LiveForen: Ensuring Live Forensic Integrity in the Cloud

Anyi Liu, *Senior Member, IEEE,* Huirong Fu, *Member, IEEE,* Yuan Hong, *Senior Member, IEEE,* Jigang Liu, *Member, IEEE,* and Yingjiu Li, *Member, IEEE*

***Abstract*—To expedite the forensic investigation process in the cloud, excessive and yet volatile data needs to be acquired, transmitted and analyzed in a timely manner. A common assumption for most existing forensic systems is that credible data can always be collected from a cloud infrastructure, which might be susceptible to various exploits. In this paper, we present the design, implementation, and evaluation of LiveForen, a system that enforces a trustworthy forensic data acquisition and transmission process in the cloud, whose computer platforms' integrity has been verified. To fulfill this objective, we propose two secure protocols that verify the fingerprints of the computer platforms, as well as the attributes of the human agents, by taking advantage of the trusted platform module (TPM) and the attribute-based encryption (ABE). To transmit forensic data as a data stream and verify its integrity at the same time, a unique fragile watermark is embedded into the data stream without altering the data itself. The watermark allows not only the data integrity to be verified, but also any malicious data manipulation to be localized, with minimum communication overhead. The experimental results demonstrate that LiveForen achieves good scalability and limited performance overhead for authentication, data transmission, and integrity verification in an Infrastructure as a Service (IaaS) cloud environment.**

***Index Terms*—Cybersecurity; Cybercrime; Trusted computing; Trusted platform module; Attribute-based encryption; Fragile watermark.**

I. Introduction

THE proliferation of cloud computing has gained great attention as a high-performance and low-cost computing paradigm, which creates a new frontline for cybercrime investigations with some challenges. On one hand, due to its characteristics of elasticity and low cost, the cloud has been increasingly abused by adversaries. The advanced cybercrimes, such as VM-based malware [1], dark web [2], rogue cloud [3], cybercrime-as-a-service [4] can be easily launched from the guest VM. The "*pay-as-you-go*" pricing model allows adversaries to launch DDoS attacks [5], host command and control (C&C) server [6], and facilitate the ecosystem for cybercriminals [7], [8] on the fly and eliminate the trace evidence after the usage. On the other hand, federal agencies, including NIST, categorized the "*incident first responders*" (e.g., trustworthiness of cloud providers) and "*role management*" (e.g., data owners, identity management, users, and access control.) as the primary challenges of cloud forensics [9]. The most recent legislation of the Clarifying Lawful Overseas Use of Data Act (CLOUD Act) [10] allows law enforcement to acquire warrant-requested data from the servers regardless of their locations, which amplifies the opportunity of live forensics. Nevertheless, numerous forensic misapplications occurred in "*O. J. Simpson's case*", such as the discredited lab equipment, the unauthorized human agent, and the missing evidence during the transmission [11], are comparable to the technical challenges in the cloud, which also motivate this research. The proper resolution to these challenges will allow the forensic investigator to respond quickly to cybercrimes and collect credible forensic data[1] from a trustworthy infrastructure, which guarantees the confidentiality and integrity of evidence, as well as the credibility of the human agents.

Moreover, a forensic system could be overwhelmed by excessive data generated by the Virtual Machine (VM) and the widely deployed endpoints. According to the reports of the U.S. Department of Homeland Security (DHS), the total amount of data of forensic cases has grown 68.8 times (from 82.3 TB to 5,667 TB) from 2003 to 2016 [12]. It is uncommon for the cloud provider to store the suspect's volatile data, such as disk images and memory layout, on the mass storage before a warrant is issued. Furthermore, even though the virtual machine introspection (VMI) [13] has demonstrated its effectiveness in pulling the in-guest OS information to the outside hypervisor or virtual machine monitor (VMM), the acquired evidence can still be tampered with, if the integrity of the hypervisor, firmware, and network cannot be guaranteed. The chain-of-custody [9] could be broken at any missing link of a whole forensic process.

Existing approaches support evidence acquisition from the cloud through the existing tools [14]–[17]. A common weakness of these approaches is that they typically assume that both cloud infrastructure and human agents are trustworthy. They also assume that the data confidentiality can be well protected by the security protocols, such as SSL/TLS during the data transmission process. Unfortunately, both assumptions become increasingly unrealistic due to at least three technical challenges. First, it is non-trivial to verify the integrity of the entire trusted computing base (TCB) that facilitates the forensic process. The OS, firmware, and supporting libraries can be compromised by the malware, trojan horse, or rootkits. Thus, to collect the

A. Liu and H. Fu are with the Department of Computer Science and Engineering, Oakland University, 2200 N. Squirrel Road Rochester, MI 48309, USA.
E-mail: {anyiliu, fu}@oakland.edu
Y. Hong is with the Department of Computer Science, Illinois Institute of Technology, 10 W 31st Street, Chicago, IL 60616, USA.
E-mail: yuan.hong@iit.edu
J. Liu is with the Department of Computer Science and Cybersecurity, Metropolitan State University, St. Paul, MN 55106, USA.
Email: Jigang.Liu@metrostate.edu
Y. Li is with the School of Information Systems, Singapore Management University, 81 Victoria St, 188065, Singapore.
E-mail: yjli@smu.edu.sg

[1]We use *forensic data* and *evidence* interchangeably in the rest of the paper.

evidence from a platform whose TCB has not been fully verified is risky [18]. Second, the authentication of a trustworthy human agent is difficult. Since the human agent has the privilege to control the data, a malicious but privileged agent can easily tamper with the data without any restriction [15]. When a data acquisition job is delegated to a human agent without meeting the forensic criteria, the integrity of the chain-of-custody is difficult to ensure. Third, although the evidence can be collected from a trustworthy platform, it is still susceptible to the malicious manipulation by the man-in-the-middle (MITM) attacks during the transmission process. For instance, an adversary can partially tamper with a data stream, and thus force the entire evidence stream to be retransmitted. Existing data integrity verification techniques, including the Checksum, Message Authentication Codes (MAC), and Digital Signature, are designed to verify the data integrity as a monolithic piece and are incapable of localizing any manipulation in a data stream. For live forensics, it is crucial and yet challenging to know the integrity of the evidence, as well as to localize any data manipulation. Any one of these challenges might defeat the forensic investigation process.

In this paper, we present a novel framework that secures evidence acquisition and transmission in the IaaS cloud environment. This paper aims to overcome the following four research challenges (RCs) and thus makes contributions accordingly:

- **RC1**: Constructing a TCB that guarantees the integrity of the cloud infrastructure, on which a trustworthy relationship between parties can be established and maintained. We overcome this challenge by leveraging the commodity security hardware (e.g., TPM) as the building block to support both security-critical functionalities and data integrity. Thus, it is resilient to the subversion of malicious privileged domains, OS, and firmware. Moreover, the time-consuming crypto operations are minimized to reduce performance overhead.
- **RC2**: Authenticating the human agents and enforcing the control policy to access data in the cloud. To overcome this challenge, our approach enforces access control policies in the design of the secure protocols. The correctness of the secure protocols has been formally proven and experimentally verified by two formal security protocol verifiers.
- **RC3**: Verifying the integrity of evidence on the fly and keeping the integrity information accountable. Our approach overcomes this challenge by embedding the integrity information into streaming data groups as the the fragile watermark. In such a way, any malicious data manipulation, such as insertion, deletion, and modification, can be detected and localized to the particular data groups during the real time. The verifiable integrity information will be written back to the TPM, which allows forensic data acquisition to be resumed upon future request.
- **RC4**: Developing a feasible prototype that addresses the common technical issues of a live forensic process. To do that, we first compare our work with the prior solutions in the field. Then, we present a prototype system, *Live Integration Verifiable Environment for Cloud Forensics* (LiveForen), which is deployed in the actual cloud infrastructure. The evaluation results show that LiveForen can achieve low overhead for attestation, data integrity verification, and good scalability in the IaaS cloud environment.

In contrast to our previous work [19], [20], which proposed a framework that describes the functionalities of the key components with limited technical details, we present a comprehensive coverage, more technical details, and security analysis and proofs in this paper. The major extensions include: i) a comprehensive coverage of the attack vectors and a thorough comparison of our approach with the state-of-the-art in the field of live cloud forensics (Sections III-A); ii) an updated and detailed system architecture that authenticates the infrastructure and exchanges the credentials at run time (Section IV-A and Section IV-B); iii) the security analysis of the proposed protocols (Section IV-C and a part of Section IV-E); iv) a theoretic proof of the false negative rate of the data transmission scheme (Section IV-D); v) an empirical proof of the security protocols with two popular protocol verifiers (Section VI-A); vi) the updated algorithms that support data integration verification (Sections IV-D); and vii) a new implementation and new evaluation results on runtime overhead, integrity verification, and scalability (Section VI-B, VI-C, and VI-D).

The rest of the paper is organized as follows. Section II reviews the related work. Then, Section III elaborates the possible exploits and the threat model. Section IV first presents system design and then details the key components of LiveForen. Section V describes the implementation details. Section VI shows evaluation results. Section VII discusses limitation and possible improvements. Finally, Section VIII concludes the paper.

II. Related Work

Cloud Forensics Even though commodity forensic tools, such as EnCase [17] and FTK [16], have demonstrated their effectiveness in acquiring, recovering, and carving the evidence from PC and mobile devices, some challenges of cloud forensics still remain unsolved. Although most of them [14], [21]–[24] agree that the volatile data of a guest VM must be acquired by a trusted administrator from the VMM layer [25] or via the VMI technology, they commonly leave the technical solution, such as how to establish the root of trust (RoT), to the cloud provider. For instance, Dykstra et al. [25] presented a tool, namely FROST, which leverages the built-in APIs of OpenStack to collect the evidence from a cloud platform. Grispos et al. [26] proposed the solution of seizing the residual artifacts that have been accessed by a cloud service from a smartphone. Zawoad et al. [15] proposed the proofs of past data possession (PPDP), which preserves as a secure and nonvolatile proof of data ownership in an untrusted cloud. The common weakness of the prior work is obvious: they do not explicitly address the technical solution of how to construct the TCB of all parties involved in a forensic process, and simply entrust the cloud administrator to acquire the data. Unfortunately, researchers have demonstrated that once the TCB is breached by the

compromised hardware [27], infected firmware [1], or even the malicious cloud administrators [18], there is no guarantee of the data integrity.

