaaS schemes.

However, if we consider an extra symmetric key encryption (with individual user's secret key) before asymmetric key encryption with investigator's public key to hide data, then processing time would increase minimally in comparison to that of the SecLaaS scheme's time, which is depicted in Fig. 6. This shows the difference in execution time between SecLaaS and CLASS is very small which is ratified by the fact that, symmetric key encryption takes a little time. However, we replaced SecLaaS's asymmetric key encryption that uses the investigator's public key with that of individual user's public key. Thus, a number of executed operations remain same and CLASS's time remains same as that of SecLaaS.

7.2.2 Verification Time

In the case of testifying proof of log, we consider 5000 to 85000 log entries at each epoch with an increment of 5000 log entries at each step. In CLASS, we concatenate each log entries of an epoch and generate the fingerprint. Later, this fingerprint's membership in a bloom filter is calculated and published: for one epoch one fingerprint and one bloom filter. In this case, fingerprint reflects each log entry of an epoch and bloom filter's accuracy improves as only one member resides in a bloom filter. By comparison, SecLaaS generates bloom tree, which is a multi-level bloom filter and inserts each log entry separately into bloom filter. Thus, SecLaaS' verification time is longer than that of CLASS' which is shown in Fig. 7.

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/TSUSC.2018.2833502, IEEE Transactions on Sustainable Computing

