LINCS: Towards Building a Trustworthy Litigation Hold Enabled Cloud Storage System

By

Shams Zawoad, Ragib Hasan and John Grimes

From the proceedings of

The Digital Forensic Research Conference

DFRWS 2015 USA

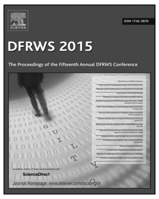
Philadelphia, PA (Aug 9th - 13th)

Digital Investigation 14 (2015) S55–S67

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin



DFRWS 2015 US

LINCS: Towards building a trustworthy litigation hold enabled cloud storage system

Shams Zawoad*, Ragib Hasan, John Grimes

University of Alabama at Birmingham, USA

Keywords:
Litigation hold
Cloud security
Cloud forensics
Storage security
Regulatory compliance

A B S T R A C T

Litigation holds are inevitable parts of modern civil lawsuits that mandate an organization to preserve all forms of documents related to a lawsuit. In current data storage models, this includes documents stored in clouds. However, due to the fundamental natures of today's clouds, incorporating a trustworthy litigation hold management system is very challenging. To make the situation more complicated, defendants or plaintiffs may collude with the cloud service provider (CSP) to manipulate the documents under the hold. Serious consequences can follow if a litigant party fails to comply with the litigation hold for evidence stored in the cloud, resulting in legal sanctions for spoliation. This will not only harm the reputation of an organization but also levy of sanctions, such as fines, penalties, etc.

In this paper, we define a model of trustworthy litigation hold management for cloud-based storage systems and identify the key security properties. Based on the model, we propose a trustworthy **LI**itigation hold e**N**abled **C**loud **S**torage (*LINCS*) system. We show that *LINCS* can provide the required security properties in a strong adversarial scenario, where a plaintiff or defendant colludes with a malicious CSP. Our prototype implementation reveals that the performance overhead of using *LINCS* is very low (average 1.4% for the user), which suggests that such litigation hold enabled storage system can be integrated with real clouds.

Introduction

Litigation has become an unavoidable part of doing business. Nearly 90% of US companies are engaged in some sort of litigation (FULBRIGHT, 2005). A *litigation hold* is a notice to an organization to preserve all the electronically stored information (ESI) for a current lawsuit for a certain time period. The court can charge litigant parties for evidence spoliation, i.e., deliberate loss, modification, or destruction of evidence during the litigation hold period, if the proof of any such alteration is presented (FRCP, 2006b; Araiza, 2011). There are numerous cases, where a defendant failed to adhere to the litigation hold and was sanctioned for spoliation (FRD, 2003; Dist. Court, SD Ohio, 2014; K&L Gates, 2012). Most often, the defendant has to pay the court fees or fines as high as several hundred thousand dollars (Dist. Court, SD Ohio, 2014; K&L Gates, 2012).

With the emergence of cloud computing, consumers are moving towards the cloud for their storage needs. According to Gartner, consumers will store more than one third of their digital content in the cloud by 2016 (Gartner, 2012). Because of the large scale migration to the cloud-based storage and computation services, a massive amount of evidence required for a lawsuit are now stored in the cloud. Some incidents of storing contraband documents in cloud-based storage systems have already been reported (BBC, 2013; Dist. Court, SD Texas, 2014). Evidence residing

* Corresponding author.
E-mail addresses: zawoad@uab.edu (S. Zawoad), ragib@uab.edu (R. Hasan), jwgrimes@uab.edu (J. Grimes).

http://dx.doi.org/10.1016/j.diin.2015.05.014
1742-2876/© 2015 The Authors. Published by Elsevier Ltd on behalf of DFRWS. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

on clouds has great impact on legal rules and regulations, especially for litigation holds (Araiza, 2011; Smith, 2012; Dykstra and Riehl, 2012; Nicholson, 2012).

Unfortunately, traditional notions of data possession do not always easily apply to the storage model of a cloud. In a traditional computing model, a party usually has physical possession of most of its ESI. But in clouds, the ESI is generally under the physical control of the cloud service provider (CSP). Litigants using cloud services risk spoliation because of the CSP's physical control over their data (Pham, 2013). Moreover, a defendant can collude with the CSP to remove evidence without keeping any trace of the evidence destruction. A plaintiff can also collude with the CSP to remove ESI from a defendant's cloud storage, in order to falsely accuse the defendant for evidence spoliation. Failing to preserve evidence not only prevents a party from adequately proving or defending a claim at trial but can also cost massive amount of money (K&L Gates, 2012). Recent court cases include two incidents (Dist. Court, SD Ohio, 2014; Dist. Court, SD Texas, 2014), where the problem of litigation hold in clouds has been addressed.

To address the problem of litigation hold management in clouds, Schmidt proposed to build a legal hold framework in clouds (Schmidt, 2012). However, the trustworthy management of litigation holds was not addressed in this work, especially when defendants and plaintiffs can collude with the CSP. In the existing provable data possession schemes in clouds (Ateniese et al., 2007; Erway et al., 2009], users (in this case, the defendants) are responsible to create the metadata and the data possessions can be verified by users or a third-party. However, in the problem domain of this work, the end user/the defendant can be dishonest and does not want to preserve any proof of data possession to hide an act of spoliation. A dishonest defendant can simply avoid sending metadata to a third-party so that any act of spoliation will be untraceable. Since a cloud can be accessible from anywhere, we cannot force defendants to send verification metadata to the auditor; they can always find a new device to upload files to clouds without sending the metadata to the third party. The existing secure logging schemes (Accorsi, 2006; Holt, 2006; Zawoad et al., 2013) also do not consider the threat model that we consider here, especially the collusion between a dishonest CSP, defendants, and plaintiffs.

In this paper, we address the challenges of establishing trustworthy litigation hold management systems in clouds and propose *LINCS* – a litigation hold enabled cloud storage framework. *LINCS* provides secure verifiable proof of the preservation of a litigation hold during the holding period. Unlike the existing work on litigation holds management in clouds (Schmidt, 2012), we do not consider the CSP as honest. We consider that a defendant or a plaintiff can collude with a malicious CSP. We propose two protocols to upload and delete a file in a litigation hold enabled cloud storage to preserve the proof of file creations and deletions securely. We follow the forward-integrity approach (Holt, 2006) and publish the proofs periodically to ensure the integrity of the proofs. Performance of the proposed system is measured on an Amazon EC2 based cloud storage. We develop a tool for auditors to determine any incident of spoliation and identify any fake evidence using the proofs.

Implementing *LINCS* in real clouds can help to ensure that all relevant documents are retained during the litigation hold period and can confirm any incident of spoliation. Moreover, even if the CSP is honest, the security properties ensured by *LINCS* can help CSPs to establish trust with the cloud users. By ensuring trustworthy litigation hold management, *LINCS* can make clouds more compliant with regulatory compliance, such as Sarbanes-Oxley (SOX) (Congress of the United States, 2002), which requires trustworthy data retention. Integrating the *LINCS* framework can have advantage on the financial aspects of business organizations. By avoiding a false accusation of evidence spoliation, defendants can save significant amount of money as well as their business reputation. Cloud providers can also attract more customers with the assurance of providing trustworthy litigation hold management.

Contribution: The contributions of this work are as follows:

1. To the best of the authors' knowledge, this is the first work to address the trustworthy litigation hold management problem in the context of cloud computing. We systematically analyze the threats on litigation hold management in clouds and present a novel threat model, which can provide future research directions in this area.
2. We present a litigation hold enable cloud storage framework – *LINCS* and show that it provides the required security properties for a trustworthy management of litigation holds in the cloud.
3. We evaluate the feasibility of the *LINCS* framework by developing a prototype of the framework. Our results suggest a very low performance overhead (average 1.4% for the user) for integrating the proposed framework.

Organization: The rest of the paper is organized as follows: Section 2 provides the background knowledge about litigation holds. In Section 3, we present the litigation hold model in clouds and the threat model. In Section 4, we present the *LINCS* framework. Section 5 discusses the security analysis of *LINCS*. In Section 6, we evaluate the performance of the proposed framework and present a verification tool for auditors. Section 7 presents the related work and finally, we conclude in Section 8.

Background

Litigation hold and spoliation

A *litigation hold* is a legal notice to a defendant that triggers the preservation of ESI, which may require to terminate the routine operation of an information system to suspend the normal destruction of ESI (Araiza, 2011). Litigation holds are also known as *preservation letters* or *stop destruction requests* (Stacy, 2014). The preservation obligation may arise from many sources, including common law, statutes, regulations, or a court order. FRCP Rule 37 implies that ordinary data retention and cleaning policies should not be applied to ESI under a litigation hold (FRCP, 2006b).