Trusted Execution Environments To protect sensitive data from being compromised, a number of systems leverage security hardware to secure cloud computing. Flicker [28], vTPM [29], Trusted TVEM [30], Excalibur [31], Nexus [32], SSC [33], and CloudVisor [34] have demonstrated their effectiveness in constructing the root of trust and authentication with strong cryptographic support. Some of them use TPM as a secure foundation for online data access control [35] and cross-device attestation [36]. Unfortunately, these approaches cast little light on the threat model and security needs of live forensics. For instance, the evidence can be tampered with when it is under the control of the privileged but malicious human agent who owns the secrete key. The encrypted data can still be breached during the transmission phase. Moreover, live forensics requires evidence to be resumed when it is needed. Finally, because the physical TPM could become the computational bottleneck, the number of computationally expensive TPM-related operations should be limited.

In a broader sense, our work is related to the Trusted Execution Environments (TEEs) technologies, which have been widely adopted in commodity systems for enhancing system security. This category of technologies focuses on constructing a hardware-assisted isolated execution environment, within which the code and data of the software are isolated from the hostile environment (e.g., malicious hypervisor or firmware). Commodity technologies include but are not limited to: Trusted Execution Technology (TXT) [37] and Intel Software Guard Extensions (SGX) [38]. However, most of these well-designed technologies still suffer from some obvious limitations: First, the TEEs cannot guarantee that the program that runs within is bug-free. Any software vulnerabilities in the program could still subvert the entire system, despite how secure the TEEs are. Second, with the proliferation of TEE, the software increases in complexity and code base, and thus inevitably increases the attack surface. Finally, the security features of TEE can exaggerate the adversary's capability to develop advanced attack and malware, which makes analysis and detection difficult.

III. The Attack Vectors and Threat Model

A. The Attack Vectors

While there is a lack of consensus definition and taxonomy of the security issues for cloud forensic, Zawoad et al. [23], Ab Rahman et al. [24], and Dykstra et al. [14] comprehensively surveyed the security challenges of cloud forensics, which can be roughly classified into three categories, namely *multi-tenancy*, *user control*, and *multi-location*. Their current solutions are presented in Table II [2]. Our survey shows that our approach comprehensively addresses the challenges in these categories. The comparisons of our work with prior work are discussed in Section VII-A.

[2]In this paper, we mainly focus on the technical issues listed in the literature, and leave the non-technical and legal issues, such as the robust SLA, the cross-border law, for our future work.

To comply with this classification, we classify the attack vectors that can be defeated by the LiveForen into three categories: *malicious infrastructure* (C1), *malicious users* (C2), and *weak error detection and accountability* (C3). In particular, the attack vectors in C1 (E1 - E3) are related to the untrusted cloud nodes, from which data are collected, sent, and received. The attack vectors in C2 (E4 - E6) are related to the malicious users who compromise data confidentiality and integrity during a live forensic process. The attack vectors in C3 (E7 and E8) are related to the weak capability of error detection and lacking accountability. The detailed description of the attack vectors and their affected layers defined by Dykstra el al.[3] are listed below.

- *Unattested hardware and software* (`E1`): One of the critical security concerns of a cloud user is that a cloud node might not run on the top of the attested BIOS, bootloader, and hypervisor [31]. The cutting-edge technologies, such as Intel SGX [38], allows a minimal TEE to be constructed at the application layer; however, a cloud infrastructure without secure bootstrap, still enables the adversary to access credential in the TEE with various attacks [39], [40].
- *Unprotected cryptographic environment* (`E2`): Even though data confidentiality can be properly protected by the cryptographic keys, it does not prevent an adversary from running arbitrary code to subvert the OS, hypervisor, and firmware, from which the secrete keys can be stolen or leaked [41]. The crypto-keys are also susceptible to the alternation and leakage at run time [39], [40].
- *Time of Check to Time of Use (TOCTOU) attack* (`E3`): A race condition attack [42] can be triggered if the trustworthiness of a forensic software runtime environment is first attested from a trustworthy node, in which the fingerprints of hardware and software have been verified, and then be migrated to an untrusted node. This attack can either be triggered by the automatic agentless migration services [43], [44] or purposely by the adversary who intends to perturb the timing of live migration [45].
- *Malicious cloud administrator* (`E4`): A malicious cloud administrator with root privilege can access the guest VM images even if they are encrypted. The integrity of guest VM could also be compromised because the VM image can revert to a previous vulnerable version and be purged with the audit logs [15]. A malicious administrator can first authenticate data acquisition job with a privilege and trustworthy user and then delegate it to an untrusted user, whose privilege does not conform to the access control policy.
- *Malicious cloud user* (`E5`). Both the malicious data sender and receiver can deny the provision or possession of the forensic evidence. The accountability can be defeated if the non-repudiatable proof of past procession has not been kept on both sides [9].
- *Man-in-the-middle (MITM) attack* (`E6`): An MITM attack can easily intercept the secrets transmitted between the

[3]According the definition of Dykstra et al. [14], a trust model of the IaaS cloud forensics consists of six layers: Layer 1 (Network), Layer 2 (Physical hardware), Layer 3 (Host OS), Layer 4 (Virtualization), Layer 5 (Guest OS), Layer 6 (Guest Application). We only focus on the trustworthiness of Layers 1 - 4, while the customer retains the control over Layers 5 and 6.

cloud provider and cloud users. Recent research claims that the malicious cloud administrator can redirect the connection to a decoy VM, which intercepts the traffic between the allocated VM and the cloud user [18].

- *Force-to-retransmit attack* (E7): Although the confidentiality and integrity of forensic data can be ensured by the security protocol (e.g., Transport Layer Security) during data transmission, it is always desirable to detect the location of the data corruption on the fly, rather than verify data integrity as a monolithic piece when the transmission is complete. An adversary can deliberately corrupt a data fragment and cause a corrupted checksum of the whole dataset. If a data corruption cannot be precisely limited to the affected portion, the entire dataset might be forced to be retransmitted and thus waste network bandwidth.
- *Deniable attack* (E8): As a key requirement of live cloud forensics, evidence might be acquired whenever it is generated in the cloud, which avoids the complication of recovering the volatile data [22]. Thus, a live cloud forensic process should be resumed promptly when new evidence is generated. However, without proper authentication, the malicious cloud administrator may still send arbitrary data, rather than resuming from the previous breakpoint of the session.

Besides the attack vectors mentioned above, we also admit the possibility that the target VM is still susceptible to brute-force attack or run time subversion. However, its corresponding data is considered unbreachable if the TCB has been attested to be secure. From the forensic perspective, the validity of the data, which might contain the trace of a security breach, is the primary concern for the data acquisition process after a security breach has occurred.

B. The Threat Model

Our threat model is intended to address attack vectors mentioned in the previous subsection. To specify our threat model without loss of generality, however, we assume that the security of LiveForen should be built upon three prerequisites without the hardware attacks. First, although the TPM is vulnerable to hardware attacks, such as TPM rollback attack [46], CPU bus hijack attack [47], system rebooting attack [48], and malicious SMM handler attack, we assume that the physical security chips, including the TPM of the cloud node, is secured. This assumption implies that the secure keys, non-volatile RAM (NVRAM), and Platform Configuration Registers (PCRs) are unbreachable. Second, we assume that the crypto-keys, hashes, and the pseudorandom numbers generated by the security chips are unbreachable by using collision attack or brute-force attack. Nevertheless, it is cryptographically impossible to override or roll-back the PCR values of a physical TPM unless the trusted code of LiveForen is loaded and measured from a trusted reboot sequence. Third, since the private part of the crypto keys are nonmigratable and protected by the physical TPM, it is impossible for an adversary to emulate TPM by software, spoof the attestation result, or sign data without owning the valid private keys.

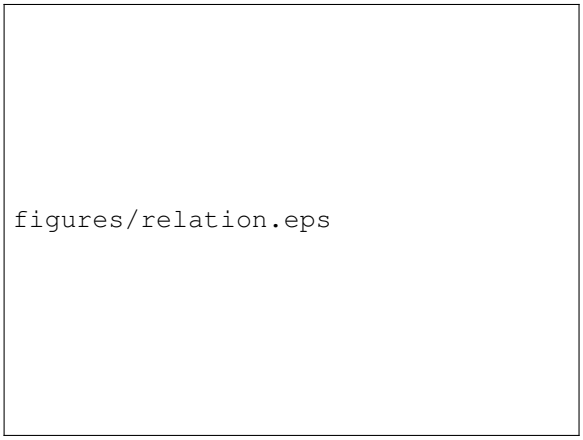


Fig. 1. The relationship between entities.

When a cloud forensic investigation process is conducted automatically, *cloud provider* refers to the entity that provides infrastructure, facilitates cloud services, and protects the customer's data for her own benefits. *Cloud administrator* refers to the person who manages cloud services and maintains the cloud infrastructure. In our system, *forensic provider* is specified to provide forensic infrastructure, services, and secure storage to the digital evidence.

LiveForen assuems that a cloud forensic investigation process is conducted automatically between two entities, namely the *forensic provider* (FP) and the *cloud provider* (CP). For cloud provider or forensic provider, we split their jobs into two dimensions: the *infrastructure* and the *operator*. Thus, the two dimensions of FP are specified as the *forensic infrastructure* (FI) and the *forensic operator* (FO). Similarly, the two dimensions of CP are specified as the *cloud infrastructure* (CI) and the *cloud operator* (CO). The relationship between the entities and their dimensions are illustrated in Figure 1. Recall that the infrastructure must be equipped an unbreachable TPM to keep the crypto key secure and to provide cryptographic guarantees about the software stack executing on the machine. On the other hand, the operator of a service provider can act as the *data sender* or the *data receiver*. An operator could be malicious, she has the access to dom0 and the privileges that it entails. In addition, although TPM hardware provides security features for attesting the hypervisor and managing keys, the crypto keys and data may potentially be abused by the operator because she has a privilege to use a root key (e.g., storage root key (SRK) of TPM) of the host.

We assume that the cloud provider has signed service-level agreements (SLAs) with the user and the forensic provider that enforce various access control policies, based on an operator's role attributes. Both the data sender and the receiver have no prior knowledge of each other, except for the certificates obtained from the privacy certificate authority (PCA). A certificate contains the PCR values that imply the valid fingerprint of a node, the public keys (including AIK and CP-ABE key). To speed up certificate lookup, a manifest file M is required that associates the attributes with the certificate. Both the sender and the receiver maintain a store of certificates of all potential data

custodians. The notations used in the remainder of the paper are listed in Table I.

TABLE I. Notations used in this paper.

