


RESEARCH

Open Access



LogDrive: a proactive data collection and analysis framework for time-traveling forensic investigation in IaaS cloud environments

Manabu Hirano^{1*} , Natsuki Tsuzuki¹, Seishiro Ikeda¹ and Ryotaro Kobayashi²

Abstract

This paper presents the LogDrive framework for mitigating the following problems of storage forensics in Infrastructure-as-a-Service (IaaS) cloud environments: volatility, increasing volume of forensic data, and anti-forensic attacks that hide traces of incidents in virtual machines. The proposed proactive data collection function of virtual block devices mitigates the problem of volatility within the cloud environments and enables a time-traveling investigation to reveal overwritten or deleted evidence files. We employ a sector-hash-based file detection method with random sampling to search for an evidence file in the record of the write logs of the virtual storage. The problem formulation, the investigation context, and the design with five algorithms are presented. We explore the performance of LogDrive through a detailed evaluation. Finally, security analysis of LogDrive is presented based on the STRIDE (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege) threats model and related work. We posted the source code of LogDrive on GitHub.

Keywords: Cloud forensics, Surveillance, Anti-forensics, Hypervisor, Virtual machine monitor, Parallel distributed processing

Introduction

Our everyday lives depend on reliable and trusted cloud services located at distant data centers. For example, many products can be purchased from Internet-based marketplaces, and cloud-based office software and teleconferencing applications enable users to collaborate with colleagues in remote places. The systems of many Internet-based marketplaces and recent cloud-based applications are deployed in Infrastructure-as-a-Service (IaaS) cloud data centers [1].

This paper first describes three problems of cloud forensics: volatility, increasing volume of forensic data, and anti-forensic attacks. Reliable and efficient forensic investigation of the current cloud environments is essential as the volume of activity within them increases. The next section presents two key technologies for mitigating the three problems: a wayback machine for a time-traveling

investigation and sector-hash-based file detection with random sampling. The notation, the problem formulation, and the investigative context are presented. Then, LogDrive, the proactive data collection and analysis framework for IaaS cloud environments, is proposed. The framework can be employed in any organization where an investigating authority or an administrator has control over the user infrastructure. This paper presents an implementation strategy and a performance evaluation. Finally, a discussion of the framework from the perspective of security, performance, and size of logs is presented.

Contribution

This is an extended version of our previous work [2, 3] that presented the concept of a novel write-logging mechanism for virtual block devices in Infrastructure-as-a-Service clouds. Here, we extend the previous work by introducing sector hashing with random sampling for processing a large amount of forensic data. Figure 1 shows the overall architecture of the proposed LogDrive framework. The LogDrive framework preserves all write operations

*Correspondence: hirano@toyota-ct.ac.jp

¹National Institute of Technology, Toyota College, 2-1 Eisei, 471-8525 Toyota, Japan

Full list of author information is available at the end of the article

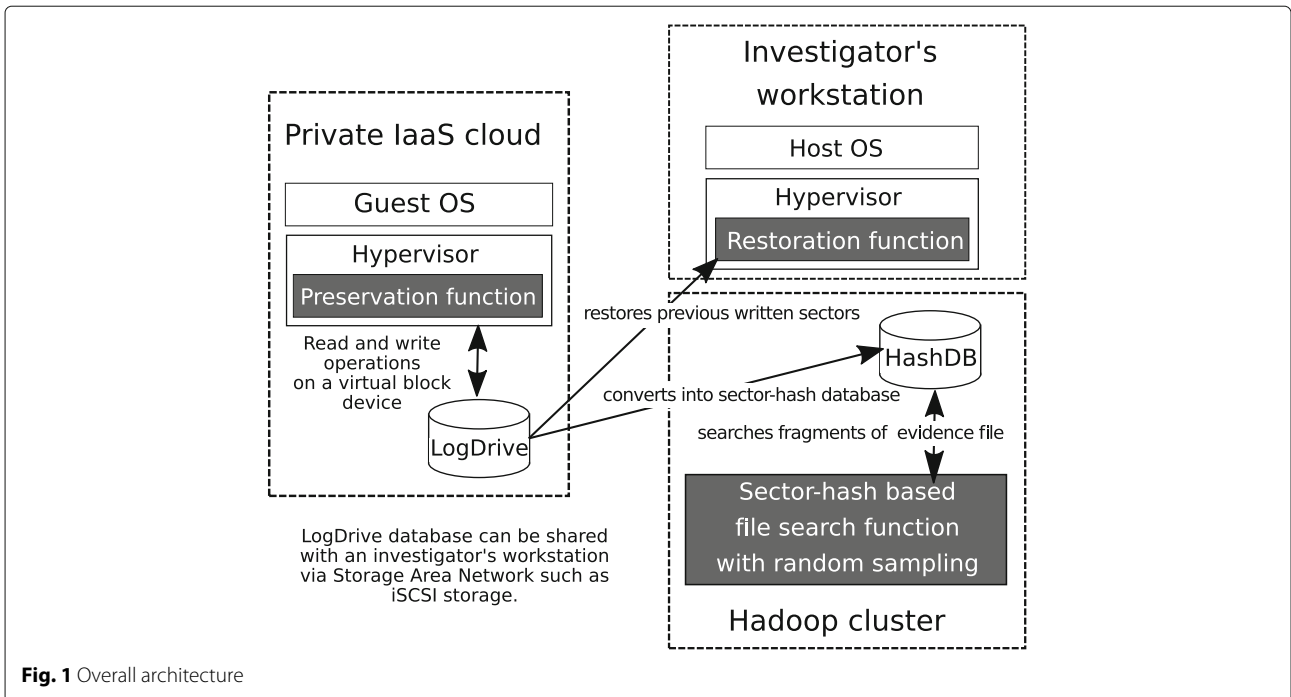


Fig. 1 Overall architecture

on a virtual block device of an customer’s guest operating system. Then, the LogDrive database is converted into a sector-hash database (i.e., HashDB) to search fragments of evidence files in the past write operations. Finally, an investigator’s workstation restores an virtual block device at an arbitrary point in time from the LogDrive database to collect evidence files in a past incident.

Furthermore, we extend the previous work by introducing the formal notation, the problem formulation, the investigative context, the detailed design with five new algorithms, and security analysis based on the STRIDE (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege) threats model [4]. We explore the performance of LogDrive through a more detailed evaluation.