After a defendant is given notice of a litigation hold, the destruction of information available from reasonably accessible sources is considered to be *spoliation* (FRCP, 2006b). Hence, spoliation is the deliberate or inadvertent loss, modification, or destruction of evidence by a party on notice of litigation (Araiza, 2011). It is the responsibility of the defendant to produce proof of preservation of litigation hold. Similarly, if the plaintiff accuses the defendant for spoliation, then the plaintiff has to provide the evidence of spoliation. Two recent incidents of spoliation have been observed in *US Equal Employment Opportunity Commission (EEOC) v. JP Morgan Chase* and *EEOC v. Ventura Corp.*, where the defendants were sanctioned for failing to preserve employment records that were important evidence for their respective cases (Zapproved, 2013).

Litigation hold in clouds

A major difference for litigation holds on cloud-based ESI for a public cloud environment is that a defendant's data is now under the direct control of a third party – the CSP (Rashbaum et al., 2014). FRCP 34(a)(1) states about the preservation of evidence, which are under the control or possession of the defendants (FRCP, 2006a). However, there are various cases, where a third party was involved to store ESI, but the court determined that the defendant was still in possession or control of the ESI, since the defendant had or should have had the ability to obtain the requested data from the third-party (Dist. Court, ED Michigan, 2008; Dist. Court, SD New York, 1999; Court of Appeals, 10th Circuit, 2011). Though the cloud-based ESI is under the possession of the CSP, relevant ESI within the possession or control of a third party may be obtained by serving a subpoena upon the third party, including a CSP (InfoLawGroup LLP, 2010). Moreover, the CSP may be the subject of a civil subpoena, government agency demand, or a governmental subpoena directly (Rashbaum et al., 2014). Hence, if a case involves evidence stored in clouds, customers and CSPs share a mutual need and duty to ensure the litigation hold on cloud evidence.

Case Study. The *Quantlab Technologies Ltd. V. Godlevsky* case is an excellent recent example, where the court imposed litigation hold on cloud evidence (Dist. Court, SD Texas, 2014). In this case, plaintiffs brought suit against defendants for copyright infringement, breach of contract, misappropriation of trade secrets, and fraud. Quantlab alleged that Kuharsky, a previous employee of Quantlab, used their intellectual property including code for high volume trade information management system. The plaintiffs claimed that Kuharsky spoliated evidence from different sources including cloud-based storage during the lawsuit. Kuharsky said that he might have stored information on the cloud, including the code. He said that one account may have been on the Amazon Cloud and admitted that the accounts expired years ago. It was not possible for the court to establish either defendants' or plaintiffs' claims of evidence destruction, as today's clouds do not ensure litigation holds for cloud-based ESI nor provide proof of spoliation.

In *Brown v. Tellermate Holdings Ltd*, the court also addressed the need of appropriate measures to provide proof of litigation hold in the cloud, without which a litigation hold process might not be defensible (Dist. Court, SD Ohio, 2014). In this case, the defendant failed to preserve sales records of the plaintiff, which were stored in Salesforce.com – one of the largest Software-as-a-Service (SaaS) providers.

Modeling trustworthy litigation hold in clouds

Litigation hold model

We present the litigation hold model in Fig. 1. In this model, ΔL is the time when the litigation hold is issued. T_e is the time, when the litigation hold will end. Hence, the litigation hold remains active during $(T_e - T_s)$ period, which we refer as ΔL. According to the definition of litigation hold, during the ΔL period, the CSP or the defendant cannot remove any files that are stored in the defendant's cloud storage. Hence, as shown in Fig. 1, file deletion during ΔL is considered as spoliation. T_c is the time, when the CSP colludes with a defendant or a plaintiff and turns malicious. Hence, $(T_c - T_s)$ is the time period when the CSP was honest during the litigation hold period, which is referred as ΔH, where $\Delta H \geq 0$. ΔM is the time period, when the CSP was malicious, where $\Delta M = (T_e - T_c)$.

According to the definition of litigation hold, users are free to remove any files before T_s and a defendant will not be charged with spoliation for this action. Hence, file deletion before T_s is not considered as spoliation, which is presented as safe deletion in Fig. 1. A file modification operation is treated as creating a new file. A defendant can also add new files to the cloud storage during the ΔL period. However, a defendant usually does not include a new suspicious file in the storage when the storage is already under a litigation hold. Hence, we do not consider such files for litigation hold management.

If a plaintiff accuses a defendant for evidence spoliation, the plaintiff needs to present adequate evidence to the court that can prove the incident of file deletion by the defendant during the ΔL period. On the other hand, when accused with spoliation, the defendant needs to provide the proof of file preservation during the ΔL period. Based

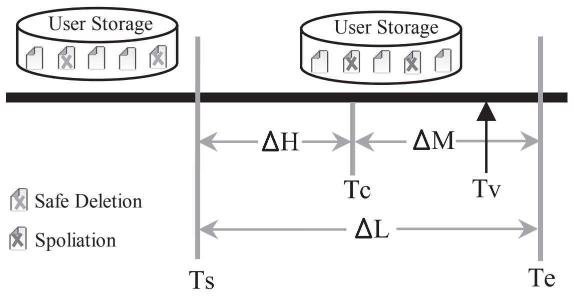


Fig. 1. Litigation hold model.

upon the evidence provided by the litigant parties, the court can verify the preservation of litigation hold at T_v, where T_v can be any time between ΔL. A plaintiff can also be a cloud user in a separate case and the defendant can be the plaintiff for that case. In this scenario, the role of the defendant and the plaintiff will be just the opposite. For the new case, the plaintiff will be treated as the defendant and the defendant will be treated as the plaintiff. Below we describe the important terms of the system.

- *Cloud Service Provider (CSP):* The CSP in its entirety or an employee of the CSP can be malicious and collude with a defendant or a plaintiff. We assume that the CSP turns malicious after the litigation hold is issued. The incentive for being malicious can have monetary value. Moreover, cloud providers may use another third party for some services. Those third party providers can also be malicious. Any bug in the CSP's storage system or malware in the storage server can also cause unwanted file deletions.
- *Defendant:* A defendant is a cloud user, whose cloud-based storage is under litigation hold for an active lawsuit and he/she can collude with the CSP.
- *Plaintiff:* A plaintiff brings a case against the defendant and can collude with the CSP to falsely accuse a defendant for evidence spoliation.
- *Proof of File (PF):* The *PF* preserves the proof of every file of a user that resides in the user's cloud storage.
- *Proof of Deletion (PD):* The *PD* preserves the proof of any file deletion.
- *Auditor:* An auditor is the court authority, who verifies whether the litigation hold was maintained or not till T_v.

Threat model

We consider that the CSP can be dishonest and collude with a defendant or a plaintiff. Assets in this problem are files, which are under a litigation hold and need to be protected from illegal destruction. We consider the following attack scenarios to design the system.

Case 1. In this case, only the defendant is malicious and following attacks are possible.

- After receiving a litigation hold, i.e., after T_S, a malicious defendant can remove some incriminating files from the cloud storage by avoiding the proof of deletion preservation system. Later, the defendant can provide incomplete evidence (excluding the deleted files) to the plaintiff.
- The defendant can deny the ownership of a file presented by the plaintiff at the trial or deny the deletion of a file.

Case 2. In this case, the defendant colludes with the CSP and following four attacks are possible.

- The defendant and the CSP together bypass the proof of deletion preservation system. Hence, there will be no traces of file deletions during the $FCM_U{}^i$ period.
- Besides avoiding the proof of deletion preservation system, a colluding CSP can also remove the proof of file (PF) and act as if the file was never created by the defendant.
- Defendants can collude with the CSP to prove an act of spoliation as a safe deletion operation, i.e., a defendant removes a file after T_s but the malicious CSP alters the proof of deletion and records the deletion as it occurred before T_s. The CSP can also change the system time of the file server and set the time anytime before T_s to represent the deletion as a safe deletion.
- The defendant and the malicious CSP can deny hosting a file, which is presented to the court by the plaintiff.

Case 3. In this scenario, the defendant is honest but a malicious plaintiff colludes with the CSP to frame the defendant for spoliation. Following attacks are possible in this case.

- The malicious CSP, colluding with a plaintiff can remove a file from the defendant's cloud storage without the defendant's consent. Later, the plaintiff can present the proof of deletion of the file to accuse an honest defendant for evidence spoliation.
- A CSP colluding with a plaintiff can alter the proof of file deletion to present a safe deletion operation by the defendant as an act of spoliation.
- A plaintiff can also collude with a CSP to plant a back-dated fake file to the defendant's cloud storage without the defendant's consent. Later, the defendant will not be able to produce that fake file to the court since he/she was not aware of the file. This will give an opportunity to the malicious plaintiff to wrongly accuse the defendant for evidence spoliation.