Notation	Definition
$PCR[S_{PCR}]_n$	The PCR of the selection S_{PCR} for node n.
n_i	The ith nonce.
AIK_n^+, AIK_n^-	The public and private attestation identity keys (AIK).
BK_n^+, BK_n^-	The public and private keys that are bounded with the particular PCRs.
$Attr_u$	The attributes of the role u.
sk	The symmetric session key.
$\{M\}_u^P$	The message M encrypted with policy P that satisfies role u's attributes.
$\{M\}_k$	The message M encrypted by key k.
$\{M\}_k^-$	The message M decrypted by key k.
$\langle M\rangle_u$	The message M signed by the role u.
$H(M)$	The hash of message M.
$[\![e_1, e_2, ..., e_i]\!]$	The structure that contains the elements $e_1, e_2, ..., e_i$.
$x\|y$	The concatenation of x and y.
G_i	The i_{th} data group in data stream.

IV. System Design

In this section, we first present the system design of LiveForen. We then describe the protocol used to attest the parties involved in data transmission and analyze its security properties. After that, we detail the procedure and algorithms that embed watermarks and verify the integrity of streaming data. Finally, we present an accountable scheme that allows newly generated evidence to be retransmitted with minimal TPM-related operations.

A. The System Architecture

As mentioned in Section III-A, LiveForen systematically addresses the challenges of the existing cloud forensic systems, facilitates live evidence acquisition with a guarantee of data confidentiality and integrity, and defeats against compromised nodes and malicious operators. Compared with the existing cloud-based forensic systems, LiveForen is designed to achieve at least three objectives: 1) ensure that the forensic evidence is collected from and destined to the secure systems, from which a trustworthy relationship is established with respect to the BIOS, master boot record (MBR), firmware, hypervisor, and the roles of operators; 2) ensure the confidentiality and integrity of forensic evidence when it is transmitted as a data stream and make sure the tampered data can be detected in real time; and 3) produce non-repudiated proof such that a forensic acquisition session can be resumed with minimal cost of authentication.

Figure 2 illustrates the system design of LiveForen, which comprises two major components: the *node authenticator* (NA) and the *data transmitter* (DT). The high-level functionality of the NA is to attest to the platform and exchange cryptographic credentials between the involved nodes. Specifically, it is responsible for 1) attesting the nodes involved in data acquisition and transmission; 2) authenticating the roles of operators; 3) saving and reading non-fungible cryptographic proof from NVRAM. The high-level functionality of the DT is to perform data transmission by integrating the credentials obtained by NA so that streaming data is partitioned and later verified as groups. Specifically, it is responsible for 1) determining groups in streaming data by using the exchanged credentials, 2) computing group integrity signature (GIS) of each group and embedding it into a group, and 3) verifying the integrity of data in real time on the recipient's side. The detailed design of both components will be presented in the following subsections.

Fig. 2. The system architecture.

To bootstrap a live cloud forensic process, the node authenticator first receives a data acquisition request, which includes the attestation data, from the privileged VM (step 1). The request is forwarded to the physical TPM (step 2). Then, TPM validates both NA and DT and saves the crypto-keys and exchanged credentials in each of them (step 3a and 3b). After that, the response, which includes the TPM quote, crypto-keys, and the attributes of the operator, are sent to the data receiver (step 4). The detailed attestation protocol is covered in Section IV-B. In step 5, the NA authenticates the DT again and transfers the control to DT to access data (e.g., virtual disk, memory, and snapshots) from the cloud storage through the validated VMI interface (step 6a), and sends the data out. Meanwhile, data integrity information is embedded into streaming data as a watermark (step 6b). When the data transmission process is completed, the DT writes the integrity information back into the NVRAM of TPM (step 7), which is used to resume data transmission.

The system architecture of LiveForen and its workflow comprehensively addresses the attack vectors mentioned in Section III-A. In particular, step 3.a attests the trustworthiness of the TEE and thus prevents the attack vector of `E1` and `E2`. The attack vector `E3` can be blocked by steps 3.a and 5 sequentially, in which the former step allows the NA and DT to be authenticated by the genuine TPM while the latter step allows NA to re-authenticate DT before data transmission. A TOCTOU attack that either fails to receive a valid quote from the TPM or fail to show a valid credential at run time will fail the process. Furthermore, since any data retransmission requires a `TPM_NV_ReadValue` operation, which is also TPM-bounded, the TOCTOU attack launched at the retransmission phase will also fail if the data acquisition process is migrated to a malicious

TABLE II. The security challenges and solutions in the six-layer trust model of IaaS cloud forensics [14], [23]. Note that the challenges that are neither presented nor solved are marked as ✗. The challenges that are presented and solved are marked as ✓. The challenges are presented but not solved are marked as ✓✗.

Security categories	Challenges	Possible exploits	Layer(s) affected	References					Our Approach
				[26]	[15], [49]–[51]	[18], [31]	[14], [25]	[21], [22], [52]	
Malicious infrastructure (C1)	a) Lack of trust model	E1 - E3	L2 - L4	✗	✗	✓	✗	✓✗	✓
	b) Lack of data confidentiality and privacy			✗	✓	✓	✗	✓✗	✓
	c) Difficult to identify virtual artifacts			✗	✗	✗	✓	✓✗	✓
	d) Lack of forensic readiness			✗	✓	✓	✗	✓✗	✓
Malicious users (C2)	a) Lack of access control	E4 - E6	L2, L3	✗	✗	✓	✗	✓✗	✓
	b) Malicious insider			✗	✓	✓	✗	✓✗	✓
	c) Lack of specification of responsibility			✗	✓	✗	✓	✓✗	✓
Weak error detection and accountability (C3)	a) Lack of synchronization	E7 - E8	L1	✗	✗	✗	✗	✓✗	✓
	b) Lack of error detection			✗	✓	✗	✗	✓✗	✓
	c) Lack of logs			✗	✓	✗	✓	✓✗	✓

node. The attack vector E4 can be blocked by step 6.a because the virtual artifacts can only be retrieved from the virtual disk through the validated VMI interface and by the user whose role attributes satisfy the access control policy. Since the provision and/or possession of the forensic evidence is written back to the NVRAM in step 7, it serves as a non-repudiated proof and thus defeats attack vectors E5 and E8. The attack vector E7 can be defeated if any violation of data integrity has been detected. In summary, LiveForen constructs the "*root-of-trust*" by integrating the TPM and the access control policy into the system design to defeat the aforementioned attack vectors.

B. Trust Establishment between the FP and the CP

Recall that a trust relationship must be established between two involved entities, namely the *forensic provider* (FP) and the *cloud provider* (CP). The mutual attestation protocol requires at least three security prerequisites to be satisfied. First, both data sender and receiver must attest with the entire forensic data generation environment (the BIOS, MBR, firmware, and hypervisor). This is accomplished when both parties obtain the public parts of the AIK key from a privacy CA (PCA). A local credential database that contains the mapping relation between PCRs and the trustworthy nodes should also be constructed on both sides. Second, the attributes of the data operators (e.g., the FO and the CO) should be known by both parties so that the access control policy can be generated by using the logical operations of attributes [53]. In addition, the party's private ABE key should be obtained from a trusted key generator, who owns the master key. Third, the secret, such as the symmetric key, the nonce, or the initialization vector, can be revoked and thus they must be synchronized.

Figure 3 illustrates the mutual attestation protocol applied for the purpose of establishing trust. It is based on a standard remote attestation protocol, which starts with an encrypted quote M_1, containing a signed PCR of FI's node PCR_{FI}, a nonce n_1[4],

Fig. 3. The mutual attestation protocol.

a selection of PCR indexes, and the attributes of FO $Attr_{FO}$, which allows an access control policy to be composed later. A digital warrant signed by FO's ABE key is also included. Upon receiving M_1, NA invokes a procedure, namely *Verify_FI*, to authenticate FI and construct the response. In particular, NA first checks the authenticity of PCR_{FI} from the credential database. The computational platform is considered as trustworthy if the values of the PCRs reflect a valid measurement BIOS, MBR, firmware, and hypervisor. Then, a key pair BK^{+}_{CP} and BK^{-}_{CP} that are bound to the selected PCR values are generated. Since the key pair is bound to the particular PCRs value at present, the decryption key is useless if the selected PCRs do not contain the particular values and so as to defend against a TOCTOU attack. The disclosure of $Attr_{CO}$ in M_2 serves a similar purpose, as $Attr_{FO}$ in M_1, to construct an access control policy. Forensic operator validates M_2 by invoking the procedure*Verify_CI*, which is also similar to *Verify_FI*. It generates the response M_3, which includes three parts: 1) a new nonce n_3 encrypted by the public ABE key and enforced by the policy P, 2) the received nonce n_2, and 3) the public bind key BK^{+}_{CP}. A policy P can be specified in such a way that only the authenticated CO can decipher n_3. The forensic

[4]Without otherwise note, the nonces in the protocol are used to check the freshness of the attestation request.

operator confirms M_3 with the response M_4, which includes n_4 and the symmetric session key *sk*, both will be used as the shared secret in the data transmission phase. To indicate the successful reception of the secrets, FP responds with a cipher of n_4 encrypted by *sk* in M_5. Regardless of the number of the parties, the protocol allows the involved parties to mutually authenticate each other's entire data generation environment, as well as the roles of the operators.

Procedure *Verify_FI*

Input: The request to acquire cloud data M_1.
Output: The response M_2.
$[\![PCR_{FI}, n_1, S_{PCR}, Attr_{FP}]\!] \leftarrow$*ReadMessage*$(M_1)$;
$R_1 \leftarrow$*ReadCertificate*(*CP*);
if $(acceptable(R_1))$ **then**
  $[\![BK^{+}_{CI}, BK^{-}_{CI}]\!] \leftarrow$ `TPM_CreateWrapKey`(*PCR*$_{CI}$,S_{PCR});
  $R_2 \leftarrow$ `TPM_PcrRead`(*PCR*$_{CI}$));
  $M_2 \leftarrow \{\langle R_2 \rangle_{CI}, n_1, n_2, Attr_{CO}, BK^{+}_{CI}\}_{FI}$
else
  $M_2 \leftarrow$ *FAIL*;
return M_2;
End Procedure

C. Security Analysis of the Protocol

As mentioned in Section III, the proposed cloud forensic service is designed to defeat the attack vectors in C1. In particular, with the measurements of the selected PCRs, the health of the BIOS, MBR, firmware, and hypervisor can be validated. Since the private part of the AIK key and the bind key are non-migratable and the nonce and session keys are ephemeral, it is impossible for the malicious administrator to steal and replay the credentials on a different physical machine. By protecting the secrets with the ABE key, any access to the key is restricted to the role, rather than the privilege of a user. This defensive mechanism prevents the exploits that the root user abuses the keys. Therefore, the attack vectors of E1 and E2 are defeated. The proposed protocol also defeats an E3 attack. As previously mentioned, the bind key pair is bound to the particular values of PCRs. If the platform is compromised, the measurements of PCRs will be different from the PCRs, which the keys were bound with. Thus, the decryption key becomes useless. In our scheme, the procedure *Verify_FI* and *Verify_CI*, the TPM primitive `TPM_CreateWrapKey` generates a bind key pair, BK^{+}_{CP} and BK^{-}_{CP}, in which the decryption key BK^{-}_{CP} is usable only when the following operations are executed on a physical node's PCRs. For example, a malicious CP launches a TOCTOU attack between M_2 and M_4 by migrating the control to an untrusted node, since M_4 is encrypted by BK^{+}_{FI}, the malicious CP cannot decipher M_4 because the decryption key is useless in a different node with the different measurement of PCRs. The similar mechanism defeats the TOCTOU attack launched between M_2 and M_4 by a malicious FP.