We posted the source code of LogDrive on GitHub. Therefore, users can test the authors’ implementation for proactive data collection on users’ IaaS infrastructure. Time-series logs taken by LogDrive can be used to analyze disk access patterns of applications. For example, a user can use output of LogDrive to model disk access patterns of ransomware using machine learning technique. LogDrive includes data-processing functions using Hadoop MapReduce so that users will be able to process large amount of data easily.

Problems of volatility

This section shows that the problem of volatility in cloud environments negatively impacts forensic procedures and why proactive data collection is needed.

The Digital Forensic Research Workshop (DFRWS) released a forensic framework that consists of identification, preservation, collection, examination, analysis, presentation, and decision [5]. The NIST SP800-86 defines the following four forensic steps: collection, examination, analysis, and reporting [6]. Recently, Elyas et al. proposed a refined digital forensic readiness framework [7]. In contrast to investigations in conventional stand-alone computers, forensic investigation in a cloud environment is not easy. Ruan et al. proposed a definition of cloud forensics as a mixture of traditional forensic techniques and applications in cloud environments in 2011 [8]. Martini and Choo presented a conceptual digital forensic framework for cloud computing in 2012 [9]. Rahman et al. proposed a forensic-by-design framework for cyber-physical cloud systems [10].

Although most frameworks have preservation and isolation steps of the evidence, Garfinkel stated that performing these basic forensic steps is technically difficult in cloud environments [11]. Typical IaaS cloud environments virtualize physical hardware resources as virtual storage, virtual memory, virtual CPU, and other virtual devices, and the cloud provider leases them to customers in a “pay-as-you-go” manner [12]. Reilly et al. noted that conventional search and seizure procedures are impractical because digital evidence in cloud data centers is scattered in multiple physical machines that are shared by many customers simultaneously [13]. Taylor et al. stated that digital evidence in virtualized environments will be more ethereal [14].

Another important aspect in many cases of cloud forensics is legal issues. As described above, the IaaS cloud customers share the same physical hardware simultaneously (e.g., a customer runs a virtual machine on a physical server while another customer runs a virtual machine on the same physical server). Although customers rent virtualized resources for a predetermined period, they usually do not have ownership of the physical hardware from the perspective of their legal contracts. Grispos et al. discussed how the traditional preservation and collection model is impacted by the cloud [15]. They stated that imaging entire physical drives is impractical and that partial imaging faces legal challenges. To solve legal issues regarding cloud investigations, Ruan et al. proposed creating a Service Level Agreement (SLA) that enables cloud forensic investigations. The proposed Service Level Agreement includes proactive forensic preparation, data collection, incident response and recovery, time synchronization, cross-border jurisdiction, and multi-tenant data issues [16]. Although they noted that proactive forensic data collection on cloud environments is important, such professional tools are not yet available.

The problem is that investigative authorities cannot obtain physical disks even if they have warrants because the disks are shared by multiple tenants. Moreover, the virtual block device is released when the contract is finished. Therefore, cloud providers need a mechanism to keep records of previous virtual disks for a certain period of time for a future possible investigation or incident response.

In this paper, we assume that providers and customers sign the Service Level Agreement (SLA) that enables the future use of the customer's previous records of disks before starting the contract so that the provider can enforce the proactive data collection function on the customers. In addition, customers' data might be present in a data center that is in a different country that has its own law regarding the ownership of that data. Nanda and Hansen pointed out the difficulties of multiple jurisdictions in their proposed Forensic-as-a-Service (FaaS) architecture [17], an infrastructure-level forensic support from cloud providers. The FaaS architecture includes a procedure of legal teams to fetch the data from different jurisdiction. Our proposal also needs a procedure of legal teams to assure that the locations of virtual machines and their write logs are in the countries that have appropriate jurisdiction.

The proactive data collection function might not be preferred by providers and customers who do not care about auditability, but it will be preferred by providers and customers who need auditability to ensure regulatory compliance for their business operations. If a customer does not agree with the SLA, the provider can reject the

customer's use of the cloud service. It is dependent on the operational and business policies of the providers.

Problems of increasing volume of data

In the "pay-as-you-go" cloud environments, virtual storage is leased to customers. Such cloud environments generate a large amount of data over a short period. Public cloud services provide large amount of disk space to users and employ some techniques to save their storage space. File hosting services such as DropBox employ data deduplication as compression technique to save their storage capacity. The term *data deduplication* refers to techniques that store only a single copy of redundant data, and provide links to that copy instead of storing other actual copies of the data [18]. On the other hand, Amazon Elastic Compute Cloud (EC2), one of the major players of public IaaS cloud service, does not employ data deduplication techniques for their virtual storage (i.e., Elastic Block Store; EBS) because of its latency and degradation of customer experience. For example, users of Amazon EC2 can employ ZFS [19], a copy-on-write file system with built-in compression function, on their own EBS to reduce the user's storage space.

In addition to the large amount of data generated by customers, if a provider employs proactive data collection for a future investigation, an investigator needs to process the large amount of collected forensic data when an incident occurred. Here, a brief review of previous studies on how to process such a large amount of forensic data is presented.

In 2004, Richard and Roussev showed that a regular expression search of a 6 GB disk image can be performed 18 to 89 times faster by using 8 machines with 1 GB RAM in a 1000BASE-T network [20] in a laboratory setup. They avoided the disk bound problem by storing all files in each machine's RAM. MapReduce [21] is a more scalable solution to process a large amount of data using multiple computing nodes. Roussev et al. proposed MPI MapReduce [22] for CPU-bound processing in a forensic analysis. They employed the Message Passing Interface (MPI) and conventional Network File System (NFS) instead of using the Hadoop Distributed File System (HDFS). Although MPI MapReduce provides linear scaling with respect to the number of CPUs and CPU speed, the paper did not present an evaluation of a cluster composed of more than three machines.

Garfinkel noted that parallel processing is not suitable for conventional industry standard forensic tools based on the "Visibility, Filter, and Report" model [11]. This model depends on the tree structures of file systems. If forensic data are distributed to computing nodes for parallel processing, the system will not be able to identify each segment's meaning of the file system level. Ayers stated that the current industry standard tools, such as EnCase and FTK, are the first generation computer forensic tools.