Case 4. The proofs of files or the proofs of file deletions can also be targets for external attackers. An attacker can learn the content of a file from the proof of the file or the proof of deletion.

Security properties

Considering the aforementioned attack scenarios, we argue that a secure litigation hold management system should ensure the following security properties.

I1 A defendant cannot deny the ownership of a file if it is actually created by the defendant, which is referred as nonrepudiation in the legal and records management field of study.
I2 A defendant cannot deny a proof of file deletion if the file is genuinely deleted by the defendant.
I3 If a file is removed before T_s, then the proof of this deletion cannot be reordered by any adversary to place it after T_s.
I4 If a file is deleted after T_s, the proof of this deletion cannot be reordered by any adversary to place it before T_s.

- I5 If a malicious defendant whether acting alone or colluding with a CSP removes a file during ΔL and the plaintiff presents a copy of that file to the court, then the auditor must be able to detect the incident of spoliation.
- I6 A malicious plaintiff, colluding with a CSP cannot delete a file from a defendant's cloud storage and prove the deletion as an act of spoliation to the auditor.
- I7 A malicious CSP, colluding with a plaintiff cannot add a fake backdated file in a defendant's storage without being detected by the auditor.

C1 External attackers including malicious insiders of the CSP cannot learn the content of a file from the proofs of files or the proofs of deletions.

Challenges

After a litigation hold is issued, an honest CSP can track all the document removal requests during the ΔL period and store the requested documents in a separate storage. This will ensure the preservation of all ESI along with the attempted spoliation. However, a CSP as a whole or a malicious employee of the CSP can collude with any of the party of a litigation and can produce incomplete evidence.

A CSP could prove their honesty by providing the storage device to investigators. The investigator can probe the device's unallocated slack space to reveal the deleted files and from the timestamp of the deleted files, spoliation can be identified easily. However, there are two fundamental problems with this traditional approach. *First*, because of the multi-tenant nature of clouds, CSPs cannot provide the storage device to investigators without violating the privacy of honest co-tenants. *Second*, cloud infrastructures are not centralized, ESI for one customer can be located at dispersed location, and it will be impossible to give access to the storage of all the data centers to investigators.

A CSP can also export all the documents that need to be on hold to a trusted third party (TTP). However, this scheme will also violate users' privacy. For example, if the third party gets access to trade secret information, which was stored in a cloud that could destroy the legal protection of trade secrets. Moreover, this type of TTP needs to accommodate massive amount of storage facility, in the worst case similar to CSPs, which may prove impractical and cost inefficient overtime.

When a litigation hold is issued, a defendant and plaintiff can copy all the data in a read-only medium to prevent spoliation and protect the integrity of the evidence. However, this scheme also raises some challenges. For example, it can be challenging to make sure that all the data get copied. The defendant can hide some files or the plaintiff can add fake files. Moreover, there are challenges of managing such a read-only storage. If a third party manages this storage, there will be the same issues that are addressed for the TTP-based scheme previously.

A defendant can also digitally sign all files at the time T_s, so that no deletion by anyone can occur without invalidating the signature. However, a malicious defendant may not sign some crucial documents and can remove those documents during the ΔL period without keeping any trace of spoliation. Having multiple backups of the storage can also help to determine an act of spoliation. Unfortunately, a malicious defendant can try to avoid using multiple backups to easily destroy the evidence. However, the redundancy can be useful for honest defendants and can help in case of a false accusation of spoliation.

Reliability of a litigation hold management system could be ensured by placing a trusted surveillance system in the defendant's devices to monitor communications between the devices and a cloud-based storage system. Unfortunately, since a cloud-based storage can be accessible from any device, the defendant can always find a new device in which the surveillance system is not running. Hence, such a surveillance mechanism is an incomplete solution, which can provide a dishonest defendant an opportunity of removing a file from the cloud storage without leaving any trace behind.

The LINCS framework

Overview

We propose that in a litigation hold enabled cloud storage system, a user will generate a File Creation Metadata FCM_U for each file and send the file along with the FCM_U to the CSP. Whenever the Litigation Hold Manager (LHM) module of the cloud storage receives a file, it generates its own file creation metadata FCM_C. The LHM creates forward-secured Proof of File PF from the FCM_U and the FCM_C and attaches it with the file as a metadata. Likewise, for each file deletion operation, the user and CSP generate File Deletion Request FDR and File Deletion Acknowledgment FDA respectively. These metadata are encapsulated as Proof of Deletions PD. After some certain epoch, the PF and PD will be published to the Internet to make these forward-secured, i.e., if the system get compromised at time t, no entity can manipulate the metadata and the proofs published before time t.

Notation and Assumptions:$H(M)$ is a collision resistant and one-way hash function, which produces a hash of a message M (Penard and Werkhoven, 2008). The $Sig_{S_K}(M)$ function generates a signature of a message M using the secret key S_K (Bellare and Rogaway, 1996). $Mac_K(M)$ is the MAC (message authentication code) generation function to produce a MAC of a message M using the key K. $MKey(S)$ is the function to produce a secret key for MAC generation from a given secret S. For concatenation of two or more attributes, we use the symbol '|'. A tuple is encapsulated in '$< >$'. We assume that the defendant and the CSP have setup their secret keys and public keys and distributed the public keys. The secret key and the public key of the user are S_{KU} and P_{KU} and for the CSP, these are S_{KC} and P_{KC}. We discuss the construction of the proposed framework below.

File upload in LINCS

The workflow for uploading a new file in *LINCS* is depicted in Fig. 2 and we discuss the details below.

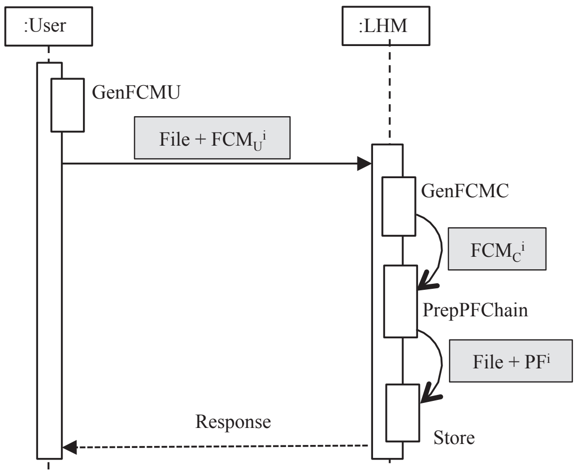


Fig. 2. File upload workflow.

- A user first executes the $\mathrm{GenFCMU}(\mathrm{S_{KU}}, \mathrm{CT_U}, \mathrm{F^i}) \rightarrow \{\mathrm{FCM_U}{}^i\}$ algorithm to produce the file creation metadata $FCM_U{}^i$ for the F^i file. To prepare the $FCM_U{}^i$, the algorithm requires the secret key of user S_{KU}, current timestamp of the user CT_U, and the F^i file that the user will send to clouds. The file creation metadata $FCM_U{}^i$ is constructed as follows:

$$FCM_U{}^i = <\left(H(F^i)\middle|F_{ID}{}^i\middle|CT_U\right), Sig_{S_{KU}}\left(H(F^i)\middle|F_{ID}{}^i\middle|CT_U\right)> \tag{1}$$

Here, $F_{ID}{}^i$ is an alphanumeric unique identity of a file, which is generated by the algorithm. The user then attaches the $FCM_U{}^i$ with the file F^i as a user-defined file creation metadata and sends the file to the CSP.

- After receiving a file from the user, the LHM module of the cloud storage system will run $\mathrm{GenFCMC}(\mathrm{S_{KC}}, \mathrm{CT_C}, \mathrm{F^i}, \mathrm{F_{ID}}{}^i) \rightarrow \{\mathrm{FCM_C}{}^i\}$ algorithm to produce the file creation metadata $FCM_C{}^i$ for the F^i file. As arguments, the algorithm takes the secret key S_{KC}, current timestamp of the cloud storage server CT_C, the F^i file, and its identity $F_{ID}{}^i$. The algorithm constructs the file creation metadata $FCM_C{}^i$ as follows:

$$FCM_C{}^i = <\left(H(F^i)\middle|F_{ID}{}^i\middle|CT_C\right), Sig_{S_{KC}}\left(H(F^i)\middle|F_{ID}{}^i\middle|CT_C\right)> \tag{2}$$

- In this step, the LHM module preserves the proof of file PF in a forward secure way using the algorithm $\mathrm{PrepPFChain}(\mathrm{FCM_U}{}^i, \mathrm{FCM_C}{}^i, \mathrm{MK_C}{}^{i-1}) \rightarrow \{\mathrm{PF^i}, \mathrm{MK_C}{}^i\}$. The algorithm requires the $FCM_U{}^i$ and $FCM_C{}^i$ of the i^{th} file, and the key to generate the PF for the $(i-1)^{th}$ file $MK_C{}^{i-1}$. The algorithm outputs PF^i as the proof of creation of the i^{th} file.