D. Data Transmission

When evidence is transmitted, its integrity should be verified. The existing techniques, such as the Checksum and the Digital Signature, can verify the integrity of data as a monolithic piece but is unable to identify the location of manipulation in the data stream. This limitation, however, leaves space for the attack vectors E7 and E8. It also potentially leads to the exploit that modifies the artifact directly relevant to the particular criminal case and leaves other artifacts intact. In the following, we define some key terminologies in the process of transmitting and verifying the data stream:

figures/watermark_embedding.eps

Fig. 4. GIS embedding scheme.

Definition 1 (Custodian C_i). The *custodian* in secure evidence transmission process is defined as an entity that protects and guards evidence. When evidence D is transmitted between the trusted custodians, we denote the custodian, who sends D as C_i and the custodian, who receives D as C_{i+1}. In the context of this paper, the CP is C_i and the FP is C_{i+1}.

Definition 2 (Group Integrity Signature GIS_j). When evidence D is transmitted in the form of a stream, it can be denoted as $D = \{d_1, d_2, ...d_n\}$, where d_i $(1 \leq i \leq n)$ is an *indivisible* data element. We partition D into m data groups, such that $D = \{G_1, G_2, ...G_m\}$. For group $G_j = \langle d^j_1, ..., d^j_o \rangle$ $(1 \leq j \leq m)$, its last element d^j_o is defined as the *boundary element* of G_j. The size of G_j is denoted as $G_j.size$, and thus $\sum_{j=1}^{m} G_j.size = n$. For the jth group G_j, we define its *Group Integrity Signature* (GIS) as follows:

$$GIS_j = \{H(H(G_j) \| H(G_{j+1}))\}_{sk} \tag{1}$$

where sk is a symmetric key, while $H(G_j)$ is defined as $H(d^j_1 || ... || d^j_o)$.

1) GIS Generation and Embedding: Although the GIS is designed to verify the integrity of data groups, it is challenging to keep the GIS's secrecy for several reasons. First, specifying the group size $G_j.size$ is critical. If $G_j.size$ is a global constant or can easily be guessed, the adversary can manipulate data in a group, recompute the GIS of the modified group, and re-inject the GIS into data stream on the fly. Second, if the GIS is transmitted along with the data and the adversary knows the message format, the GIS can be stripped away from the message and be replaced by ones that verify the modified data. For these reasons, at least three constraints must be ensured when specifying the group size: 1) $G_j.size$ should be almost impossible to guess; 2) $G_j.size$ should be relatively easier to be determined by trusted custodians; and 3) the GIS should be tightly coupled with data and consume minimal bandwidth.

Procedure *Embed_GIS*

Input: streaming data D, nonce n, and session key sk.
Output: network packets in *N_Buff*.

```
D_Buff_1 ← ∅; D_Buff_2 ← ∅;
j ← 1;
D_Buff_2 ←fill_data(D, n);
continue_process = true;
while continue_process == true do
    N_Buff ← ∅;
    if D hasMore data then
        j ← j + 1;
        D_Buff_1 ← D_Buff_2; D_Buff_2 ← fill_data(D, n);
    else
        D_Buff_1 ← D_Buff_2; D_Buff_2 ← fill_data(dummy_data);
        continue_processing ← false;
    H_1 ← H(D_Buff_1); H_2 ← H(D_Buff_2);
    N_Buff ←assemble_packets(D_Buff_1);
    GIS ← {H(H_1||H_2)}_sk;
    embed_watermark(N_Buff, GIS);
end while
TPM_NV_Write(n, H_2, GIS);
```

End Procedure

Procedure *Verify_GIS*

Input: the network packets P in *N_Buff* and nonce n.
Output: the set of tamper group T indexes.

```
T ← ∅; D_Buff_1 ← ∅; D_Buff_2 ← ∅;
i ← 1; flag ← true;
D_Buff_1 ←fill_data(P, n); D_Buff_2 ←fill_data(P, n);
continue_process ← true;
while continue_process == true do
    if N_Buff hasMore data then
        i ← i + 1;
        D_Buff_1 ← D_Buff_2; D_Buff_2 ←fill_data(P, n);
    else
        continue_processing ← false; continue;
    H_1 ← H(D_Buff_1); H_2 ← H(D_Buff_2);
    W ← H(H_1||H_2); GIS = extract_watermark(N_Buff);
    W' ← {GIS}^-_sk;
    if flag == true and W != W' then
        flag ← false; start ← i;
    else if flag == false and W != W'
        end ← i;
    else if flag == false and W == W'
        end ← i − 1; T ← T ∪ {start, end};
        flag ← true;
```

End Procedure

To meet the first two constraints, our scheme allows both custodians to use the following process to determine the boundary element s_m^j on both sides with minimal cost of synchronization: First, the hash of data element d_i is computed as $H(d_i)$. Remember that the mutual authentication protocol exchanges the nonces n_3 and n_4. Our scheme uses the XOR ($\oplus$) of n_3 and n_4 to determine the boundary element of a group. In particular, if $H(d_i)$ mod $(n_3 \oplus n_4) == 0$, then d_i will be chosen as the boundary element. Recall that, as the result of executing the mutual attestation protocol, n_3 and n_4 are the common but ephemeral secret shared between both custodians before data transmission, they will become obsolete when the data transmission session is complete. Therefore, the adversary cannot reuse them after the data transmission or when the transmission resumes thereafter. If both parties agree to this scheme, both custodians adopt the common secrets to determine the group size without any synchronization scheme.

To meet the third constraint, our solution is inspired by the concept of information hiding, more specifically, the fragile watermark [54]. Compared with other techniques that transmit the integrity data along with the data, the fragile watermark demonstrates several advantages. First, it does not incur additional communication overhead in network payload as the watermarks are simply embedded in the unused network packets fields. Second, the embedded watermark does not degrade the perceptual quality of the host data and is thus invisible to the adversary. Therefore, it is extremely challenging for the adversary to guess the location of the watermarks and remove them. Third, for the purpose of obsoleting the adversary's knowledge about how the watermarks are generated, C_i and C_{i+1} will always alter the parameters of the scheme, from which the watermarks are embedded and verified.

Figure 4 illustrates the watermark embedding process. To embed GISes as the watermark, the DT maintains three buffers: two data buffers *D_Buff*$_1$, *D_Buff*$_2$, and one network buffer *N_Buff*. Specifically, *D_Buff*$_1$ is used to store the data of the current group, while *D_Buff*$_2$ is used to store the data of the next group. *N_Buff* is used as the buffer to hold a group of network packets, which assemble data elements from *D_Buff*$_1$ and the watermarks.

The process of embedding GISes as the watermark into network packets is delineated in Procedure *Embed_GIS*. Specifically, data elements of group G_j and G_{j+1} are first fed into *D_Buff*$_1$ and *D_Buff*$_2$, respectively. The subroutine *fill_data* fills the data buffer by first computing the hash of data element, and then uses the nonce n to determine the boundary element of a group. The group hashes, H_1 and H_2, are used to compute GIS_j, which is then embedded into the packets in *N_Buff* via the subroutine *embed_watermark*. The subroutine *embed_watermark* chooses an unused packet header field to embed the bits of GIS_j. The ending vector at the end of the data stream can be padded by the dummy data that is agreed upon by C_i and C_{i+1} in advance. Hence, the embedded watermarks are eventually chained over groups and the malicious modification in a group can be detected.

2) GIS Verification: When C_{i+1} receives the network packets, the Procedure *Verify_GIS* is invoked to verify data integrity by groups. More specifically, the GIS verification component first reads data from *N_Buff* into *D_Buff*$_1$ and *D_Buff*$_2$ by invoking routine *fill_data*. Then, the hashes of *D_Buff*$_1$ and *D_Buff*$_2$ are computed as H_1 and H_2, respectively, and were concatenated as W. Meanwhile, the *GIS* is extracted from *N_Buff*, decrypted by sk, and compared with *W*. Finally, if the extracted watermark matches with the computed one, the data in G_i passes the integrity verification; otherwise, it is considered to have been tampered with. If any tampering has been detected, the index range (*start* and *end*) of the manipulated group will be included in the set *T* and be returned by the procedure. When a malicious data manipulation is detected, it can be categorized as one of three categories of attacks: *data modification*, *data deletion*, or *data insertion*. A data modification attack refers to an attack that modifies the payload of some network packets in the path of data transmission. A data deletion attack refers

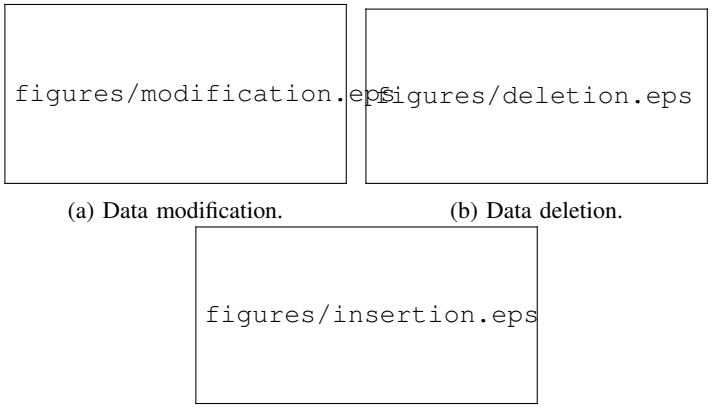
to an attack that deletes some contents of network flow. A data insertion attack refers to an attack that injects additional malicious content into the network flow.

Figure 5 shows the scenarios and the detection of these three attacks. The network packet group and the data groups are denoted as PG_i and G_i $(1 \leq i \leq 5)$, respectively. The arrows from PG_i to G_i show the flows of raw data that are passed from *N_Buff* to *D_Buff*. To detect these attacks, C_{i+1} first extracts the watermarks W_i $(1 \leq i \leq 5)$ from each packet group PG_i $(1 \leq i \leq 5)$ and then compares them with those computed from G_i and G_{i+1}. The watermark matches and mismatches are also illustrated in the subfigures.