He defined second generation forensic tools as tools that include parallel processing for a massive amount of data [23].

One method to tackle the problem is the combination of parallel processing without file system information and conventional standalone forensic tools based on the file system information. In this paper, we employ sector hashing with random sampling [24–27, 42] to search for an evidence file in a large amount of write logs by using a MapReduce cluster. Moreover, we propose the restoration function of a previous virtual block device at an arbitrary point in the past so that investigators will be able to perform detailed analysis of the restored disk using conventional forensic tools based on the file system information.

Problems of anti-forensics

Harris stated that anti-forensics is any attempt to compromise the availability or usefulness of evidence for the forensics process [28]. The volatility of virtual environments of clouds has a negative impact on preserving the traces of such anti-forensic attacks. Cloud environments make erasing attackers' traces easy. However, if a cloud provider employs proactive data collection as described earlier, the provider can keep track of the contents of the customers' virtual disks. The recorded logs will help reveal some types of anti-forensic attacks. This section presents what type of anti-forensic attacks can be detected by proactive data collection. Please note that we describe this process not to prevent anti-forensic attacks but to preserve the traces of the attacks for future investigation.

First, we classify anti-forensic attacks in the cloud into two types: **type 1** attacks are launched inside virtual machines. For example, a customer could delete evidence of his or her crime or could overwrite the date and time of important access logs to evade capture. An outside attacker or malware who intruded into a virtual machine could also delete traces of an intrusion. **Type 2** attacks are launched outside virtual machines, for example, from an IaaS cloud provider's management plane. **Type 2** attacks include direct attacks on forensic software.

Type 1 attacks: In **type 1** attacks, a customer or an outside attacker inside the virtual machine deletes, destroys, or manipulates his or her traces of files of incidents (e.g., access logs, rootkits, or temporary files) or hides evidence in slack space or other hiding places in the file systems [29]. The conventional collection method of IaaS clouds, which is taking snapshots, cannot preserve traces of **type 1** attacks because the evidence may be removed or tampered with before acquisition of the virtual machine's snapshot.

In addition, Garfinkel noted that anti-forensic attacks can exploit computer forensic tools by inputting data that are not properly validated by generating a massive amount

of data or bypassing fragile heuristics (e.g., file detection by using a "magic number" of the first few bytes of files) [29]. Kessler reported malicious programs that alter file signatures and flip bits in order to evade file-hash detection in forensic software [30]. These attacks are launched in virtual machines to hide evidence from forensic software.

Type 2 attacks: In **type 2** attacks, cloud providers can plant false evidence on a customer's virtual machine image from outside virtual machines, e.g., from their cloud management plane. Schneier and Kelsey proposed secure audit logs to support computer forensics in 1999 [31]. Zawoad et al. extended Schneier and Kelsey's previous work to Secure Logging-as-a-Service (SecLaaS) to prevent attacks on logs in cloud environments by publishing proofs of past logs on public web sites or on Rich Site Summary (RSS) feeds periodically [32]. The limitation of both works [31, 32] is that the system cannot prevent the falsification of logs before creating the proof of the genuine logs. **Type 2** attacks are also launched on forensic software. For example, an attacker might disable a forensic agent program on his or her machine to hide evidence.

As described earlier, proactive data collection for future investigation enables a time-traveling investigation by using recorded logs. Even if someone launches **type 1** attacks inside his or her virtual machine, the traces of the attacks can be verified by using the records of disk changes. Please note that our proposal preserves traces of **type 1** attacks, but it does not prevent **type 2** attacks. We have no silver bullet that prevents all types of attacks. We discuss possible measures against these attacks in the "Discussion" and "Related work" sections.

Preliminaries

This section introduces two key technologies of the Log-Drive framework: a wayback machine for restoring virtual block storage at an arbitrary point in time and sector hashing for searching an evidence file from a wayback machine. The wayback machine mitigates the problems of volatility in IaaS clouds and **type 1** anti-forensic attacks. The sector hashing mitigates the problems of increasing volume of forensic data using parallel distributed processing and random sampling.

Wayback machine

A wayback machine is a mechanism for recording the whole write history for future investigations and for restoring the virtual block device at an arbitrary point in the past to discover evidence of incidents. This section presents the related work of the log-structured file system and copy-on-write system that can be used for creating the wayback machine.

The pioneering research on versioning and backup mechanisms was conducted by Rosenblum and Ouster-

hout in the early 1990s [33]. They presented the Log-structured File System. The idea of the original Log-structured File System is to collect a large amount of writes in a page cache and to flush the file system data onto a disk sequentially in a single large I/O for high throughput. This behavior is similar to appending records to the tail of a log file.

After the pioneering work [33], Cornel et al. employed the Log-structured File System for their user-level versioning file system called Wayback by using the FUSE framework [34]. Strunk et al. also employed the Log-structured File System for their self-securing storage called S4 to prevent attackers from tampering with data [35]. Morrey and Grunwald applied the Log-structured File System design to their time-traveling disk called Peabody [36]. Peabody is implemented as an iSCSI target instead of a file system. Recently, the log-structured design has been employed for file systems of non-volatile memories (NVMs) [37, 38] because of its high throughput and space efficiency. The design of the Log-structured File System is also suitable for the latest cloud-based storage services. For example, Vrable et al. presented the BlueSky network file system backed by the cloud storage service of Amazon S3 [39]. They noted that the log-structured design on cloud storage services lowers costs and improves performance.

Another candidate for versioning and backup systems for block devices is copy-on-write. Copy-on-write systems save a data block in another place when the write operation is issued, and then, the original data block is overwritten by a new data block. The important detail is that copy-on-write systems save only the oldest data blocks until a new snapshot is created. Therefore, they cannot hold the complete history of write operations. Current virtual machine monitors or hypervisor software use a virtual disk format such as QEMU Copy-on-write version 2 (QCOW2) [40] and Microsoft's virtual hard disk (VHD) [41]. They do not employ log-structured design but rather copy-on-write design for creating space-efficient snapshots of virtual block devices.

Although the copy-on-write system restores a previous block device from a snapshot, the log-structured system can restore a past state of a virtual block device from an arbitrary point in time. For this reason, the copy-on-write system cannot reveal **type 1** anti-forensic attacks in certain circumstances. For example, an attacker in a virtual machine can create files and delete the files between two distant checkpoints in time. On the other hand, the log-structure system records all write operations sequentially; therefore the traces of **type 1** anti-forensic attacks can be restored for the appropriate time in the past. In this paper, we employ the log-structured design for recording all written sectors on a virtual block device.