The key to produce the MAC for the i^{th} file $MK_C{}^i$ is generated as follows:

$$MK_C{}^i = <MKey\left(H\left(MK_C{}^{i-1}\right)\right)>, where\ MK_C{}^0 = <MKey(H(S_C))> \tag{3}$$

The S_C is stored in a secure server. After creating the new MAC generation key $MK_C{}^i$, the LHM module removes the previous MAC generation key $MK_C{}^i$. The algorithm then produces the PF^i as follows:

$$PF^i = <Mac_{MK_C{}^i}(CS^i), (CS^i)> \tag{4}$$

where $CS^i = <FCM_U{}^i \middle| FCM_C{}^i>$.

One can argue that PF^i can be included in the chain to prevent alteration of the file creation history. The chain is created using only the MAC key intentionally so that no entity can re-compute the chain without having the initial secret key. Including PF^i will increase the storage overhead but will not give extra security. Since the *PFs* are included as a metadata of the files, defendant will have the ability to verify that they share their belief of the file history with the CSP. Moreover, the *VerifyPFChain* algorithm, described in the Section 4.4 can identify any modification of the chain of *PF*.

- The LHM module attaches the PF^i with the F^i file as a metadata and stores the file in the cloud storage system. Additionally, the last *PF* of each day will be published to the Internet. The proofs can be available by RSS feeds to protect it from manipulation by the CSP after publishing. Users can subscribe to such RSS feeds to keep a copy of the proofs. The proofs can also be published in every hour or every minute depending on the required security level of the defendant's storage system. The task of proof publication can be encapsulated as a Cron job. Specific time and frequency of publishing *PF* can be controlled by the Cron job parameters.

File deletion in LINCS

File deletion in *LINCS* follows a special protocol (presented in Fig. 3) to preserve secure proof of file deletion. Details of this protocol are presented below.

- When a user wants to delete a file F^i from clouds, first the user runs $\mathrm{GenFDR}(\mathrm{sS_{KU}}, \mathrm{DT_U}, \mathrm{F^i}, \mathrm{F_{ID}}{}^i) \rightarrow \{\mathrm{FDR^i}\}$ algorithm to create the file deletion request FDR^i for the F^i file. To generate the FDR^i, the algorithm requires the secret key S_{KU}, current system time of user DT_U, the file F^i that the user wants to delete, and its identity $F_{ID}{}^i$. The file deletion request FDR^i is constructed as follows by the algorithm:

$$FDR^i = <\left(H(F^i)\middle|F_{ID}{}^i\middle|DT_U\right), Sig_{S_{KU}}\left(H(F^i)\middle|F_{ID}{}^i\middle|DT_U\right)> \tag{5}$$

The user then sends the FDR^i to the cloud as the file deletion request of the F^i file.

- After receiving a file deletion request from a user, the LHM module executes $\mathrm{GenFDA}(\mathrm{S_{KC}}, \mathrm{P_{KU}}, \mathrm{DT_C}, \mathrm{F^i}, \mathrm{F_{ID}}{}^i) \rightarrow \{\mathrm{FDA^i}\}$ algorithm to verify the file deletion request FDR^i for the F^i file and creates acknowledgement of the deletion request FDA^i. Arguments of this algorithm are the secret key of CSP S_{KC}, public key of the user P_{KU}, current timestamp of storage server DT_C, the file F^i that is requested by the user for deletion, and its identity $F_{ID}{}^i$.

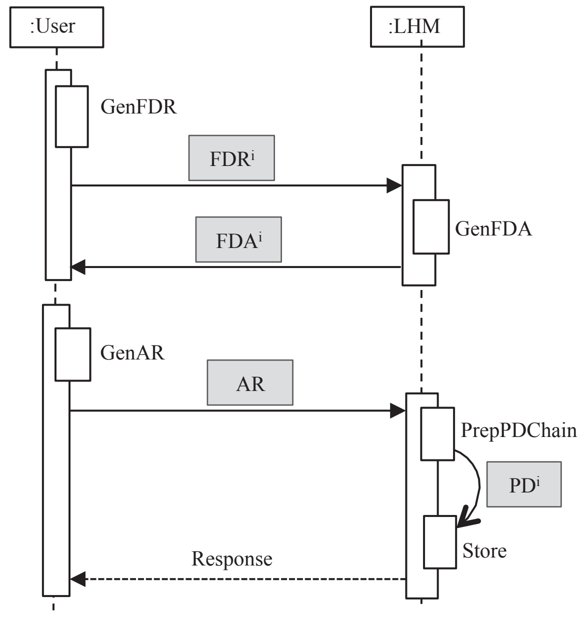


Fig. 3. File deletion workflow.

The LHM module first verifies the signature of the user, which is included in the FDR^i using the pubic key of the user P_{KU}. If the signature is valid, the LHM module generates the file deletion acknowledgement FDA^i as follows:

$$FDA^i = <\left(H(F^i)\middle|F_{ID}{}^i\middle|DT_C\right), Sig_{S_{KC}}\left(H(F^i)\middle|F_{ID}{}^i\middle|DT_C\right)> \quad (6)$$

This acknowledgement message is then sent to the user.

- After receiving the FDA^i from the CSP, the user generates acknowledgement receipt AR using GenAR($F_{ID}{}^i, S_{KU}, P_{KC}, FDA^i) \rightarrow \{AR\}$ algorithm. The algorithm requires the private key of the user S_{KU}, public key of the CSP P_{KC}, FDA^i, and the identity of the i^{th} file $F_{ID}{}^i$ to generate the AR. The algorithm first verifies the signature of the CSP in the FDA^i using the public key of the CSP P_{KC}. If the signature is valid, it will check the file hash of FDA^i with the file hash of FDR^i, and prepares the AR for the F^i file as follows,

$$AR = \langle\left(Response\middle|F_{ID}{}^i\middle|AT_U\right), Sig_{S_{KC}}\left(Response\middle|F_{ID}{}^i\middle|AT_U\right)\rangle \quad (7)$$

Here AT_U is the timestamp of the AR message creation and *Response* can be *true* or *false* based on the signature and file-hash verification results. The user finally sends the AR to the CSP.

- Upon receiving AR from the user for the F^i file, the LHM module removes the F^i file and stores the proof of deletion securely using algorithm PrepPDChain (FDR^i, FDA^i, $MK_D{}^{i-1}) \rightarrow \{PD^i, MK_D{}^i\}$. Inputs of this algorithm are FDR^i and FDA^i of the F^i file deletion operation, and the key to generate the PD for the $(i-1)^{th}$ file $MK_D{}^{i-1}$. The algorithm outputs PD^i as the proof of deletion of the F^i file.

The key to produce the MAC for the i^{th} file deletion $MK_D{}^i$ is generated as follows:

$$MK_D{}^i = \langle MKey\left(H\left(MK_D{}^{i-1}\right)\right)\rangle, MK_D{}^0 = \langle MKey(H(S_D))\rangle \quad (8)$$

The S_D is stored in a secure server. After creating the new MAC generation key, the LHM module deletes the previous MAC generation key $MK_D{}^{i-1}$ and produces PD^i as follows:

$$PD^i = \langle Mac_{MK_D{}^i}(DS^i), (DS)\rangle \quad (9)$$

where, $DS^i = <FDR^i|FDA^i|AR>$

The LHM module stores the PD^i in the proof of deletion database and returns the file deletion result to the user. After some certain epoch, the last PD will be published to the Internet.

Verification of litigation hold

From the proof of file and proof of deletion chain, an auditor can determine any incident of spoliation. There are two verification processes. The auditor checks the proof of deletion PD chain to determine any modification in the chain. The auditor also needs to check the validity of the chain of PF for a given set of files. The auditor determines any alleged incident of spoliation using these verification processes. We describe each of the verification algorithms below.

- VerifyPDChain($[PD^0, \ldots PD^v], P_{KU}, P_{KC}, S_D) \rightarrow$ {accept, reject} algorithm can detect any alteration of the PD chain. Inputs of this algorithm are: proof of deletion from the beginning to the verification time T_v, public keys of the user and the CSP, and the secret S_D. The algorithm either accepts or rejects the chain of PD. This verification algorithm is depicted in Fig. 4 and it works as follows.