Figure 5a illustrates a scenario of data modification attack, in which packet group PG_3 is modified to PG'_3 by the adversary and thus causes two watermark mismatches: the first mismatch occurs when comparing W_2 with $H(H_2\|H_3)$, while the second mismatch occurs when comparing W_3 with $H(H_3\|H_4)$. The modification of the packet group PG_i can be detected if the verification of two adjacent watermarks, W_{i-1} and W_i, fails.

Similarly, Figure 5b illustrates a scenario of data deletion attack, in which packet group PG_3 is completely deleted from the data stream. In this case, only one mismatch occurs when comparing W_2 with $H(H_2\|H_4)$. This mismatch does not propagate unless the adversary deletes more than one packet group. Therefore, the deletion of one packet group PG_i can be detected if the verification of W_{i-1} fails. At last, Figure 5c illustrates a scenario of data insertion attack, in which an extra packet group PG'_2 is inserted between PG_2 and PG_3. The detection criteria are similar to that of the data modification, meaning that the verification of two adjacent watermarks fails. It is possible but extremely challenging to circumvent the detection for at least two reasons. First, in order to modify the content of one data group without being detected, it requires the adversary to generate a new watermark that matches with altered data. To do that, the adversary needs to know the symmetric key sk, the nonce n, the specific hash function, and the particular field of the network packet where the GISes were embedded, which is extremely challenging for a man-in-the-middle attacker. Second, if the adversary only has partial knowledge of the secret, the mismatch of the watermarks will propagate to all the mismatched data groups, following the tampered data group, which can easily be detected.

3) Security Analysis of the Watermark Scheme: The advantage of our scheme is that any tampering made to a data stream can be detected and narrowed down to a group. Also, a failure of verifying the integrity of one group only affects the detection of two adjacent data groups and does not propagate to the future groups. As previously mentioned, to circumvent the detection requires the adversary to determine the boundary element, which is extremely challenging; the deletion and insertion of data elements within a group can be simply considered as the modification of that group. Although the detection scheme incurs no false positive, the false negative (FN) still exists during the detection. That is, the adversary makes the malicious modification, deletion, and insertion, while the malicious actions remain undetectable. The formal analysis of the FNs is listed below:



(a) Data modification. (b) Data deletion.

(c) Data insertion.

Fig. 5. Three data manipulation attacks and their detection. The ✓indicates watermark match; while the × indicates watermark mismatch.

Definition 3 (Group-related Parameters). Given a streaming evidence D, which is partitioned as a number of data groups in sequence. The value of the pseudorandom number generated by the TPMs is denoted as m. The upper bound, lower bound, and average length of a group are denoted as U, L, and M.

Theorem 1. The false negative that the scheme fails to detect a single malicious modification is $FN_{modification}$, which is upper-bounded by $\frac{1}{2^{L-1}}$.

Proof. Let's assume that the probability that the maliciously modified element becomes a boundary element is $1/m$, and the probability that the modified element does not become a boundary element is $1 - 1/m$. Let's consider two conditions below:

Condition 1. The maliciously modified element becomes a boundary element. We have two sub-conditions: 1) one of the first $L-1$ elements becomes the boundary element. Since L is the lower bound of a group, two groups will be combined into one group, whose average length is M; 2) one of the later $M-L+1$ element becomes the boundary element, the current group will be split into two groups, each of which has at least L elements.

Condition 2. The maliciously modified element does not become a boundary element. Since the modification can happen at any of the M positions. Under the condition, the probability that verifies a group without detecting any error is $1/2^M$.

Combining these two conditions, the false negative of this case is:

$$\begin{aligned} FN_{modification} &= \frac{1}{m}\left(\frac{L-1}{M}\cdot\frac{1}{2^M}+\frac{M-L+1}{M}\cdot\frac{1}{2^{L-1}}\right) \\ &\quad +\left(1-\frac{1}{m}\right)\cdot\frac{1}{2^M} \\ &< \frac{1}{m}\left(\frac{L-1}{M}\cdot\frac{1}{2^{L-1}}+\frac{M-L+1}{M}\cdot\frac{1}{2^{L-1}}\right) \\ &\quad +\left(1-\frac{1}{m}\right)\cdot\frac{1}{2^{L-1}} \\ &= \frac{1}{2^{L-1}} \end{aligned} \tag{2}$$

Finally, from the Definition 1, 2, and 3, (2) follows. □

Lemma 1. The false negative that the scheme fails to detect

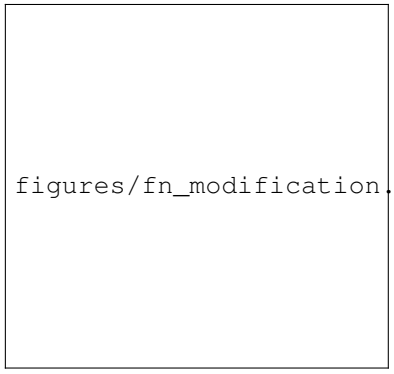


Fig. 6. The false negative (FN) of detecting malicious modification.

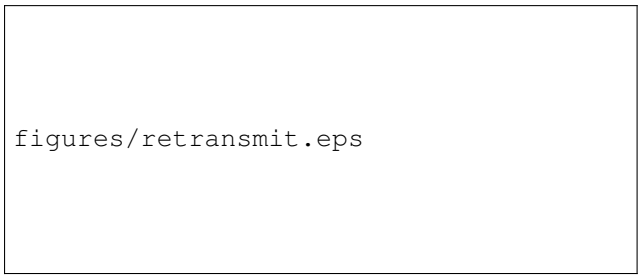


Fig. 7. The data transmission resumption protocol.

a single malicious deletion is $FN_{deletion}$, which is upper-bounded by $\frac{1}{2^{M-1}}$. □

Lemma 2. The false negative that the scheme fails to detect a single malicious insertion is $FN_{insertion}$, which is upper-bounded by $\frac{1}{2^{L-1}}$. □

The detailed proofs of Lemma 1 and Lemma 2 are included in the Appendix. From the theoretic analysis, we know that the false negatives of detecting the malicious modification, deletion, and insertion can be suppressed exponentially, if the parameters of L and M are large enough. Figure 6 clearly shows that the false negative of detecting malicious modification exponentially decreases with the increasing value of L. Obviously, the negligible false negative can be easily accomplished if we choose large enough nonces for n_3 and n_4 (e.g., with 256 bits or 512 bits).

E. Data Transmission Resumption

As an important requirement for cloud forensics, the provision of evidence should be able to resume under the same warrant [21], [22]. To meet this requirement, newly generated evidence needs to be transmitted to the forensic operator upon the request. However, it is non-trivial to resume data transmission unless two questions can be properly answered: 1) how could the involved parties re-authenticate without repeating the time-consuming authentication process described in Section IV-C? 2) is it possible to ensure that the retransmitted data is not stale and is continued from the previous transmission process?

Procedure *Request_Resumption*

Input: NULL.
Output: The data resumption request M_1.
$[\![n_i, H(G_\ell), GIS_\ell]\!] =$ `TPM_NV_ReadValue()`;
$t_1 \leftarrow$ `TPM_TickStampBlob();`
$P \leftarrow ((GIS{==}GIS_\ell)$ **and** $(nonce{==}\, n_i)$ **and** $(time \leq t_1))$;
$M_1 \leftarrow \{\{t_1, PRC_{FI}\}_{AIK_{FI}}, \{n_{i+1}, sk_{i+1}, H(G_\ell),\}^{P}_{CO}\}_{BK^{+}_{CP}}$
return M_1;
End Procedure

To answer the first two questions, we proposed the data transmission resumption protocol, which is illustrated in Figure 7. To request data resumption, the forensic provider prepares data resumption request M_1 with the procedure *Request_Resumption*. This procedure first reads TPM's NVRAM (via `TPM_NV_ReadValue` primitive) to obtain the nonce n_i and the last GIS saved at the end of the previous data transmission process GIS_ℓ. Then, the signed clock tick count is read from the TPM. To challenge the CP, an access control policy *P* is composed to include the value of GIS and nonce used in the previous transmission process, along with the current clock tick count. Finally, the message is encrypted with the public bind key BK^{+}_{CP}. The purpose of reusing the bind key of the CP is to ensure that the CP does not migrate the control to an untrusted platform. When the CP receives M_1, it invokes the procedure *Response_Resumption* to validate the credentials in M_1 and compose the response. Given $H(G_{\ell-1})$ being read from the NVRAM, it first recomputes the last GIS of the previous data transmission process GIS_{prev}. Then, it composes the set of attributes *Attr_Set*, based on GIS, nonce, and clock tick count. Based on *Attr_Set*, the private key is generated by the subroutine *Key_Gen* to decrypt the cipher-text, which is determined by the *Attr_Set* that satisfies the policy. If the deciphering process is successful, both n_{i+1} and sk_{i+1} will be used as the common secret in the procedure to generate and verify the watermark.

Security Analysis The process *Request_Resumption* meets the security requirements by performing data resumption from an authenticated infrastructure and a qualified operator. First, the successful generation of M_1 depends on the data generated from an authenticated FP infrastructure with a fresh quote. Second, in order to decipher M_1, the message receiver must be executed on the infrastructure, which has been entrusted by the mutual attestation protocol. In addition, the credentials used for the transmission resumption, including the newly generated nonce and session key, can be deciphered only when the attributes of the previous session satisfy the access control policy on both sides. Finally, on the cloud side, the protocol incurs limited TPM-related operation, such as `TPM_NV_ReadValue` and the quote authentication, which reduces the potential cloud bottleneck risk of authenticating multiple sessions in parallel [31] and even avoids victimization of the DoS attacks.

V. IMPLEMENTATION

We implemented the prototype of LiveForen on the top of Xen 4.5 [55], which comprises the key components of NA and DT. The key components of LiveForen were implemented with nearly 1400 lines of `C` code. Cryptographic functions such as symmetric key operation were implemented with OpenSSL [56]

and Intel TPM Software Stack (TSS) 2.0 [57] for TPM-related operations. To ensure a secure bootstrap, the physical host is booted through TrustedGRUB [58] and TrustedSeaBios [59]. The volatile forensic artifacts are acquired with LibVMI [60] libraries

To evaluate scalability and throughput, `pthread` is used to emulate parallel data streams. The watermarks are embedded into the TCP packet header field Timestamp Value (TSVAL). The CP-ABE-related operations leverage CP-ABE Toolkit [53]. The evaluation results were collected from the physical machine which is equipped with an Intel i7-2600 3.40 GHz, 4-core processor, 16GB of RAM, and TPM 2.0 with Intel Trusted Execution Technology (TXT) enabled.