Sector-hash-based file detection

Sector hashing is a technique to check the existence of evidence files on a suspect's media without reference to the underlying file system. This section shows the related work on sector hashing and presents how to apply sector hashing to a large amount of write logs that are generated from a wayback machine.

The pioneering work on sector hashing [24, 42, 43] was conducted by Garfinkel et al. They proposed using cryptographic hashes of small data blocks to identify file fragments and entire files. In contrast to file hashing, sector hashing can find the fragments or remains of a piece of a sector from partially deleted or corrupted physical disks.

Young et al. [26] examined the validity of sector hashing on user-generated contents using three datasets (Govdocs [44], OpenMalware2012, and 2009 NSRL RDS) that include word processing files, photos, and videos. For example, they reported that 98.93% of the 512-byte sector hashes in the Govdoc dataset, which is a collection of documents from .gov web sites, are distinct. Their study indicated that sector hashing is effective for searching user-generated files, for example, photographs taken by pedophiles.

Garfinkel et al. [26, 42] described a performance improvement method for sector hashing for a rapid and largely automated analysis. They proposed random sampling of sectors instead of searching every sector on drives. Equation 1 shows the well-known "urn problem" that describes the probability of pulling red beans out of an urn that contains a mix of evenly distributed red and black beans. The probability p of not finding even a single red bean in n draws is as follows.

$$p = 1 - \prod_{i=1}^n \frac{((N - (i - 1)) - T)}{(N - (i - 1))} \quad (1)$$

Here, N is the total number of beans, T is the number of red beans, and $N - T$ is the number of black beans. In our context, an investigator searches T target sectors (red beans) obtained by splitting a target file out of an urn that contains N sectors (red and black beans) in n random samplings. Thus, p is the probability that at least one sector of T target sectors will be found in the n random samplings. For example, an investigator searches the sectors of a 100 MiB video file out of a set of 512-byte sectors that are obtained from many drives that contain 100 TiB of data in total. If the 5,000,000 sectors are randomly sampled, $p \approx 0.9915$ ($N = 214,748,364,800$, $T = 204,800$, and $n = 5,000,000$). The result means that there is a greater than 99% chance that at least one sector of the 100 MiB video file will be found in the 5,000,000 random samplings.

Sector hashing has limitations in the following cases. First, it cannot detect embedded files, such as photographs in the latest Microsoft Office documents (e.g., .docx and .pptx files), because the contents are compressed in a ZIP archive file. To find embedded objects in the latest Office documents, a mechanism to extract their contents before applying sector hashing is needed. Additionally, sector hashing is not supported for encrypted files or encrypted file systems. For encrypted file systems, it is necessary to decrypt the file system by mounting the media using an appropriate decrypting driver [24]. Although sector hashing has these limitations, it provides efficient parallel search functions. Sector hashing can also be applied to fragment detection of memory forensics.

The aim of this research is to extend the previous research on sector hashing for media [24, 26, 42, 43], such as HDD or SSD, by integrating this method with the wayback machine described in the previous section. Thus, sector hashing is used for searching fragments of a target file from past write logs of virtual block devices.

Notation

This paper uses the following notation.

C , A , P , and I - a customer, an outside attacker, a cloud provider, and an investigator, respectively.

VM_c - a virtual machine of a customer C .

E - evidence on a customer's virtual machine.

VBD_c - a customer C 's virtual block device, denoted as $VBD_c = \{s_0, s_1, s_2, \dots, s_{n-1}\}$, where n is the number of sectors. In the collection phase, a VBD_c is accessed through a C 's guest operating system. In the analysis phase, any past state of VBD_c can be reconstructed from a $LogDrive_c$ database on an investigator I 's forensic workstation. Therefore, $VBD_c \in LogDrive_c$.

$VBD_{c,t}$ - customer C 's restored virtual block device at the point in time t . An investigator I can access $VBD_{c,t}$ in the read-only mode.

s_i - a sector that is written on an i th sector of VBD_c . i represents a Logical Block Address on VBD_c . A Logical Block Address (LBA) is the value used to reference a logical sector of HDDs or SSDs [45].

$LogDrive_c$ - the database of recorded sectors that are written on VBD_c , denoted as $LogDrive_c = \{Log_0, Log_1, Log_2, \dots, Log_{n-1}\}$, where n is the number of sectors in VBD_c . A $LogDrive_c$ database is created for each customer C 's VBD_c .

Log_i - logs of past written sectors at LBA i , denoted as $Log_i = \{Log_{i,0}, Log_{i,1}, Log_{i,2}, \dots, Log_{i,m-1}\}$, where m is the number of writes on LBA i . $Log_{i,j}$ means the log of the $(j + 1)$ th written sector at LBA i of VBD_c .

$Log_{i,j}$ - each log is denoted as $Log_{i,j} = \{\text{timestamp in seconds } t_s, \text{ timestamp in nanoseconds } t_{ns},$

$\text{byteOffsetOfPreviousLog}, \text{size}, s_{i,j}\}$. The proposed system uses the UNIX timestamp that is defined as the number of seconds since 00:00:00 UTC, 1st January, 1970. size represents the size of a sector $s_{i,j}$. The maximum value of size is 4096 in our current implementation. The data structure of Log_i is a singly linked list of a set of $Log_{i,j}$. The newly written block is inserted in the head of the list. When the system needs to read the previous sector $s_{i,j-1}$, it traverses the list Log_i by using $Log_{i,j}.\text{byteOffsetOfPreviousLog}$. The value of $\text{byteOffsetOfPreviousLog}$ holds a byte offset of the previous $Log_{i,j-1}$ that is calculated from the beginning of the $LogDrive_c$ database file. If the value of $\text{byteOffsetOfPreviousLog}$ is null, then there are no more previous sectors.

$LogDrive.index[i]$ - this value holds a byte offset of the latest written $Log_{i,m-1}$. This value points to the head of a singly linked list Log_i . The byte offset is calculated from the beginning of the $LogDrive_c$ database. If the value is null, then any sector is not yet written at LBA i .

$LogDrive.disk_size$ - total size of a VBD_c in bytes.