For every PD^i, $i = 0$ to v, the algorithm first calculates the key for MAC generation using the Equation (8). Using the pubic key of the user and the CSP, it then verifies the signature of the components of the DS^i. If the signatures are valid, the algorithm generates MAC of DS^i using the previously created MAC key $MK_D{}^i$. When the generated $MAC_A(DS^i)$ is equal to the $MAC(DS^i)$ of the chain, the algorithm accepts PD^i as in the correct order. If the PD^i is the last proof of the day t, the algorithm compares the PD^i with published PD of the day t $PD_t{}^p$. The algorithm rejects the chain if these two are not equal.

- VerifyPFChain($[F^0, \ldots F^v], P_{KU}, P_{KC}, S_C) \rightarrow$ {accept, reject} verifies the chain of PF for a set of files $[F_0..F^v]$. Besides the set of files, this algorithm takes the public keys of the user and the CSP, and the secret S_C as arguments.

First, the set of files will be sorted in ascending order of the file creation time. After sorting the files, the verification algorithm works similar to the VerifyPDChain algorithm.

For every file F^i, where $i = 0$ to v, the algorithm calculates the key for MAC generation using the Equation (3). After extracting the PF^i from the header of the F^i file, the

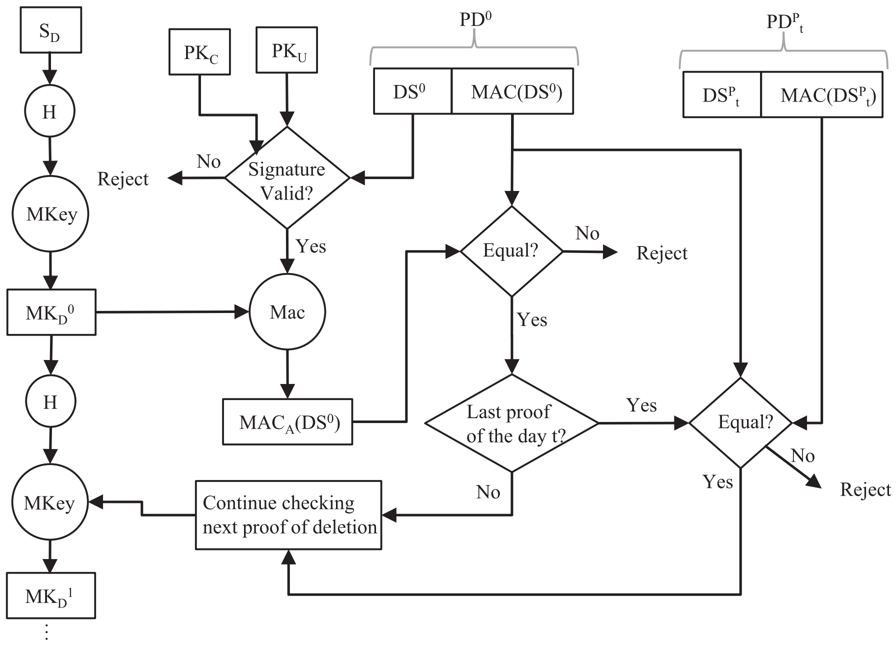


Fig. 4. Verification of the proof of deletion chain.

algorithm first compares the hash of the F^i file with the $H(F^i)$ component of the PF^i. Later, the algorithm verifies the signature of the components of the PF^i. If the signatures are valid, the algorithm generates MAC of CS^i using the previously created MAC key $MK_C{}^i$. If the generated $MAC_A(CS^i)$ is equal to the $MAC(CS^i)$ of the F^i file, the algorithm accepts the file F^i as in the correct order, otherwise it rejects the file. If the file F^i is the last file of the day t, then the algorithm compares PF^i with the published PF of the day t $PF_t{}^p$. The algorithm rejects the list of files if these two are not equal.

Security analysis

In this section, we discuss how *LINCS* provides the security properties mentioned in Section 3.3. The following lemmas are based on the assumption of the existence of a collision-resistant hash function, secure encryption, and MAC generation function.

Lemma 1. PF^i is the proof provided by the user and the CSP about the existence of the F^i file.

Lemma 2. $FCM_U{}^i$ is the proof provided by the user and the CSP about the deletion of the F^i file.

Lemma 3. The secret keys and the initial secrets S_D and S_C cannot be accessed by an adversary afterwards, but can be accessed by the auditor.

Lemma 4. Once a proof of file *PF* and a proof of deletion *PD* are published, the CSP cannot alter the proofs or deny the existence of the proofs.

Security propositions and proofs

Proposition 1. If a file F^i is actually created by a defendant, then the defendant cannot deny the possession of the F^i file.

Proof. According to the proposed file upload protocol, each file created by the user/defendant contains the proof of file *PF* with the file-header. Hence, for the F^i file, there will be a PF^i attached as a metadata of the file, which includes the $FCM_U{}^i$. There is a signature of the defendant with the $FCM_U{}^i$ and according to Lemma 3, no adversary has access to the secret key of the defendant. Hence, a defendant cannot deny the file F^i, when the file-header contains the defendant-signed $FCM_U{}^i$. Hence, the proposition 1 is true, which ensures the security property I1.

Proposition 2. If a file F^i is deleted by a defendant by following the proposed file deletion protocol, then the defendant cannot deny the proof of deletion PD^i for the F^i file.

Proof. The proposed file deletion protocol ensures mutual agreement between the defendant and the CSP to delete a file. The proof of deletion PD^i for the F^i file contains two components that are signed by the user: file deletion request FDR^i and acknowledgement receipt *AR*. The FDR^i ensures that the file deletion actually initiated by the defendant and the *AR* proves that the CSP and the defendant agreed upon the file deletion. Since the defendant signed these two components and according to Lemma 3, no adversary has access to the secret key of the defendant, the defendant cannot deny the proof of file deletion PD^i. Therefore, proposition 2 is true, which ensures the security property I2.

Proposition 3. If a file F^d is removed before T_s, then the proof of deletion of this file PD^d cannot be placed after T_s in the proof of deletion chain without being detected by an auditor.

Proof. Suppose the file F^d was actually removed before T_s and F^{d+1} is the file that was removed after the file F^d and F^{d-1} is the file that was removed before the file F^d. If an

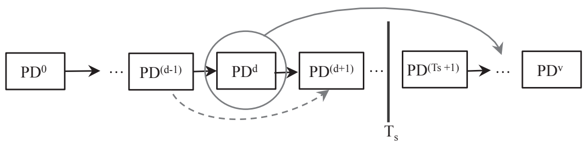


Fig. 5. An attempt of presenting a fake spoliation by placing PD^d after T_s

adversary removes the proof of deletion PD^d of the file F^d from its actual position and puts it after T_s (as illustrated in Fig. 5), then PD^{d+1} will be appeared after PD^{d-1} in the altered proof of chain. According to the VerifyPDChain algorithm, for i = 0 to i = (d − 1), the auditor generated $MAC_A(DS^i)$ and the CSP provided $MAC(DS^i)$ will be equal. At i = d, the auditor creates $MAC_A(DS^{d+1})$ from the MAC key $MK_D{}^d$. The newly generated $MAC_A(DS^{d+1})$ will not match with $MAC(DS^{d+1})$ because the $MAC(DS^{d+1})$ was created using $MK_D{}^{d+1}$.

According to Lemma 3, an adversary cannot access the initial secret S_D later and hence, cannot recreate the chain. However, an honest but curious CSP may store the MAC generation keys. Later, the CSP may turn malicious and can use the stored MK_Ds to alter the chain. Suppose, the file F^d is removed on day t and the CSP modified the chain starting from that day. In that case, the last proof of day t will not be similar to the published last proof of that day. This inconsistency will also be found for each day after the day t. According to Lemma 4, the CSP cannot modify or deny the published proof. Therefore, any modification in the chain of *PD* will be detected by an auditor. Therefore proposition 3 is true, which ensures the security property I3.

Proposition 4. If a file F^d is removed after T_s, then the proof of deletion of the file PD^d cannot be placed before T_s in the proof of deletion chain without being detected by an auditor.

Proof. Suppose an adversary wants to place the PD^d between PD^j and PD^{j+1} as presented in Fig. 6. The proof of deletion chain verification algorithm VerifyPDChain creates $MAC_A(DS^{j+1})$ using the key $MK_D{}^{j+1}$. However, the generated $MAC_A(DS^{j+1})$ will not match with the $MAC(DS^d)$ of the PD^d, since the $MAC(DS^d)$ was not calculated using the key $MK_D{}^{j+1}$ at the time of deleting the file F^d. Since the CSP cannot modify or deny any published proof (Lemma 4), they cannot recreate the chain starting from (j + 1) without being detected by the auditor. Therefore, PD^d cannot be placed after PD^j without being detected by the auditor. Hence, proposition 4 is true, which ensures the security property I4.