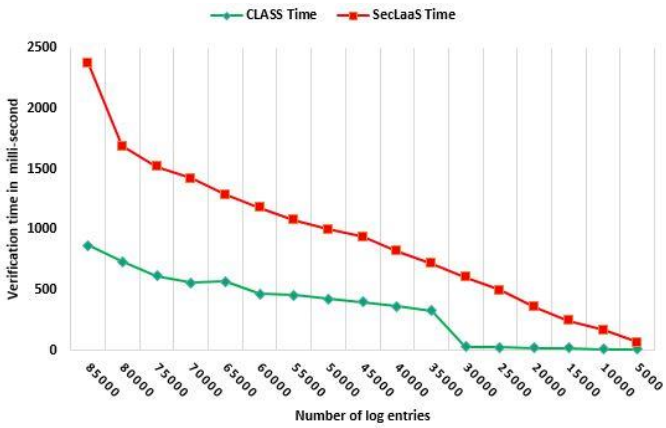


Fig. 7. Verification time comparison

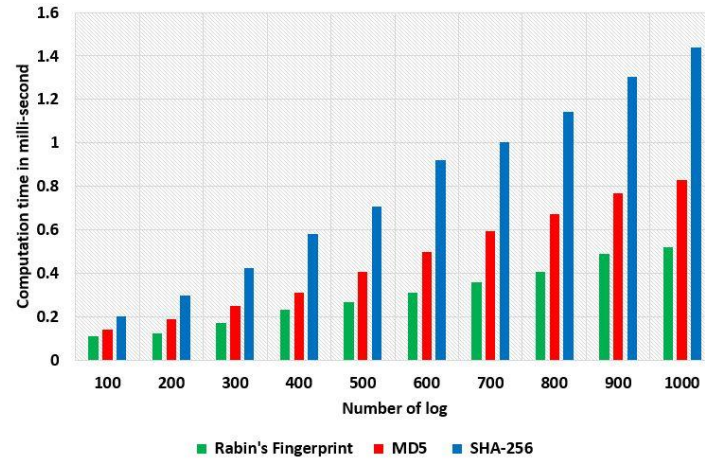


Fig. 9. Comparison between different hash digest

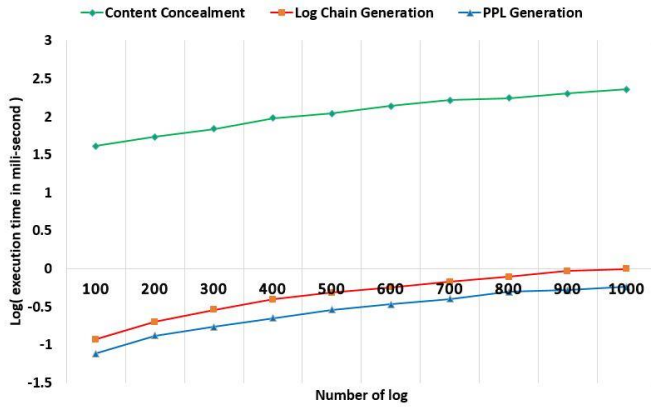


Fig. 8. Comparison between different operations execution time

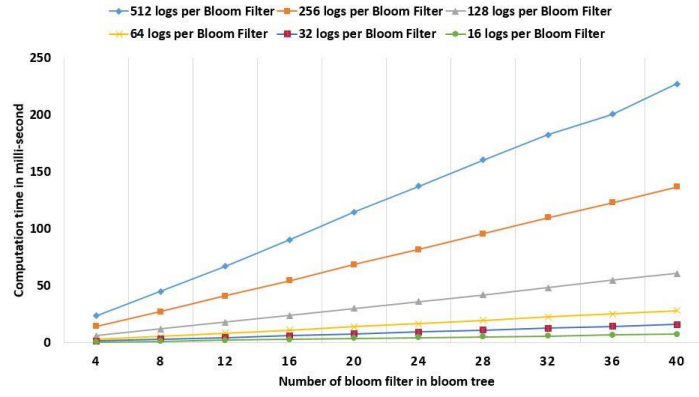


Fig. 10. Different Bloom Tree comparison

7.2.3 Summary

The proposed secure logging mechanism consists of two phases, one is the log processing phase and the other is the log verification phase. The log processing phase has three computation steps: content concealment, log chain generation and PPL generation. Content concealment is ensured by asymmetric encryption via the user's public key. Log chain generation with current log and previous log chain come after content concealment. This is as simple as collecting the previous log, concatenating it with current encrypted log and computing (MD5) hash digest. PPL generation is putting fingerprint of all log chains of an epoch into bloom filter. Fig. 8 illustrates the computation times for content concealment, log chain generation and PPL generation. We prefer to compute Rabin's fingerprint whenever we need a hash digest of a one-way hash function (i.e. for PPL Generation) since Rabin's fingerprint has faster computation time. We collected some empirical data and present it in Fig. 9.

Despite the fact that BloomTree has lower false positive rate than the Bloom filter, we prefer fingerprint based accumulator because BloomTree takes a longer time to verify even a single log entity. However, BloomTree still has the potential for false positives. In our experiment, we constructed BloomTree with a different number of bloom filters and observed that BloomTree with more bloom filter takes more time to generate accumulator or to verify log. Fig. 10 illustrates this fact for BloomTree with a different number of bloom filters (i.e. 4, 8, 12, 16, 20 and so on) per BloomTree. BloomTree with more logs grows exponentially than that of less number of logs. Using the fingerprint of all the log chains in an epoch and single bloom filter per fingerprint decreases false positive results as well as computation time. Fig. 11 demonstrates this fact. Thus, the CLASS scheme is able to reduce verification time in comparison to the SecLaaS scheme, as shown in Fig. 8.

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/TSUSC.2018.2833502, IEEE Transactions on Sustainable Computing

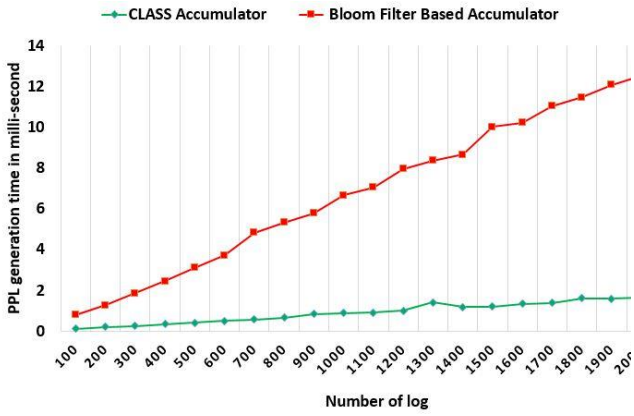


Fig. 11. Comparison between our accumulator and Bloom Filter based accumulator

8 Conclusion

In this paper, we proposed a secure logging scheme (CLASS) for cloud computing with features that facilitate the preservation of user privacy and that mitigate the damaging effects of collusion among other parties. CLASS preserves the privacy of cloud users by encrypting cloud logs with a public key of the respective user while also facilitating log retrieval in the event of an investigation. Moreover, it ensures accountability of the cloud server by allowing the user to identify any log modification. This has the additional effect of preventing a user from repudiating entries in his own log once the log has had its PPL established. Our implementation on OpenStack demonstrates the feasibility and practicality of the proposed scheme. The experimental results show an improvement in efficiency thanks to the features of the CLASS scheme, particularly in verification phase. Potential future extensions include the following:

1. Normally logs are low-level data and hard for the common user to understand what exactly those logs signify. Thus, we will explore leveraging big data techniques to facilitate user retrieval and visualization of information from log data. Standardization of log format is also an associated research area.
2. To ease searching, we kept some crucial and sensitive information in plaintext format. This makes them vulnerable to be exposure. Thus, designing secure and efficient searchable encryption would extend this work.
3. There is also the need for an online credibility system designed to develop trust and credibility of a cloud user so that the CSP can enable stricter auditing policies for low-trust users in comparison to high-trust users.
4. Designing and implementing a prototype of the proposed scheme in collaboration with a real-world CSP, with the aim of evaluating its utility (e.g. performance and scalability) in a real-world environment.

References

[1] X. Liu, R. H. Deng, K.-K. R. Choo, and J. Weng, "An efficient privacy-preserving outsourced calculation toolkit with multiple keys," IEEE Transactions on Information Forensics and Security, vol. 11, pp. 2401-2414, 2016.

[2] Y. Mansouri, A. N. Toosi, and R. Buyya, "Data storage management in cloud environments: Taxonomy, survey, and future directions," ACM Computing Surveys (CSUR), vol. 50, p. 91, 2017.

[3] M. Tao, J. Zuo, Z. Liu, A. Castiglione, and F. Palmieri, "Multi-layer cloud architectural model and ontology-based security service framework for IoT-based smart homes," Future Generation Computer Systems, vol. 78, pp. 1040-1051, 2018.

[4] Z. Xia, Y. Zhu, X. Sun, Z. Qin, and K. Ren, "Towards privacy-preserving content-based image retrieval in cloud computing," IEEE Transactions on Cloud Computing, pp. 276-286, 2018.

[5] L. Zhou, Y. Zhu, and A. Castiglione, "Efficient k-NN query over encrypted data in cloud with limited key-disclosure and offline data owner," Computers & Security, vol. 69, pp. 84-96, 2017.

[6] Q. Alam, S. U. Malik, A. Akhunzada, K.-K. R. Choo, S. Tabbasum, and M. Alam, "A Cross Tenant Access Control (CTAC) Model for Cloud Computing: Formal Specification and Verification," IEEE Transactions on Information Forensics and Security, vol. 12, pp. 1259-1268, 2017.

[7] L. Li, R. Lu, K.-K. R. Choo, A. Datta, and J. Shao, "Privacy-preserving-outsourced association rule mining on vertically partitioned databases," IEEE Transactions on Information Forensics and Security, vol. 11, pp. 1847-1861, 2016.

[8] K.-K. R. Choo, M. Herman, M. Iorga, and B. Martini, "Cloud forensics: State-of-the-art and future directions," Digital Investigation, pp. 77-78, 2016.

[9] C. Esposito, A. Castiglione, F. Pop, and K.-K. R. Choo, "Challenges of Connecting Edge and Cloud Computing: A Security and Forensic Perspective," IEEE Cloud Computing, vol. 4, pp. 13-17, 2017.

[10] Z. Qi, C. Xiang, R. Ma, J. Li, H. Guan, and D. S. Wei, "ForenVisor: A tool for acquiring and preserving reliable data in cloud live forensics,"

IEEE Transactions on Cloud Computing, vol. 5, pp. 443-456, 2017.

[11] C. Hooper, B. Martini, and K.-K. R. Choo, "Cloud computing and its implications for cybercrime investigations in Australia," Computer Law & Security Review, vol. 29, pp. 152-163, 2013.

[12] O. Osanaiye, K.-K. R. Choo, and M. Dlodlo, "Distributed denial of service (DDoS) resilience in cloud: review and conceptual cloud DDoS mitigation framework," Journal of Network and Computer Applications, vol. 67, pp. 147-165, 2016.