$LogDrive.sector_size$ - internal sector size of a VBD_c in bytes. LogDrive uses 4096 in LogDrive. Please note that LogDrive provides virtual block devices of 512 byte sector size (logical and physical) to virtual machines.

$HashDB$ - key-value pairs for sector-hash-based file search.

$H(s)$ - hash value of sector s .

N , T , r , and p - total number of sectors in $HashDB$, total number of sectors in a file F , sampling rate, and probability that at least one sector will be found, respectively.

Problem formulation

The threat model of this paper involves the following four entities: cloud providers P , cloud customers C , outside attackers A , and cloud investigators I , as shown in Fig. 2. This paper assumes that cloud customers C and outside attackers A are not trusted, while cloud providers P and investigators I are trusted. Investigators I can be law enforcement agents, incident response teams, or other internal groups in the human resources or legal departments.

The crimes or incidents of the threat model include possession of child pornography, leakage of classified information, cyber espionage, fraud, ransomware, DDoS attacks, or terrorist activities. A cloud customer C commits a crime on his or her virtual machine VM_c . Additionally, an attacker A , who is not the customer of the cloud provider P , commits crimes on customer C 's virtual machine VM_c , for example, by sending malware remotely to VM_c or by intruding into VM_c illegally. Although it

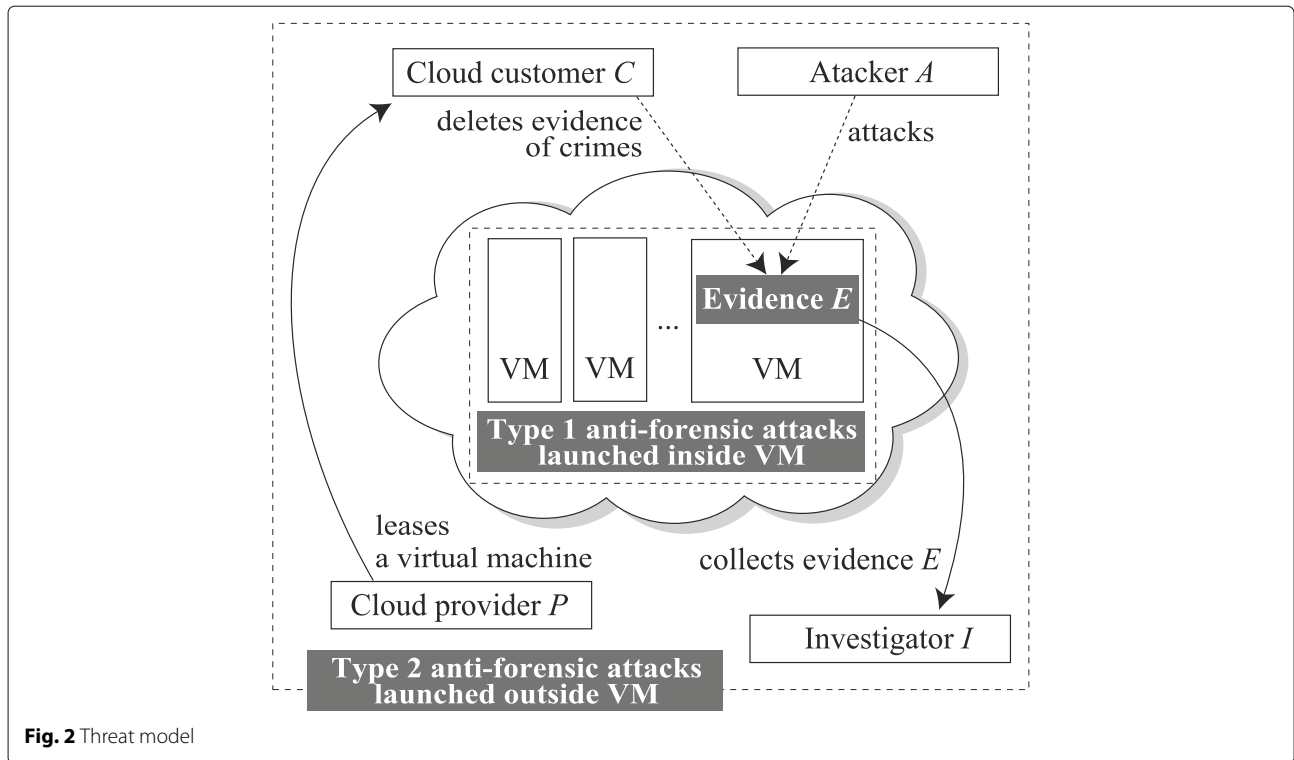


Fig. 2 Threat model

is impossible to prevent every incident, it is important to securely retain evidence even if **type 1** anti-forensic attacks (e.g., tampering with important logs or legal documents and hiding traces of crimes) are launched on VM_c . From the perspective of threat modeling, our model is directly involved with “tampering with data” and “repudiation” of the STRIDE threats model [4]. Please note that we discuss how to detect false evidence created outside VM_c (i.e., **type 2** attacks) in the “Related work” section.

If incidents occurred in the virtual machine VM_c , an investigator *I* has to collect and analyze digital evidence *E* from cloud provider *P*’s physical storage devices. However, the investigator *I* cannot seize the physical storage devices because they may have data owned by other customers who are not directly related to the crimes. Moreover, in most cases, the cloud provider *P* does not need to keep customer *C*’s VM_c for a long time after the end of the contract. A part of physical storage may be recycled to other customers’ services immediately. Consequently, the investigator *I* can no longer collect the digital evidence *E*.

Investigation context

The proposed system assumes the following investigation context. First, a provider *P* and a customer *C* agree with the Service Level Agreement (SLA) that permits proactive data collection from the customer’s virtual machines VM_c in the provider’s IaaS cloud environments to allow

for future audits and investigations. At this point, the customer *C* has no intention of doing illegal or malicious activities. For example, a customer *C* will be able to sign the SLA that permits proactive data collection for monitoring the company’s employees who might commit crimes in the future on their virtual machines (e.g., tampering with important legal documents and access logs to evade capture). If the provider *P* and the customer *C* agree to the SLA, the provider begins to record the customer’s activities on virtual machines VM_c . Although the proposed system cannot be used to investigate a crime on a non-suspected or non-monitored customer’s virtual machine VM_c after the fact, it can be used to investigate a crime after starting surveillance based on the SLA between the provider *P* and the customer *C*.