An honest but curious CSP can store the MAC generation keys and later may turn malicious. Such a malicious CSP can try to place PD^j before T_s and recompute the chain since the last publication. However, continuously publishing proofs,

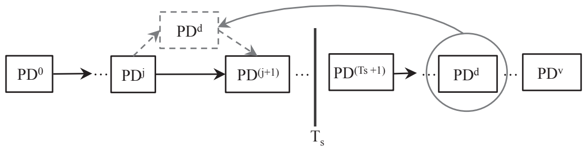


Fig. 6. An attempt of hiding spoliation by placing PD^d before Ts

such as via RSS feeds will narrow down the window of opportunity for manipulating the chain to zero file operations.

Proposition 5. If a defendant removes a file F^d during the ΔL period and the plaintiff presents the file F^d to the court, the auditor must be able to detect the act of spoliation.

Proof. When the CSP is honest, the deletion of the file F^d must follow the proposed protocol. Hence, when the defendant removed the file F^d, according to Lemma 2, there must be a proof of deletion PD^d and since proposition 2 is true, the defendant cannot deny the PD^d. If the deletion of the F^d file is an act of spoliation, the PD^d will appear after T_s. This can be identified using the VerifyPDChain algorithm.

A defendant can collude with a malicious CSP and can remove a file without following the proposed protocol. Hence, there will be no proof of deletion PD^d for the F^d file. However, according to Lemma 1, the PF^d can prove the existence of the file F^d in the defendant's cloud storage. Using the VerifyPFChain algorithm, the auditor can check whether the PF^d, attached with the file F^d is valid or not. After including the F^d file with the defendant-provided set of files, the VerifyPFChain algorithm will accept the whole set of files if the file F^d is valid.

The defendant can also collude with the CSP to remove the PF^d from the F^d file, before the file is acquired by the plaintiff. However, the VerifyPFChain algorithm can detect this removal. Suppose F^{d+1} is the file that was created after the F^d file. Since the file F^d is not provided to the auditor by the defendant, when i = d, the VerifyPFChain algorithm generates $MAC_A(CS^{d+1})$ using the key $MK_C{}^d$. The algorithm then compares the $MAC_A(CS^{d+1})$ with the $MAC(CS^{d+1})$ of the F^{d+1} file, which are not equal because the $MAC(CS^{d+1})$ was created using the $MK_C{}^{d+1}$ key. Even if an honest but curious CSP stores the MAC generation keys, the malicious CSP cannot modify the chain starting from F^{d+1} without being detected by the auditor since the last proof of each day is already published. Therefore, the proposition 5 is true, which ensures the security property I5.

Proposition 6. If a file F^d is removed without the defendant's consent, the plaintiff cannot prove this deletion as an act of spoliation.

Proof. All the file deletions in LINCS storage should follow the proposed protocol when the CSP is honest. However, a malicious CSP can collude with a dishonest plaintiff and remove the F^d file by avoiding the proposed protocol. The plaintiff still needs to present the proof of deletion PD^d to the court, which should contain the defendant's signature with the FDR^d and the *AR*. According to Lemma 3, no adversary can access the secret key of the defendant to spoof the signature. Hence, if the malicious CSP adds a fake PD^d, the auditor can detect the invalid signature of the defendant in the VerifyPDChain algorithm and will reject the proof. Hence, the proposition 6 is true, which ensures the security property I6.

Proposition 7. A malicious CSP cannot add a fake file F^f to the defendant's storage without being detected by the auditor.

Proof. The file F^f should have the proof of file PF^f in the file header. The PF^f includes $FCM_U{}^f$, which is signed by the defendant. According to Lemma 3, the secret key of the defendant cannot be accessed by an adversary. Hence, if the CSP add a fake FCM_U^f with the PF^f, the VerifyPFChain algorithm can detect this anomaly while verifying the signature. Moreover, presenting the F^f file as a backdated file requires modification in the chain of *PF*. Any modification in the chain can be detected by the auditor using the VerifyPFChain algorithm. Therefore, the file upload protocol ensures that a malicious CSP colluding with a plaintiff cannot add the fake F^f file in the defendant's storage without being detected by the auditor. Hence, the proposition 7 is true, which ensure the security property I7.

Proposition 8. An adversary cannot identify the content of the file F^i from the PF^i or PD^i.

Proof. The proof of file PF^i and the proof of deletion PD^i are created from the hash of the F^i file. Because of the one-way, collusion resistant hash function, it is not possible for an adversary to reverse engineer the proofs and extract the content of the F^i file. Therefore, the proposition 8 is true, which ensures the C1 property.

Implementation and evaluation

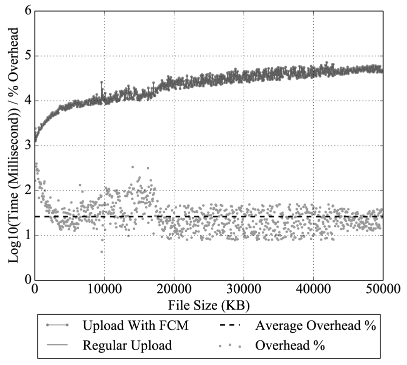
Incorporating *LINCS* can introduce overhead for users as well as the CSP. In this section, we first analyze the overhead from users' and CSP's perspective. Later, we present the performance analysis of PrepPFChain and PrepPDChain algorithms. Finally, we present the litigation hold verification tool and performance analysis of the VerifyPFChain algorithm.

System Configuration. As a prototype of cloud-based storage, we set up a ftp server in an AmazonEC2 medium (m1.medium) instance running the Ubuntu 12.04.4 LTS operating system. The litigation holds manager (LHM) module was running inside the EC2 instance. However, in a real-life implementation, the LHM module will not be deployed in an EC2 instance; it will be a part of the storage management module of the cloud infrastructure. Performance of user and auditor modules are tested on a Dell laptop running Debian 3.2.46-1 on Intel Core 2 Duo CPU (2.66 GHz) with 4 GB of RAM and the hard disk drive's capacity was 500 GB. We used Oracle JDK (version 1.7.0_51) to implement the modules of *LINCS*The LHM module uses the PostgreSQL 9.1.13 database system to store the proofs of deletion. Java Crypto and Apache Commons library were used for key generation, encryption, decryption, hash, and MAC generation. We used *RSA (2048 bit)* for encryption, *SHA-256* hash function for hashing, and *HMACSHA1* for MAC generation.

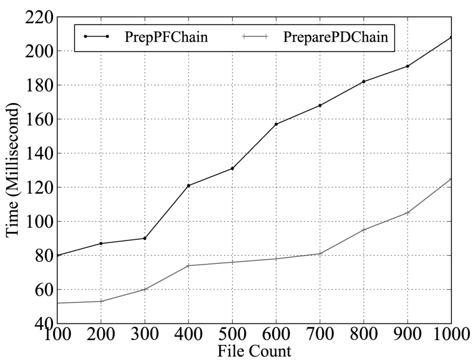
Overhead for uploading files. To measure the overhead of uploading files using our proposed scheme, we first upload 1000 files to the AmazonEC2-based cloud storage and measure the time to upload. The file size was uniformly distributed between 50 KB and 50,000 KB. Later, for each of the files, we generate the file creation metadata FCM_U, attach the FCM_U with the file, upload the file to cloud storage again, and measure the time of this whole process. Fig. 7a illustrates the experimental result. The overhead of uploading files varies between 0.6 and 3.4%, and the average overhead is 1.4%. From Fig. 7a, we observe that there is a spike between the 10 MB–20 MB range. Analyzing the upload time and FCM_U creation time separately indicates that the spike was due to the network bandwidth; it is not related with the FCM_U creation.

Storage overhead for the CSP: After integrating the LHM module, the CSP needs to store some extra information because of the metadata and can also experience lower performance due to the additional work. Using the aforementioned cryptographic properties in Java, we measured that each FCM_U and FCM_C takes 434 bytes, where the current time information requires 13 bytes, hash of file is 64 bytes, signature is 344 bytes, an alphanumeric ID of 10 bytes as the F_{ID}, and three information separator of 1 byte each. Size of MAC is 28 bytes. Hence, for one file, we need 896 bytes ($434 * 2 + 28$) of additional information. For a file of size [2, 4, 8, 16, 32] MB, the associated overhead is [0.0445, 0.022, 0.011, 0.006, 0.002]%.