[13] N. H. Ab Rahman, W. B. Glisson, Y. Yang, and K.-K. R. Choo, "Forensic-by-design framework for cyber-physical cloud systems," IEEE Cloud Computing, vol. 3, pp. 50-59, 2016.

[14] B. Martini and K.-K. R. Choo, "An integrated conceptual digital forensic framework for cloud computing," Digital Investigation, vol. 9, pp. 71-80, 2012.

[15] S. Khan, A. Gani, A. W. A. Wahab, M. A. Bagiwa, M. Shiraz, S. U. Khan, et al., "Cloud log forensics: foundations, state of the art, and future directions," ACM Computing Surveys (CSUR), vol. 49, p. 7, 2016.

[16] S. Zawoad, A. K. Dutta, and R. Hasan, "Towards building forensics enabled cloud through secure logging-as-a-service," IEEE Transactions on Dependable and Secure Computing, vol. 13, pp. 148-162, 2016.

[17] A. Shamir, "How to share a secret," Communications of the ACM, vol. 22, pp. 612-613, 1979.

[18] G. R. Blakley, "Safeguarding cryptographic keys," Proc. of the National Computer Conference1979, vol. 48, pp. 313-317, 1979.

[19] M. O. Rabin, Fingerprinting by random polynomials: Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.

[20] F. Anwar and Z. Anwar, "Digital forensics for eucalyptus," in Frontiers of Information Technology (FIT), 2011, 2011, pp. 110-116.

[21] A. Patrascu and V.-V. Patriciu, "Logging system for cloud computing forensic environments," Journal of Control Engineering and Applied Informatics, vol. 16, pp. 80-88, 2014.

[22] M. Bellare and B. Yee, "Forward integrity for secure audit logs," Technical report, Computer Science and Engineering Department, University of California at San Diego1997.

[23] B. Schneier and J. Kelsey, "Secure audit logs to support computer forensics," ACM Transactions on Information and System Security (TISSEC), vol. 2, pp. 159-176, 1999.

[24] J. E. Holt, "Logcrypt: forward security and public verification for secure audit logs," in Proceedings of the 2006 Australasian workshops on Grid computing and e-research-Volume 54, 2006, pp. 203-211.

[25] D. Ma and G. Tsudik, "A new approach to secure logging," ACM Transactions on Storage (TOS), vol. 5, p. 2, 2009.

[26] 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.

[27] H. Tian, Z. Chen, C.-C. Chang, M. Kuribayashi, Y. Huang, Y. Cai, et al., "Enabling public auditability for operation behaviors in cloud storage," Soft Computing, pp. 1-13, 2016.

[28] P. Lokhande and V. Mane, "Log Based Privacy Preservation in Cloud Forensic."

[29] S. Iqbal, M. L. M. Kiah, B. Dhaghighi, M. Hussain, S. Khan, M. K. Khan, et al., "On cloud security attacks: A taxonomy and intrusion detection and prevention as a service," Journal of Network and Computer Applications, vol. 74, pp. 98-120, 2016.

[30] S. Yakoubov, V. Gadepally, N. Schear, E. Shen, and A. Yerukhimovich, "A survey of cryptographic approaches to securing big-data analytics in the cloud," in High Performance Extreme Computing Conference (HPEC), 2014 IEEE, 2014, pp. 1-6.

[31] R. Marty, "Cloud application logging for forensics," in Proceedings of the 2011 ACM Symposium on Applied Computing, 2011, pp. 178-184.

[32] S. Zawoad and R. Hasan, "Digital forensics in the cloud," DTIC Document2013.

[33] Z. Fu, X. Wu, C. Guan, X. Sun, and K. Ren, "Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement," IEEE Transactions on Information Forensics and Security, vol. 11, pp. 2706-2716, 2016.

[34] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," IEEE Transactions on Parallel and Distributed Systems, vol. 27, pp. 340-352, 2016.

[35] P. Xu, H. Jin, Q. Wu, and W. Wang, "Public-key encryption with fuzzy keyword search: A provably secure scheme under keyword guessing attack," IEEE Transactions on computers, vol. 62, pp. 2266-2277, 2013.

[36] C. Wang, N. Cao, K. Ren, and W. Lou, "Enabling secure and efficient ranked keyword search over outsourced cloud data," IEEE

Transactions on parallel and distributed systems, vol. 23, pp. 1467-1479, 2012.