The aim of the proposal is to reveal evidence of criminal activities and **type 1** anti-forensic attacks (i.e., attacks launched inside the virtual machine VM_c). Please note that the measures against **type 2** anti-forensic attacks (i.e., attacks launched outside the virtual machine VM_c) are described in the “Discussion” and “Related work” sections.

Design goal

For efficient and reliable proactive collection and analysis of forensic data in IaaS cloud environments, the proposed LogDrive framework should achieve the following design goals.

- 1). Efficient collection: the collection function should not affect customer C 's activities. We have to minimize the overhead of a surveillance mechanism on a virtual block device VBD_c and a VM_c .
- 2). Reliable collection: an attacker A or a malicious customer C should not be able to bypass or destroy the collection function. Moreover, the timestamps of the mechanism should be protected against A and C . Thus, the proposed framework must be executed inside a trusted region.
- 3). Wayback storage: an investigator I needs to restore a virtual block device $VBD_{c,t}$ at an arbitrary point in time t to analyze the past state of the customer C 's VBD_c . The restored $VBD_{c,t}$ needs to be accessed as a normal block device via an operating system in read-only mode. This enables an investigator I to analyze the restored $VBD_{c,t}$ by using conventional forensic software, for example, EnCase or Forensic Tool Kit (FTK). For this purpose, the read throughput of the restored $VBD_{c,t}$ must be practical.
- 4). Rapid search of an evidence file in the wayback storage: an investigator I needs to find an evidence file F , which correlates with certain incidents, from the $LogDrive_c$ database across time. The $LogDrive_c$ database holds a large amount of sectors that were recorded in the past. For a rapid and efficient search, the system must support parallel distributed processing.
- 5). Cost efficiency: we need to develop a forensic framework that achieves maximum effectiveness by using affordable hardware available to most investigative organizations. The source code of the software should be provided to forensic communities.

Design

The LogDrive framework consists of the following three functions: proactive data collection, file search in a LogDrive database, and restoration from a LogDrive database.

Proactive data collection function

Figure 3 shows the system model of the collection function. The collection function monitors each written sector s on the virtual block device VBD_c . The sectors s^* can be written by a customer C 's applications (e.g., web browsers, word processing software, or e-mail client) or by an attacker A 's malicious programs such as rootkits. LogDrive saves each data of a sector s_i with Logical Block Address (LBA) i , timestamp t , and *size* onto a $LogDrive_c$ database immediately. The value of *size* can be between one and 4096 (4 KiB). If customer C 's programs or attacker A 's malicious codes on VM_c read a sector s_i on the virtual block device VBD_c , LogDrive returns the latest written sector s_i that is retrieved from the $LogDrive_c$ database.

The $LogDrive_c$ database is a set of past written sectors on customer C 's virtual block device VBD_c . Figure 3 shows that a **type 1** anti-forensic attack deletes evidence (i.e., the file F consists of the three sectors s_6 , s_7 , and s_8) of the incident at t_1 by overwriting them with three new sectors at t_3 . The LogDrive framework records all written sectors of both the original file F at t_1 and the compromised file F' at t_3 in the $LogDrive_c$ database so that it can be used for wayback storage in a future forensic investigation.

File search function of the LogDrive database

Figure 4 shows how to generate $HashDB$ from $LogDrive_c$ and transfer it to the Hadoop distributed file system on a forensic cluster. $HashDB$ is a set of key-value pairs that consist of the hash values of each sector $H(s)$ as the key and each sector's timestamp t , LBA i , and *size* of the sector as the value. $HashDB$ is used for the sector-hash-based file search function.

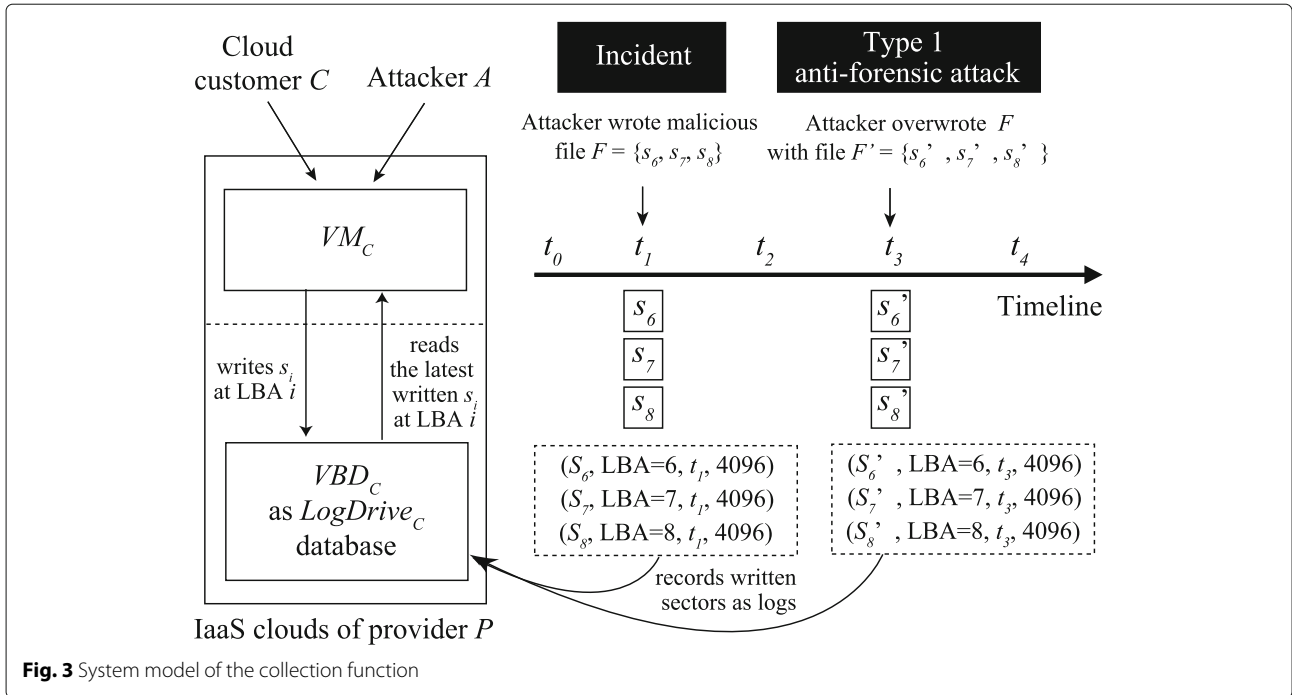
After creating a $HashDB$, an investigator I searches the evidence file F (e.g., photographs or videos of child pornography, evidence files of terrorist activities, malware, and rootkit) from the $HashDB$ by using sector hashing. The evidence file F is first sent to a forensic cluster; then, the search function splits the file F into 512-byte sectors and searches for $F = \{s_0, s_1, \dots, s_{n-1}\}$ in the $HashDB$, where n is the number of sectors of evidence file F . The results of the query are pairs of timestamp t , LBA i , and *size* of each sector. Thus, the output is the creation time of the sector s and its written location and size on the virtual block device. No output indicates that the evidence file F was not written on the monitored virtual block device in the past. Even if the evidence file F was deleted or was tampered with by a customer C or an attacker A before an investigation (i.e., **type 1** anti-forensic attack), an investigator I can search the creation time of the evidence file F from $HashDB$.