VI. Evaluation

In this section, we first analyze the security properties promised by the proposed protocol with the formal security protocol verifiers (Section VI-A). Then, we measure the step-wise overheads of the attestation protocol, which involves the TPM-related, CP-ABE-related, and crypto-related operations (Section VI-B). Next, we evaluate the effectiveness and performance overhead of generating and verifying the GISes (Section VI-C). Finally, we evaluate the scalability of the prototype by measuring the latency and CPU usage with respect to parallel forensic requests (Section VI-D).

A. Formal Protocol Verification

The first set of the experiments verifies the *correctness* of the security protocols proposed in Section IV. In particular, the correctness refers to the strong security properties of the protocol, as well as its resilience to certain attacks. A large number of techniques and tools have been proven to be efficient to automatically verify protocols with symbolic model. In this evaluation, we complementarily use two formal security protocol verifiers: ProVerif [61] and Scyther [62]. We choose them because of their wide adoptions in prior work [18], [31]. More specifically, ProVerif is used to prove the correctness of the semantics of policy-sealed data with the presence of an adversary with unrestricted network access. The attacks covered by ProVerif include eavesdropping, decomposing, shuffling, replay, and man-in-the-middle attacks. In addition, Scyther is used to verify the strong security properties, such as data secrecy, aliveness, weak agreement, agreement, and synchronization. Table III lists the detailed information about the security protocol verifiers used for evaluating the mutual attestation protocol. We refer the interested readers to the code of protocol verification for ProVerif and Scyther from the supplementary materials and our online repository[5].

TABLE III. The verification tools used for evaluation.

Tool	Security properties	SLOC
ProVerif v1.90	Reachability, secrecy, and correspondence assertion [61].	98
Scyther v1.1.3	Data secrecy, aliveness, weak agreement, non-injective agreement, and non-injective synchronization [62].	57

B. Performance of Attestation

The second set of experiments measure the step-wise overhead of the protocol. For each step of the protocol mentioned in Section IV-B, The TPM-related, CP-ABE-related, and symmetric key-related overheads are measured. In particular, the TPM-related encryption includes the operations of 1) quoting and signing the PCR values, 2) encrypting the message with the public bind key, and 3) the generation of nonces. The TPM-related decryption overhead includes the time spent to decrypt the message with the public AIK key and private bind key. The CP-AB-related overhead includes encrypting or decrypting the message with the CP-ABE keys along with the access control policy. The CP-ABE-related operations use 10 attributes. The AIK key uses 256-bytes RSA key. The symmetric key-related operations use 32-bytes AES key.

Figure 8 illustrates the step-wise overhead for TPM-related, CP-ABE-related, and symmetric key-related operations, where their means and standard deviations are also included. The overhead of each step is measured accumulatively from both sides of communication, which have the same hardware setting. It shows that the TPM-related operations incur much more overhead than other operations: it takes about one order of magnitude longer than that of CP-ABE encryption and two orders of magnitude longer than that of CP-ABE decryption.

Although researchers claim that the TPM-related operations contribute to the bottlenecking of the authentication and the crypto-related functionalities [31], [35], the analysis shows that only a few TPM operations, such as generating and loading keys, take additional time. Hence, several possible solutions might be applied to mitigate TPM-related overhead. First, to avoid inquiring the TMP at run time, the private AIK key can be pre-loaded into a protected memory location before performing TPM operations, e.g., the memory of enclave that is protected by CPU. Second, the number of binding operations can be limited. Right now, the binding operations of the attestation protocol only been invoked twice (in messages 3 and 4). Furthermore, the protocol only performs at the moment 1) when mutual attestation between two unknown parties is initialized, or 2) when the physical machine is rebooted. The protocol can provide reasonable scalability even when a large number of parallel mutual authentication requests are presented.

Given the fact that the TPM-related overheads are nearly consistent for a fixed key size, it is also important to evaluate the performance impact of CP-ABE operations on the mutual attestation protocol with respect to the increasing number of attributes. Figure 9 shows the CP-ABE-related overhead and the file size of the encrypted data with respect to the number of attributes in a policy. Both the overhead of CP-ABE en/decryption and the resulting file size are nearly perfectly linear to the number of the attributes associated with the key. The result satisfies with what was discovered in the literature [63]: the performance of CP-ABE en/decryption is linear to the specific access tree of the cipher-text and attributes available in the

[5]http://www.secs.oakland.edu/∼anyiliu/T-IFS-09071-2018/Supplementary-material-T-IFS-09167-2018.pdf

Fig. 8. Step-wise overhead for the mutual attestation protocol.

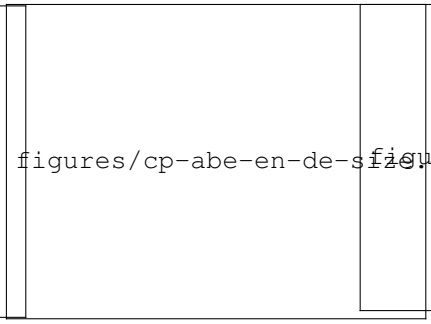


Fig. 9. CP-ABE en-/decryption overhead and the resulting file size.

Fig. 11. Performance overhead of parallel server attestation and verification.

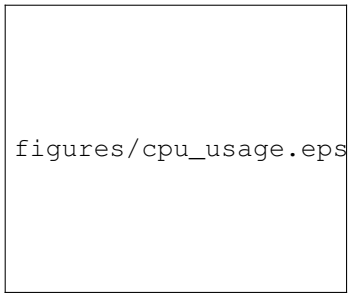
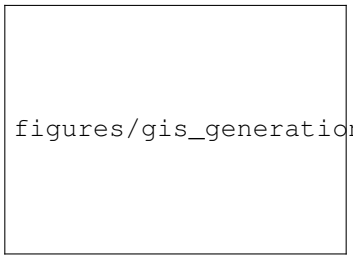


Fig. 12. CPU usage for different request frequencies.



(a) The overhead of GIS generation with different buffer size.

(b) The overhead of GIS verification with different buffer size.

Fig. 10. The overhead of GIS generation and verification.

private key.

C. Data Integrity Verification

The third set of experiments measures the computational overhead for GISes generation and verification. Figure 10a illustrates the means and standard deviations of the elapsed time for generating GISes for 100MB, 1GB, 10GB of data with different buffer sizes. There are at least three findings. First, the latency of generating GISes is virtually linear to the size of dataset. This fact can be explained as the complexity of the GIS generation algorithm is linear to the size of the input data. Second, the performance overhead is quite reasonable regardless of the buffer size. For example, the time elapsed for generating the GISes of 10GB data is less than 2 seconds. Third, as an important parameter, the size of the buffer presents a tradeoff between the performance overhead and the detective locality: when the size of the buffer increases, the time for computing data's hash in a buffer and the time for embedding watermarks increase. While at the same time, the probability of localizing the data manipulation decreases. This explains why the latency of generating GISes when the buffer size is 10KB is less than the latencies of generating GISes when the buffer size is 1KB and 100KB. The similar result can also be found for GISes verification, which is illustrated in Figure 10b.

D. Scalability

The final set of experiments evaluates the scalability of LiveForen with respect to the number of simultaneous evidence acquisition and transmission requests. Figure 11 illustrates the elapsed time of authentication and verification when LiveForen receives a number of requests. The number of simultaneous requests scales from 1 to 1000. LiveForen answers a new request by spawning a new thread. The measurement omits the elapsed time of generating and loading keys for TPM, and only counts the accumulative overhead related to cryto-related operations. Observations show that the latencies for both operations are almost sub-linear to the number of requests. Figure 12 illustrates the CPU usage with respect to the frequency of requests. When the number of requests is less than 150 per second, the CPU usage is less than 20%. However, when the number of requests exceed 350 per second, the CPU usage grows to nearly 58%. This can be explained as most TPM-related operations are serialized without further optimization. To improve performance, further work will be to consider batch attestation, as suggested in the literature [31].

VII. Discussion

In this section, we compare LiveForen with the prior work and discuss its open issues, along with the possible solution and future improvements.

A. Comparison to Prior Work

We first compare our approach with the prior work [14], [15], [18], [21], [22], [25], [26], [31], [49]–[52] as follows.

First, prior work [50], [51] present the systems that focused on sharing online forensic data [49], preserving the integrity of forensic acquisition logs [50], and providing the proof of past data possession. These systems assume the trustworthiness of the cloud infrastructure and are incapable of identifying forensic artifacts from virtual images of the VMs. Compared with them, LiveForen demonstrates at least three advantages. First, it is built upon the TCB whose integrity measurement has been validated. Once the TCB is authenticated, the data collection process is automated, including the data transmission and resumption, without human intervention. The components of LiveForen are granted the access to communicate with the TPM and each other only if their code space has been validated to be trustworthy. Second, LiveForen prevents the privileged operator from issuing malicious commands in such a way: it only answers the commands whose issuer and originated platform are both authenticated. Therefore, it denies the malicious command even if it is issued from the privileged platform and/or the

privileged operator. Third, the fragile watermark is embedded in the data stream and thus allows malicious data manipulation to be detected at run time, while the prior work does not provide such an error detection capability.

Second, some prior work [18], [31] harnesses TPM to generate credible TCB. However, their major design goals are to construct trusted cloud services in general but lack forensic readiness. Furthermore, some forensic specific features, such as accountability and tamper detection on forensic data, are also missing in their system designs. Other prior work [14], [25] do consider the extraction of virtual forensic artifacts from cloud infrastructure. Unfortunately, their systems also rely on the hardware and software and human operators, whose trustworthy have not been fully authenticated and verified.

Third, although the prior work [21], [22], [52] comprehensively surveys the open issues of cloud forensics, they do not present any solutions to address the issues. Instead, LiveForen presents solutions to address the issues.

It is worth noting that LiveForen also addresses some similar security issues as the existing secure communication protocols, such as Transport Layer Security (TLS), and is intended to achieve data confidentiality and verifiable data integrity. The recently published specification of TLS 1.3 [64] removes the known vulnerabilities and significantly improves the security features, such as pruning the legacy crypto algorithms, supporting more secured cryptographic schemes and perfect forward secrecy (PFS) in its cipher suites, and optimizing the handshake sub-protocol. Hence, TLS is primarily designed to establish a secure communication channel to ensure authentication, key exchange, the data secrecy and integrity at the transport layer.