Performance analysis of proof creation. We measured the performance of two algorithm PrepPFChain and PrePDChain, which is illustrated in Fig. 7b. For each of the file uploaded by a defendant, we measured the performance analysis of these two algorithms. We used the same set files, which we used to measure the overhead of file



(a) File Upload Overhead in *LINCS*



(b) Time Required for PrepPFChain and PrepPDChain Algorithms

Fig. 7. Performance analysis.

uploading. Fig. 7b depicts that the required time increases linearly with number of files. With the uniformly distributed 1000 files, we measured that the average time to complete PrepPFChain algorithm for each file is 1.5 ms and for PrepPDChain, it is 0.3 ms.

Litigation hold verification tool

We developed a verification tool for auditors to determine spoliation and fake evidence provided by a litigant party. The main control panel of this tool is presented in Fig. 8. The auditor can input the files received from the litigant parties, in zip format, proof of deletion (collected from the CSP) in csv format, and the litigation hold period. Defendants retrieve the files from their cloud storage. Plaintiffs can collect the files from the CSP, defendants, or a insider of the defendant. Some of the files, provided by the plaintiff may fall into the safe deletion category, whether others may fall into spoliation. Some of the files may not be even created/deleted by the defendant. The verification result panel shows these three sets of files including the reason for spoliation or rejection. An auditor can view the details of every item of these three categories, which includes information about the file owner, creation or deletion time, metadata, signature, and chronological order verification results.

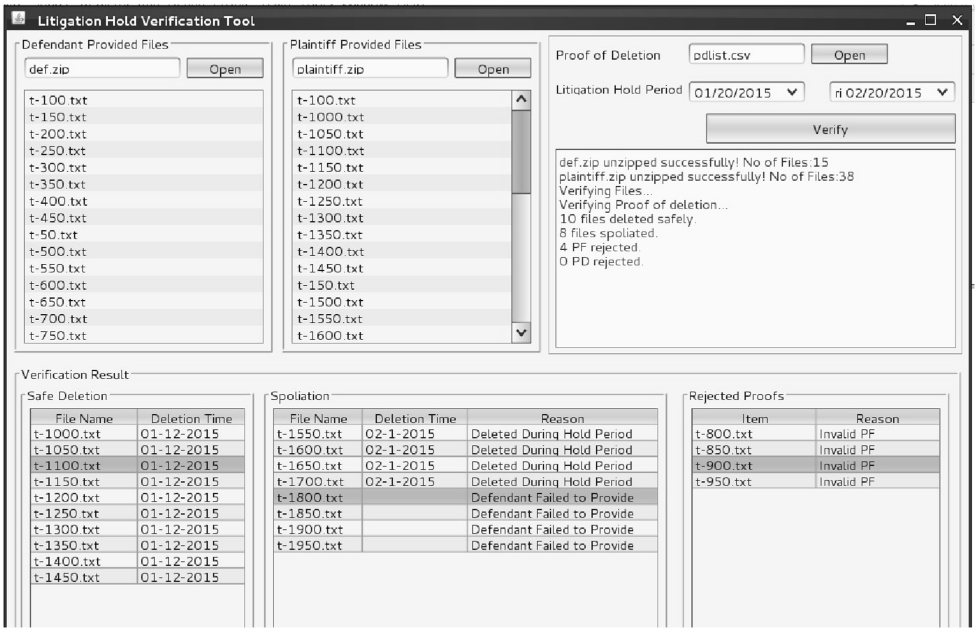
Performance analysis of proof verification. We measured the performance of the file verification procedure (VerifyPFChain algorithm) for two criteria: total number of files and total size of files. In the first experiment, we verified different number of files starting from 100 to 1000 with 100 intervals, where the total size of the files was always 5 GB. Results of this experiment are presented in Fig. 9a, from where we notice that verification time increases linearly with the number of files. An auditor can verify 1000 files of total 5 GB size in approximately 2.5 min Fig. 9b illustrates the results of our second experiment, where there were 100 files in each run and total size of those 100 files were 2, 4, 8, 16, 32, and 64 GB. The results indicate that the verification time increases linearly with the file size.

Related work

Researchers of law enforcement area have addressed the problem of maintaining litigation holds in the cloud (Araiza, 2011; Nicholson, 2012; Katz, 2009). Araiza et al. suggested to automatically isolate specified ESI and associated metadata when there is a litigation hold (Araiza, 2011). Researchers suggested that the service level agreement (SLA) should ensure the deactivation of routine destruction of ESI once a litigation hold is triggered (Smith, 2012; Pham, 2013).

Hasan et al. formalized the litigation hold model for database transaction and proposed a framework for trustworthy vacuuming that ensures retention of data in question (Hasan and Winslett, 2010). Mitra et al. proposed efficient schemes for secure management of inverted index entries for a write-once read-many (WORM) compliance storage device to support litigation hold (Mitra et al., 2008a, b). Borisov et al. proposed an encrypted index to allow litigation hold for expired document in WORM storage and restricted queries on the keyword index (Borisov and Mitra, 2008). However, the threat model that we proposed here is novel. These schemes did not address the problem of trustworthy litigation management for cloud-based storage.

Ateniese et al. first defined the provable data possession (PDP) model for ensuring possession of static files on an untrusted storage (Ateniese et al., 2007). They utilized RSA-



Defendant Provided Files: def.zip

```
t-100.txt
t-150.txt
t-200.txt
t-250.txt
t-300.txt
t-350.txt
t-400.txt
t-450.txt
t-50.txt
t-500.txt
t-550.txt
t-600.txt
t-650.txt
t-700.txt
t-750.txt
```

Plaintiff Provided Files: plaintiff.zip

```
t-100.txt
t-1000.txt
t-1050.txt
t-1100.txt
t-1150.txt
t-1200.txt
t-1250.txt
t-1300.txt
t-1350.txt
t-1400.txt
t-1450.txt
t-150.txt
t-1500.txt
t-1550.txt
t-1600.txt
```

Proof of Deletion: pdlist.csv

Litigation Hold Period: 01/20/2015 ri 02/20/2015

```
def.zip unzipped successfully! No of Files:15
plaintiff.zip unzipped successfully! No of Files:38
Verifying Files...
Verifying Proof of deletion...
10 files deleted safely.
8 files spoliated.
4 PF rejected.
0 PD rejected.
```

Safe Deletion

File Name	Deletion Time
t-1000.txt	01-12-2015
t-1050.txt	01-12-2015
t-1100.txt	01-12-2015
t-1150.txt	01-12-2015
t-1200.txt	01-12-2015
t-1250.txt	01-12-2015
t-1300.txt	01-12-2015
t-1350.txt	01-12-2015
t-1400.txt	01-12-2015
t-1450.txt	01-12-2015

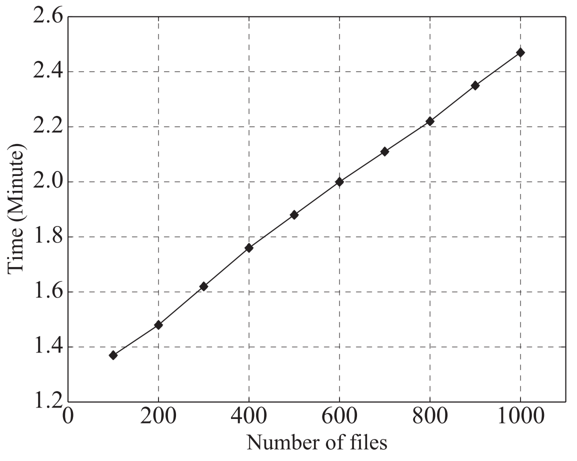
Spoliation

File Name	Deletion Time	Reason
t-1550.txt	02-1-2015	Deleted During Hold Period
t-1600.txt	02-1-2015	Deleted During Hold Period
t-1650.txt	02-1-2015	Deleted During Hold Period
t-1700.txt	02-1-2015	Deleted During Hold Period
t-1800.txt		Defendant Failed to Provide
t-1850.txt		Defendant Failed to Provide
t-1900.txt		Defendant Failed to Provide
t-1950.txt		Defendant Failed to Provide

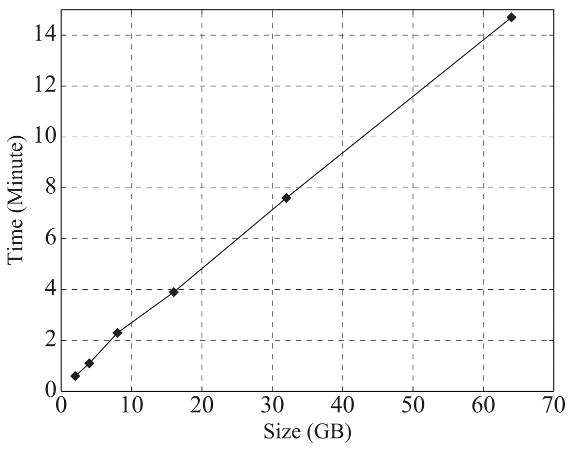
Rejected Proofs

Item	Reason
t-800.txt	Invalid PF
t-850.txt	Invalid PF
t-900.txt	Invalid PF
t-950.txt	Invalid PF

Fig. 8. Verification tool for auditor.