[37] S. Zawoad and R. Hasan, "Towards building proofs of past data possession in cloud forensics," ASE Science Journal, vol. 1, pp. 195-207, 2012.

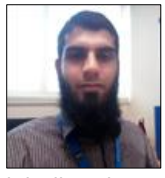
[38] "OpenStack Installation Guide for Ubuntu," 2017.

[39] S. Khan, M. Shiraz, L. Boroumand, A. Gani, and M. K. Khan, "Towards port-knocking authentication methods for mobile cloud computing," Journal of Network and Computer Applications, vol. 97, pp. 66-78, 2017.

M A Manazir Ahsan received the BSc degree in computer science and engineering from the Bangladesh University of Engineering and Technology (BUET) in February 2011.Currently he is working toward his masters (by dissertation) degree at the University of Malaya (UM) in Malaysia. He is an IEEE student member. His research interest lies in information security domain. Specially, he loves to work with cloud forensics/security, trusted computing, secure provenance and quantum computing/cryptography.

Ainuddin Wahid Bin Abdul Wahab received the BCompSc and MCompSc degrees from the University of Malaya (UM), Malaysia, the PhD degree from Surrey University, Surrey, UK. Currently he is working as senior lecturer at the Faculty of Computer Science & Information Technology at the University of Malaya, Malaysia. He is an active member of Malaysian Society of Cryptology Research. His area of expertise includes digital forensic, steganography, network security, public key infrastructure and biometrics.

Mohd Yamani Idna Bin Idris received his bachelor in engineering, the MCompSc and the PhD from University of Malaya, Malaysia. Currently he is a Senior Lecturer at the Faculty of Computer Science & Information Technology, University of Malaya, Malaysia. He is an IEEE member. His area of expertise encompasses image/signal processing, sensor network (visual sensor network, wireless sensor network), digital forensics and security systems.



Suleman Khan is a faculty member at School of Information Technology, Monash University Malaysia. He received his Ph.D. (Distinction) from Faculty of Computer Science and Information Technology, University of Malaya, Malaysia (2017). Previously, he completed several Master programs including Master of Science-MS (Distributed Systems) from Comsats Institute of Information Technology, Abbottabad, Pakistan (2011), Master of Business Administration (HRD) from Institute of Management of Sciences, Hayatabad, Pakistan (2007) and Master of Science M.Sc (Computer Science), from University of Peshawar, Pakistan (2006). Dr. Suleman has published 40+ High Impact Research articles in reputed international journals and conferences. He is currently an IEEE member and his research areas include but are not limited to Network Security, Network Forensics, Software Defined Networks (SDN), Internet of Things (IoT), Cloud Computing, and Vehicular Communications.

Eric Bachura received the MBA degree from Sam Houston State University, Huntsville, Texas in 2014. In 2016, he joined the Ph.D. program in information systems at The University of Texas at San Antonio. Now in his second year of the program, he is maintaining a 4.0 GPA, received the best paper award at the 2017 Americas Conference on Information Systems (AMCIS), is teaching a course on Data Analytics for Security, and is working as on multiple concurrent research projects with academics from multiple disciplines. His prior work experience includes a five-year honorable term as an active duty United States Marine and a number of technical and managerial positions as a United States Department of Defense contractor. His current research interests include data analytics, cybersecurity, deception and cognition, information transparency, and math theory.

Kim-Kwang Raymond Choo (SM'15) received the PhD degree in information security from the Queensland University of Technology, Australia, in 2006. He currently holds the Cloud Technology Endowed professorship with The University of Texas, San Antonio. He was named one of 10 Emerging Leaders in the Innovation category of The Weekend Australian Magazine / Microsoft's Next 100 series, and Cybersecurity Educator of the Year – APAC (produced in cooperation with the Information Security Community on LinkedIn) in 2009 and 2016, respectively. He is the recipient of various awards including ESORICS 2015 Best Research Paper Award, Highly Commended Award by Australia New Zealand Policing Advisory Agency, Fulbright Scholarship in 2009, 2008 Australia Day Achievement Medallion, the British Computer Society's Wilkes Award for the best (sole-authored) paper published in the 2007 volume of The Computer Journal, and ACISP 2005 Best Student Paper Award. He is also a Fellow of the Australian Computer Society.