The LogDrive framework records all written sectors that include the file system's metadata in addition to the contents of files. Therefore, the system can search sectors not only from files but also from hidden data in the slack space or other unused space of file systems. In addition, the investigator I can search the evidence file F in a set of $HashDB$ that are generated from potential candidates of multiple virtual machines.

Please note that the proposed system model assumes that an investigator I already has an evidence file F that is found from other sources such as a suspect's thumb flash drive or an e-mail. For now, automatic identification of an evidence file F is outside the scope of this paper.

Restoration function of the LogDrive database

After identifying the creation time of the evidence file F using the file search function, the investigator I can restore a virtual block device $VBD_{c,t}$ at an arbitrary point in time



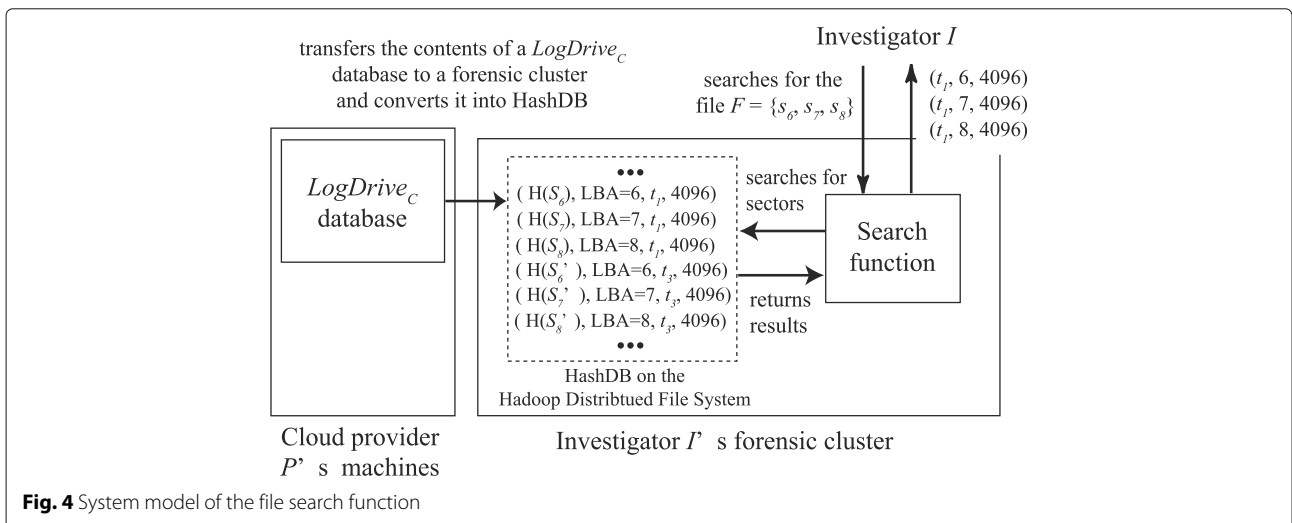
t for further analysis, where $t_0 \leq t \leq t_{n-1}$. The timestamp of the first written sector on the VBD_c is denoted as t_0 , and the timestamp of the last (nth) written sector on the VBD_c is denoted as t_{n-1} .

Figure 5 shows how the investigator I restores the virtual block device VBD_{c,t_2} that includes the evidence at t_1 before being overwritten by the anti-forensic attack at t_3 . The investigator I first specifies t_2 , and the system restores the past state of VBD_c . Then, the investigator I can read any sector from the restored virtual block device VBD_{c,t_2} in read-only mode. The investigator I can apply conventional forensic software, such as dd, EnCase, or FTK, to analyze VBD_{c,t_2} . The investigator

I can also inspect the detail of the anti-forensic attack at t_3 by comparing the restored VBD_{c,t_2} with the restored VBD_{c,t_4} .

4096-byte sector vs. 512-byte sector

The most common sector size in the current hard drive industry is 512 bytes or 4096 bytes. Although enterprise HDDs and SSDs are shifting to native 4096-byte sector size known as 4Kn, consumer HDDs or SSDs still use the 512-byte sector size. While current 512-byte sector drives employ 4096-byte sectors at the physical level, they emulate 512-byte sectors at the logical level for legacy software. Major operating systems (e.g., Windows 8 or