(a) Verification time vs. the number of files



(b) Verification time vs. the total file size

Fig. 9. Performance analysis of verification procedure.

based homomorphic tags for auditing outsourced data and provide public verifiability. Later, they extended the scheme to support dynamic data (Ateniese et al., 2008). Erway et al. extend the PDP model to support provable updates to stored files using rank-based authenticated skip lists (Erway et al., 2009). If a dishonest CSP removes a file without the client's consent, the client can detect this malicious behavior using a PDP scheme. However, these schemes require the client to generate the metadata as the PDP. Therefore, these schemes cannot solve the specific problem that we address in this paper, especially when a client is dishonest and does not want to preserve any PDP.

The closest work related to managing litigation hold in clouds was proposed (Schmidt, 2012), where the author proposed to build a legal hold framework in clouds. The framework receives legal hold information indicating a legal hold applicable to modification or deletion of a document. After receiving the legal-hold information, the framework updates a legal-hold index with an identifier for the document and updates legal-hold metadata with the identity of the legal action. However, this patent did not focus on trustworthy management of litigation hold to protect hold from dishonest CSP, defendant, or plaintiff.

Conclusion and future work

In recent years, spoliation of ESI has assumed critical importance in civil litigation, which warrants very careful attention to technical solutions to the problem. Proving whether spoliation has been occurred for cloud-based evidence is very challenging due to the fundamental natures of clouds. Collusion between different parties makes the problem more complicated. In this paper, we analyzed the legal rules for litigation holds and based on that, defined a model of trustworthy litigation hold in clouds. Using this model, we proposed *LINCS* that ensures the required security properties of a trustworthy litigation hold enabled cloud storage. Our prototype implementation suggests that such scheme can be implemented with low system overhead.

This paper is the first step towards a systematic analysis of the litigation hold problem in the cloud and provides a secure and efficient solution. We did not consider the file deletion before T_S, dynamic behavior of the cloud storage, and new files creation after T_S while designing the proposed solution. Moreover, trustworthy management of metadata needs to be handled to ensure the security of the special types of files, such as emails, which are usually stored in files using standard email formats, such as MBOX or EML. We will address these issues in the future and also investigate the case of merging *LINCS* with existing PDP schemes. We plan to integrate the LHM module with the object storage and block storage modules of OpenStack – an open source cloud platform.

Acknowledgment

We would like to thank the anonymous reviewers and the shepherd for their valuable comments to improve the paper. This research was supported by the National Science Foundation CAREER Award #CNS-1351038, the Department of Homeland Security Grant #FA8750-12-2-0254, and Sigma Xi Grants-in-Aid of Research #G201503151198201.

References

Accorsi R. On the relationship of privacy and secure remote logging in dynamic systems. In: Security and privacy in dynamic environments, vol. 201. US: Springer; 2006. p. 329–39.

Araiza AG. Electronic discovery in the cloud. Duke L. & Tech; 2011. Rev., 1.

Ateniese G, Burns R, Curtmola R, Herring J, Kissner L, Peterson Z, et al. Provable data possession at untrusted stores. In: 14th ACM conference on computer and communications security. ACM; 2007. p. 598–609.

Ateniese G, Di Pietro R, Mancini LV, Tsudik G. Scalable and efficient provable data possession. In: 4th international conference on security and privacy in communication networks. ACM; 2008. p. 9–18.

BBC. Lostprophets' Ian Watkins: 'Tech savvy' web haul. December 2013. http://www.bbc.com/news/uk-wales-25435751.

Bellare M, Rogaway P. The exact security of digital signatures-how to sign with rsa and rabin. In: Proceedings of advances in cryptology, EUROCRYPT. Springer; 1996. p. 399–416.

Borisov N, Mitra S. Restricted queries over an encrypted index with applications to regulatory compliance. In: Applied cryptography and network security. Springer; 2008. p. 373–91.

Congress of the United States. Sarbanes-Oxley act. 2002. https://www.congress.gov/107/bills/hr3763/BILLS-107hr3763enr.pdf [accessed 11.04.15].
Court of Appeals, 10th Circuit. Tomlinson v. el paso corp. F. 3d, vol. 653; 2011. p. 1281. no. 10–1385.
Dist Court, ED Michigan. Flagg v. city of detroit. 2008.
Dist Court, SD New York. Dietrich v. bauer. F. Supp. 2d, vol. 76; 1999. p. 312. No. 95 Civ. 7051 (RWS).
Dist Court, SD Ohio. Brown v. Tellermate holdings ltd; 2014a. Case No. 2: 11-cv-1122.
Dist Court, SD Texas. Quantlab technologies ltd. v. godlevsky. 2014. Civil Action No. 4: 09-cv-4039.
Dykstra J, Riehl D. Forensic collection of electronic evidence from infrastructure-as-a-service cloud computing. Rich JL Tech 2012;19:1.
Erway C, Küpçü A, Papamanthou C, Tamassia R. Dynamic provable data possession. In: 16th ACM conference on computer and communications security. ACM; 2009. p. 213–22.
FRCP. Rule 34. 2006. https://www.law.cornell.edu/rules/frcp/rule_34.
FRCP. Rule 37. 2006. https://www.law.cornell.edu/rules/frcp/rule_37.
FRD. Zubulake v. UBS Warburg LLC; 2003.
FULBRIGHT. Second annual litigation trends survey. 2005. http://www.fulbright.com/mediaroom/files/fj0536-us-v13.pdf.
Gartner. Gartner says that consumers will store more than a third of their digital content in the cloud by 2016. 2012. http://www.gartner.com/newsroom/id/2060215.
Hasan R, Winslett M. Trustworthy vacuuming and litigation holds in long-term high-integrity records retention. In: 13th International conference on extending database technology. ACM; 2010. p. 621–32.
Holt JE. Logcrypt: forward security and public verification for secure audit logs. In: 2006 Australasian workshops on grid computing and e-research, vol. 54. Australian Computer Society, Inc.; 2006. p. 203–11.
InfoLawGroup LLP. Legal implications of cloud computing part 4.5. 2010. http://goo.gl/rzJe2e.
Katz L. Balancing act: ethical dilemmas in retaining e-discovery consultants. Geo J Leg Ethics 2009;22:929.
K and L Gates. For spoliation, court holds defendant in contempt, orders $600,000 to be paid to plaintiff, $25,000 to be paid to the court. 2012. http://goo.gl/W7Ph7b.
Mitra S, Winslett M, Borisov N. Deleting index entries from compliance storage. In: 11th International conference on extending database technology: advances in database technology. ACM; 2008a. p. 109–20.
Mitra S, Winslett M, Hsu WW, Chang KC-C. Trustworthy keyword search for compliance storage. VLDB J Int J Very Larg Data Bases 2008b;17(2):225–42.
Nicholson JA. Plus ultra: third-party preservation in a cloud computing paradigm. Hastings Bus LJ 2012;8:191.
Penard W, Werkhoven T. On the secure hash algorithm family. 2008. Available at: http://www.staff.science.uu.nl/~werkh108/docs/study/Y5_07_08/infocry/project/Cryp08.pdf.
Pham C. E-discovery in the cloud era: what's a litigant to do. Hastings Sci Tech LJ 2013;5:139.
Rashbaum KN, Borden BB, Beaumont TH. Outrun the lions: a practical framework for analysis of legal issues in the evolution of cloud computing. Ave Maria L Rev 2014;12:71–149.
Schmidt, O., 2012. Managing a Legal hold on cloud documents. US Patent App. 13/543,254.
Smith J. Electronic discovery: the challenges of reaching into the cloud. St Clara L Rev 2012;52:1561.
Stacy S. Litigation holds: ten tips in ten minutes. 2014. http://www.ned.uscourts.gov/internetDocs/cle/2010-07/LitigationHoldTopTen.pdf.
Zapproved. Eeoc discrimination cases lead to serious sanctions for failures to issue litigation holds. 2013. http://goo.gl/r4kzAr.
Zawoad S, Dutta AK, Hasan R. SecLaaS: secure logging-as-a-service for cloud forensics. In: 8th ACM symposium on information, computer and communications security (ASIACCS). ACM; 2013. p. 219–30.