Compared with TLS 1.3, LiveForen works beneath the transport layer and emphasizes a different set of attack vectors. In particular, it maintains a runtime environment on the clientâs and serverâs hosts, which generate, store, and access the crypto keys in a trustworthy runtime environment. The crypto keys are secured in the TPM and only allows key access if the runtime environment is verified to be trustworthy. In addition, LiveForen can verify the integrity of evidence on the fly, which allows data insertion, deletion, and manipulation to be detected and localized in a trusted environment. Last but not least, we complementarily use two formal security protocol verifiers to verify the correctness of the security properties of the proposed protocols. Therefore, the proposed protocols of LiveForen can complement the existing protocols (e.g., TLS 1.2/1.3).

B. The Reduction of TCB

Although the threat model of the proposed system only considers software exploits, rather than the hardware ones; researchers have revealed that the TPM can be subverted by various hardware exploits [46], [65]. Besides, the proposed system, which is built on the top of the trustworthy hypervisor, Trusted Computing Group (TCG) software stack, VMI library, comprises a large TCB. To reduce the size of TCB even without the provision of a trustworthy platform, one possible solution is to construct an enclave for each component of LiveForen, whose integrity is measured first. The authentication between enclaves can be implemented as the standard local attestation or remote attestation [66]. More precisely, if the components are co-existing on the same platform, the local attestation will be applied: the enclaves first run the SGX instruction `EREPORT` to generate a signed report, namely `REPORT`. Then, they exchange `REPORT` to authenticate the public Diffie-Hellman keys. If the components are deployed on different platforms, the remote attestation will be used: the quoting enclave on each platform verifies and generates a quote signed by its `EPID` key. The credential shared between enclaves can then be protected with the sealing operations. However, several drawbacks of SGX limit its application in the scenario of a live forensics. For example, unlike the TPM, which provides non-volatile storage and built-in counter, SGX lacks of such trusted storage, secure counter, and secure clock, which make it vulnerable to the rollback attack [67]. In addition, SGX is not resilient to attacks, such as the buffer-overflow attack [39] and the side-channel attack [40]. Finally, the current attestation protocol contains the vulnerability that allows partial attestation protocols to be emulated outside the enclave [68], which is out of the scope of this paper. As a future direction, we will investigate the impacts of the attacks toward TPM and other hardware-assisted execution environments (HEEs).

C. Compliance with the Digital Forensic Principles and Laws

A fundamental principle of digital forensic practice is that neither law enforcement agencies nor their agents should change data held on a computer or storage media, which may subsequently be relied upon in court [21]. One concern is that embedding watermarks into the data stream might invalidate the widely accepted principle. We argue that the proposed approach does not invalidate the forensic principle for at least three reasons. First, the fragile watermark is computed from the raw data but do not alter the data itself. Since it only serves as the provable checksum that verifies the integrity of the streaming data, it can be decoupled from the evidence easily if the bandwidth is not a concern. More precisely, the GISes are still computed from the data, but are transmitted in a dedicated channel. Hence, a proper synchronization mechanism is needed to ensure that the GISes can verify their corresponding data groups on the fly. Second, as another attacker vector, a compromised data transmitter can tamper the raw evidence during the GIS embedding phase. However, as a pre-requisite to run LiveForen, the integrity of the data transmitter must be verified. A compromised data transmitter won't pass the verification and can thus be easily detected. Lastly, Grispos et al. [21] states that although the forensic provider can use the cloud provider's tools to verify the integrity of the evidence, the ability to validate the correctness of the tools might be limited. In LiveForen, however, the verification of data integrity depends on both parties. That is, the nonces (n_3 and n_4) used to determine the group size are generated by the CP and the FP, respectively. Therefore, the ability of validating the correctness of the data depends on both custodians and thus is not limited.

D. The Selection of the Header Field

To make sure that the data recipient knows which network packet header field contains GIS, we apply the follow scheme to

synchronize the information of the chosen packet header field by both parties. Specifically, we define the partially ordered set of packet header fields as $F = \{f_1, f_2, ...f_n\}$, where f_i $(1 \leq i \leq n)$ denotes the packet fields and the size of F, namely $F.size$, is n. In order to synchronize the information of the chosen index in F, both parties compute $Index\_of\_F = (sk \oplus n_3 \oplus n_4) \bmod n$, in which the session key sk, nonces n_3, and n_4 have been exchanged by the attestation protocol. Since the adversary cannot successfully obtain sk, n_3, and n_4 (See Section IV for details), *Index_of_F* will keep confidential during the session. Moreover, since symmetric session key sk and the nonces are ephemeral during a data transmission session and will be obsoleted in a resumed session. Therefore, it prevents the adversary from reusing the *Index_of_F* of the previous session and still allows both parties synchronize the information of the chosen packet header field for embedding the watermark. When n is small, this scheme is susceptible to brute-force attack if the adversary has the prior knowledge of the unused packet fields and exhaustively compute their entropy. We will explore possible measures for embedding watermark with enhanced security in our future work.

VIII. Conclusions

To address the open challenges of evidence acquisition and transmission in an IaaS cloud environment, a novel framework that allows forensic evidence to be acquired and transmitted between trusted platforms is presented. Forensic data are transmitted as streaming data, in which the data integrity information is generated as the fragile watermarks. The benefits of using the watermarks allow custodians to verify the data integrity, as well as detect and localize malicious modification at run time. The prototype of our system has been implemented in an IaaS cloud environment, whose security properties have been proven by formal security protocol verifiers. The experiments demonstrate that LiveForen can achieve low performance overhead for evidence acquisition and transmission, and good scalability in a trusted IaaS cloud environment.

Acknowledgment

This work is supported by the funding from the National Science Foundation under award Grant No. DGE-1723707, DGE-1623713, CNS-1745894, and the Michigan Space Grant Consortium. We are grateful to James Elliott, Nathaniel Grisham, Aditi Patil, Michelle Pfarrer, and James Huber for their preliminary implementation, constructive discussion, and proofreading.

References

[1] S. T. King, P. M. Chen, Y.-M. Wang, C. Verbowski, H. J. Wang, and J. R. Lorch, "Subvirt: Implementing malware with virtual machines," in *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, May 2006, pp. 314–327.

[2] V. Ciancaglini, M. Balduzzi, R. McArdle, and M. Rösler, "Below the surface: Exploring the deep web," 2015, [Online]. Available: https://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp_below_the_surface.pdf.

[3] Symantec, "Avoiding the hidden costs of the cloud," [Online]. Available: https://www.symantec.com/content/en/us/about/media/pdfs/b-state-of-cloud-global-results-2013.en-us.pdf.

[4] R. Samani and F. Paget, "Cybercrime exposed: Cybercrime-as-a-servic," 2013, [Online]. Available: http://www.mcafee.com/jp/resources/white-papers/wp-cybercrime-exposed.pdf.

[5] M. Darwish, A. Ouda, and L. Capretz, "Cloud-based DDoS attacks and defenses," in *Proceedings of the 2013 International Conference on Information Society*, June 2013, pp. 67–71.

[6] D. Goodin, "Zeusbot found using Amazon's EC2 as C&C server," [Online]. Available: http://www.theregister.co.uk/2009/12/09/amazon_ec2_bot_control_channel/.

[7] M. D. Ryan, "Cloud computing security: The scientific challenge, and a survey of solutions," *Journal of Systems and Software*, vol. 86, no. 9, pp. 2263–2268, September 2013.

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

[9] NIST Information Technology Laboratory, "NIST cloud computing forensic science challenges," 2013, [Online]. Available: http://csrc.nist.gov/publications/drafts/nistir-8006/draft_nistir_8006.pdf.

[10] U.S. Congress, "Cloud Act," 2018, [Online]. Available: https://www.congress.gov/bill/115th-congress/house-bill/4943.

[11] "Forensics at the OJ Simpson Trial," 2017, [Online]. Available: https://www.crimemuseum.org/crime-library/famous-murders/forensic-investigation-of-the-oj-simpson-trial/.

[12] Regional Computer Forensics Laboratory, "RCFL annual reports fy2016," 2016, [Online]. Available: https://www.rcfl.gov/downloads.

[13] Y. Fu and Z. Lin, "Space traveling across vm: Automatically bridging the semantic gap in virtual machine introspection via online kernel data redirection," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, May 2012, pp. 586–600.

[14] J. Dykstra and A. T. Sherman, "Acquiring forensic evidence from infrastructure-as-a-service cloud computing: Exploring and evaluating tools, trust, and techniques," *Digital Investigation*, vol. 9, pp. 90–98, August 2012.

[15] S. Zawoad and R. Hasan, "I have the proof: Providing proofs of past data possession in cloud forensics," in *Proceedings of the 2012 International Conference on Cyber Security*, December 2012, pp. 75–82.

[16] "Forensic toolkit," [Online]. Available: http://accessdata.com/solutions/digital-forensics/forensic-toolkit-ftk.

[17] "EnCase," [Online]. Available: https://www.guidancesoftware.com/products/Pages/encase-forensic/overview.aspx.

[18] J. Seol, S. Jin, D. Lee, J. Huh, and S. Maeng, "A trusted IaaS environment with hardware security module," *IEEE Transactions on Services Computing*, vol. 9, no. 3, pp. 343–356, May 2016.

[19] M. Nyamagwa, J. Liu, A. Liu, and T. Uehara, "Cloudforen: A novel framework for digital forensics in cloud computing," *Journal of Harbin Institute of Technology*, vol. 21, no. 6, pp. 39–45, 2014.

[20] A. Liu, J. Liu, and T. Uehara, "Secure streaming forensic data transmission for trusted cloud," in *Proceedings of the 2nd International Workshop on Secur. and Forensics in Communication Systems*, June 2014, pp. 3–10.

[21] G. Grispos, T. Storer, and W. B. Glisson, "Calm before the storm, the challenges of cloud computing in digital forensics," *International Journal of Digital Crime and Forensics*, vol. 4, no. 2, pp. 28–48, 2012.

[22] K. Ruan, *Cybercrime and Cloud Forensics: Applications for Investigation Processes*, 1st ed. Hershey, PA, USA: IGI Global, 2012.

[23] S. Zawoad and R. Hasan, "Cloud forensics: A meta-study of challenges, approaches, and open problems," *CoRR*, vol. abs/1302.6312, 2013.

[24] N. H. Ab Rahman and K.-K. R. Choo, "A survey of information security incident handling in the cloud," *Computers and Security*, vol. 49, no. C, pp. 45–69, March 2015.

[25] J. Dykstra and A. T. Sherman, "Design and implementation of FROST: Digital forensic tools for the OpenStack cloud computing platform," *Digital Investigation*, vol. 10, pp. 87–95, August 2013.

[26] G. Grispos, W. B. Glisson, and T. Storer, "Using smartphones as a proxy for forensic evidence contained in cloud storage services," in *Proceedings of the 46th Hawaii International Conference on System Sciences*, January 2013, pp. 4910–4919.

[27] A. M. Mikhail Gorobets, Oleksandr Bazhaniuk and Y. B. Andrew Furtak, "Attacking hypervisors via firmware and hardware," 2015.

[28] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki, "Flicker: An execution infrastructure for TCB minimization," in *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems*, April 2008, pp. 315–328.

[29] S. Berger, R. Cáceres, K. A. Goldman, R. Perez, R. Sailer, and L. van Doorn, "vTPM: Virtualizing the trusted platform module," in *Proceedings of the 15th USENIX Security Symposium*, July 2006, pp. 305–320.

[30] F. J. Krautheim, D. S. Phatak, and A. T. Sherman, "Introducing the trusted virtual environment module: A new mechanism for rooting trust in cloud computing," in *Proceedings of the 3rd International Conference on Trust and Trustworthy Computing*, June 2010, pp. 211–227.
[31] N. Santos, R. Rodrigues, K. P. Gummadi, and S. Saroiu, "Policy-sealed data: A new abstraction for building trusted cloud services," in *Proceedings of the 21st USENIX Security Symposium*, August 2012, pp. 175–188.
[32] E. G. Sirer, W. de Bruijn, P. Reynolds, A. Shieh, K. Walsh, D. Williams, and F. B. Schneider, "Logical attestation: An authorization architecture for trustworthy computing," in *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, October 2011, pp. 249–264.
[33] S. Butt, H. A. Lagar-Cavilla, A. Srivastava, and V. Ganapathy, "Self-service cloud computing," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, 2012, pp. 253–264.
[34] F. Zhang, J. Chen, H. Chen, and B. Zang, "Cloudvisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization," in *Proceedings of the 23rd Symposium on Operating Systems Principles*, October 2011, pp. 203–216.
[35] R. Kotla, T. Rodeheffer, I. Roy, P. Stuedi, and B. Wester, "Pasture: Secure offline data access using commodity trusted hardware," in *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation*, October 2012, pp. 321–334.
[36] C. Chen, H. Raj, S. Saroiu, and A. Wolman, "cTPM: A cloud tpm for cross-device trusted applications," in *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation*, April 2014, pp. 187–201.
[37] Intel Corporation, *Intel Trusted Execution Technology: Software Development Guide*, [Online]. Available: http://download.intel.com/technology/security/downloads/315168.pdf.
[38] F. McKeen, I. Alexandrovich, I. Anati, D. Caspi, S. Johnson, R. Leslie-Hurd, and C. Rozas, "Intel software guard extensions support for dynamic memory management inside an enclave," in *Proceedings of the 2016 Hardware and Architectural Support for Security and Privacy*, June 2016, pp. 101–109.
[39] D. Kuvaiskii, O. Oleksenko, S. Arnautov, B. Trach, P. Bhatotia, P. Felber, and C. Fetzer, "SGXBOUNDS: Memory safety for shielded execution," in *Proceedings of the 12th European Conference on Computer Systems*, April 2017, pp. 205–221.
[40] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, "Cache attacks on intel sgx," in *Proceedings of the 10th European Workshop on Systems Security*, April 2017, pp. 1–6.
[41] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," *CoRR*, vol. abs/1801.01203, 2018.
[42] D. Dean and A. J. Hu, "Fixing races for fun and profit: How to use access(2)," in *Proceedings of the 13th USENIX Security Symposium*, August 2004, pp. 14–14.
[43] Microsoft, "Azure migration center," [Online]. Available: https://azure.microsoft.com/en-us/migration/.
[44] Amazon, "Aws server migration service user guide," [Online]. Available: https://docs.aws.amazon.com/server-migration-service/latest/userguide/server-migration-ug.pdf.
[45] A. Atya, A. Aqil, K. Khalil, Z. Qian, S. V. Krishnamurthy, and T. F. L. Porta, "Stalling live migrations on the cloud," in *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, August 2017.
[46] B. Parno, J. R. Lorch, J. R. Douceur, J. Mickens, and J. M. McCune, "Memoir: Practical state continuity for protected modules," in *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, May 2011, pp. 379–394.
[47] J. Winter and K. Dietrich, "A hijackers guide to communication interfaces of the trusted platform module," *Computers & Mathematics with Applications*, vol. 65, no. 5, pp. 748–761, March 2013.
[48] E. M. Chan, J. C. Carlyle, F. M. David, R. Farivar, and R. H. Campbell, "Bootjacker: Compromising computers using forced restarts," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, October 2008, pp. 555–564.
[49] Y. Wen, X. Man, K. Le, and W. Shi, "Forensics-as-a-service (faas): Computer forensic workflow management and processing using cloud," in *Proceedings of the 4th International Conference on Cloud Computing, GRIDs, and Virtualization*, May 2013, pp. 208–214.
[50] S. Zawoad, A. K. Dutta, and R. Hasan, "Seclaas: Secure logging-as-a-service for cloud forensics," in *Proceedings of the 8th ACM Symposium on Information, Computer and Communications Security*, May 2013, pp. 219–230.
[51] S. Alqahtany, N. Clarke, S. Furnell, and C. Reich, "A forensic acquisition and analysis system for iaas," *Cluster Computing*, November 2015.
[52] D. Birk and C. Wegener, "Technical issues of forensic investigations in cloud computing environments," in *Proceedings of the 6th IEEE International Workshop on Systematic Approaches to Digital Forensic Engineering*, May 2011, pp. 1–10.
[53] "CP-ABE Toolkit," [Online]. Available: http://acsc.cs.utexas.edu/cpabe/.
[54] H. Guo, Y. Li, and S. Jajodia, "Chaining watermarks for detecting malicious modifications to streaming data," *Information Sciences*, vol. 177, no. 1, pp. 281–298, 2007.
[55] "Xen project," [Online]. Available: https://www.xenproject.org/.
[56] "OpenSSL," [Online]. Available: https://www.openssl.org/.
[57] Intel Open Source Technology Center, "TPM2 software stack implmentation," [Online]. Available: https://github.com/01org/TPM2.0-TSS.
[58] "TrustedGRUB," [Online]. Available: http://sourceforge.net/projects/trustedgrub/.
[59] "SeaBIOS," [Online]. Available: http://code.coreboot.org/p/seabios/downloads/.
[60] "LibVMI Library," [Online]. Available: http://libvmi.com/.
[61] "Proverif," [Online]. Available: http://prosecco.gforge.inria.fr/personal/bblanche/proverif/.
[62] "Scyther," [Online]. Available: http://www.cs.ox.ac.uk/people/cas.cremers/scyther/.
[63] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, May 2007, pp. 321–334.
[64] Internet Engineering Task Force, *The Transport Layer Security (TLS) Protocol Version 1.3 (RFC 8446)*, 2018, [Online]. Available: https://datatracker.ietf.org/doc/rfc8446/?include_text=1.
[65] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: Cold-boot attacks on encryption keys," *Communications of the ACM*, vol. 52, no. 5, pp. 91–98, May 2009.
[66] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for CPU based attestation and sealing," in *Proceedings of the 2nd Workshop on Hardware and Architectural Support for Security and Privacy*, June 2013.
[67] H. Raj, S. Saroiu, A. Wolman, R. Aigner, J. Cox, P. England, C. Fenner, K. Kinshumann, J. Loeser, D. Mattoon, M. Nystrom, D. Robinson, R. Spiger, S. Thom, and D. Wooten, "fTPM: A software-only implementation of a TPM chip," in *Proceedings of the 25th USENIX Security Symposium*, August 2016, pp. 841–856.
[68] Y. Swami, "Intel SGX remote attestation is not sufficient," in *Proceedings of the 2017 Black Hat USA*, July 2017.

APPENDIX A
PROOF OF LEMMA 1

Let's assume that the probability that the maliciously deleted element is a boundary element is $1/m$, while the probability that the deleted element is not a boundary element is $1 - 1/m$. Let's consider two conditions below:

Condition 1. The maliciously deleted element is a boundary element. In this case, the current group will be merged with its adjacent next group. Since the average length of a group is M, the probability that the current is merged into the next group without being detected is $1/2^M$.

Condition 2. The maliciously deleted element is not a boundary element. Since the deletion can happen at any one of the $M-1$ positions, the probability that verifying a group without detecting any error is $1/2^{M-1}$.

Combining these two conditions, the false negative of this case is:

$$\begin{aligned} FN_{deletion} &= \frac{1}{M}\cdot\frac{1}{2^M} + \frac{M-1}{M}\cdot\frac{1}{2^{M-1}} \\ &< \frac{1}{M}\cdot\frac{1}{2^{M-1}} + (1-\frac{1}{M})\cdot\frac{1}{2^{M-1}} \\ &= \frac{1}{2^{M-1}} \end{aligned} \tag{3}$$

Finally, from the Definition 1, 2, and 3, (3) follows. □

APPENDIX B
PROOF OF OF LEMMA 2

Let's assume that the probability that the maliciously inserted element becomes a boundary element is $1/m$, while the probability that the inserted element does not become a boundary element is $1 - 1/m$. Let's consider two conditions below:

Condition 1. The maliciously inserted element becomes a boundary element. In this case, the inserted element will have a equal probability to be inserted in $M+1$ positions. Then, we have two sub-conditions: 1) the element is inserted

into the first $L-1$ positions and divides one existing group into two. Since L is the lower bound of a group, two divided groups will be combined into one group and make the average length of the new group to be $M+1$; 2) the element is inserted into the latter $M-L+1$ positions, then the existing group will be split into two groups, each contains at least L elements.

Condition 2. The maliciously inserted element does not become a boundary element. In this case, the inserted element will have a equal probability to be inserted in $M+1$ positions, the probability that verifying a group without detecting any error is $1/2^{M+1}$.

Combining these two conditions, the false negative of this case is:

$$\begin{aligned} FN_{modification} &= \frac{1}{m}\left(\frac{L-1}{M+1}\cdot\frac{1}{2^{M+1}}+\frac{M-L+2}{M+1}\cdot\frac{1}{2^{L-1}}\right) \\ &\quad +\left(1-\frac{1}{m}\right)\cdot\frac{1}{2^{M+1}} \\ &\leq \frac{1}{m}\left(\frac{L-1}{M+1}\cdot\frac{1}{2^{L-1}}+\frac{M-L+2}{M+1}\cdot\frac{1}{2^{L-1}}\right) \\ &\quad +\left(1-\frac{1}{m}\right)\cdot\frac{1}{2^{L-1}} \\ &= \frac{1}{2^{L-1}} \end{aligned} \tag{4}$$

Finally, from the Definition 1, 2, and 3, (4) follows. □