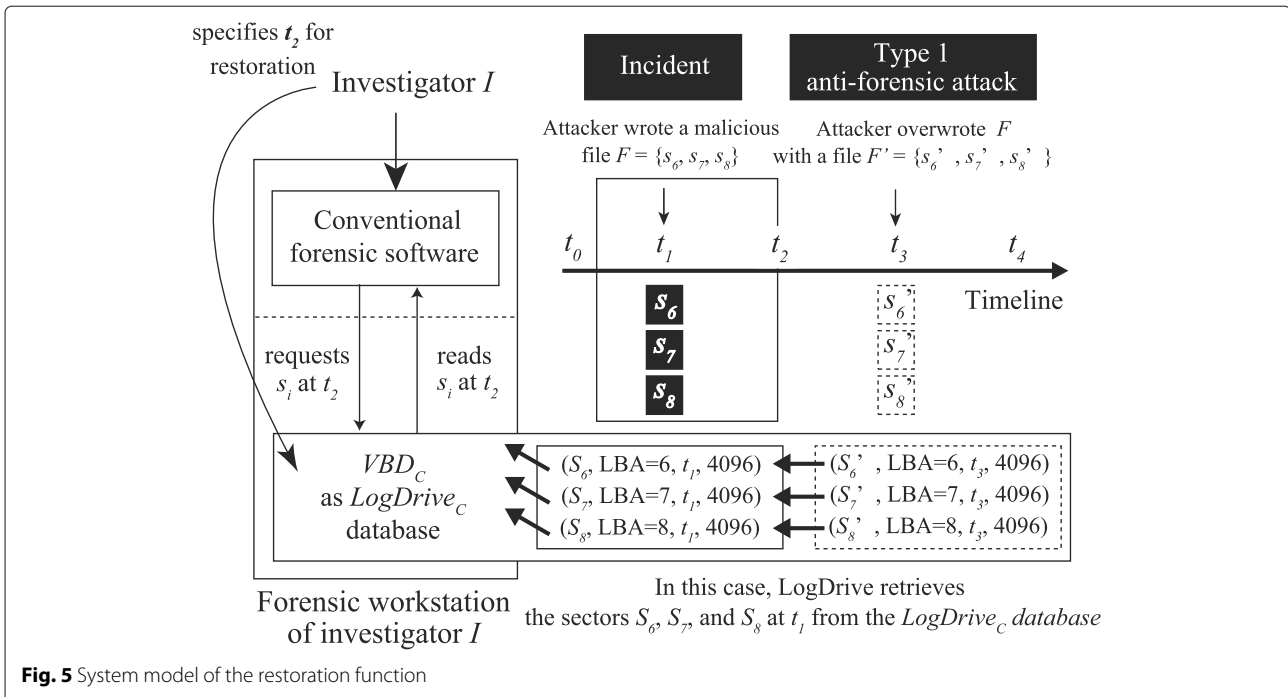


Fig. 5 System model of the restoration function

later and Linux kernel 2.6.31 or later) already support 4096-byte sectors.

Because of the existence of two sector sizes, LogDrive provides VBD of 512-byte sector size to virtual machines to support both legacy and new operating systems. Although LogDrive employs the 4096-byte sector size for internal structure of VBD , it employs 512-byte sector size for sector-hash-based file detection. LogDrive can change both sector sizes to 4096-bytes when many storage devices and software support the 4 KiB native setup.

Algorithms

This section presents five algorithms for proactive data collection, file search, and restoration of a LogDrive database.

Proactive data collection function

In the collection phase, the proposed LogDrive architecture provides normal input and output operations of a customer C 's VBD_C with the help of the underlying hypervisor software. While the system provides the normal I/O operations to C 's VM_C , the data collection function records all written sectors on a $LogDrive_C$ database. The $LogDrive_C$ database is a log-structured virtual block device.

Algorithm 1 shows how the LogDrive framework works during write operations on VBD_C . The write operation consists of creating $Log_{i,j}$ and appending it to the tail of the $LogDrive_C$ database. The process for writing a log to a $LogDrive_C$ database is executed for the group of $Log_{i,j}$,

Algorithm 1: WriteSectorAsLog

input : A written 4 KiB sector s , LBA i , current UNIX timestamp in seconds t_s , and current UNIX timestamp in nanoseconds t_{ns}
output: An updated version of a $LogDrive_C$ database

```

1 begin
2   foreach write request to  $VBD_C$  on  $VM_C$  do
3     previous  $\leftarrow LogDrive_C.index[i]$ ;
4      $Log_{i,j} \leftarrow createLog(t_s, t_{ns}, previous,$ 
5       sizeof( $s$ ),  $s$ );
6     appendLogToLogDrive( $Log_{i,j}$ ,
7        $LogDrive_C$ );
8      $LogDrive_C.index[i] \leftarrow sizeof(LogDrive_C)$ 
9       - sizeof( $Log_{i,j}$ ) + 1;
10  end
11 end
    
```

not for every $Log_{i,j}$, to minimize the overhead of frequent writes on the physical block device by using the buffering mechanism of the host operating system.

Algorithm 2 shows how the LogDrive framework works during read operations. The read operation of a sector is just retrieving the head element $Log_{i,m-1}$ of the list Log_i at LBA i . The system does not need to traverse the list structure of Log_i ; therefore, the read overhead is minimized. Both input and output 512-byte sectors from virtual machines are grouped into eight sectors and LogDrive interpret them as a single 4096-byte sector.

Algorithm 2: ReadSectorFromLog

```

input : An LBA  $i$ 
output: A 4 KiB sector  $s_{i,m-1}$ , where  $(m-1)$  is the
        latest written sector at LBA  $i$ 

1 begin
2   foreach read request to  $VBD_c$  on  $VM_c$  do
3      $Log_{i,m-1} \leftarrow$ 
       retrieveLog ( $LogDrive_c.index[i]$ );
4     if  $Log_{i,m-1} \neq null$  then
5       | return  $Log_{i,m-1}.s$ ;
6     else
7       | return SEC-
       | TOR_PADDED_WITH_ALL_ZEROS;
8     end
9   end
10 end

```

File search function of the LogDrive database

The file search function is composed of the following two steps: (1) creating *HashDB* and (2) searching *HashDB* for a file F . The creation of *HashDB* is performed using the MapReduce framework [21]. Please note that a *HashDB* is not an index such as a B-tree index but a set of hash values of each sector s for the sector-hash-based file search. The search process for a file F is executed using the MapReduce framework. When an investigator needs to triage an evidence file F from a large *LogDrive* database, the random sampling technique is applied.

We first tried to employ B-tree as an efficient and fast search algorithm. However, we finally decided to employ a simpler sequential search algorithm with statistical random sampling, because (1) B-tree algorithm is not suitable to be implemented on a scalable distributed parallel processing framework such as Hadoop MapReduce, (2) B-tree search algorithm is fast but the creation time of a B-tree index can be very long, especially when we need to create the index from a large *LogDrive* database, and (3) statistical random sampling is faster than B-tree algorithm at a certain confidence level.

Algorithm 3 shows how the LogDrive framework creates *HashDB* from the *LogDrive* database. A *LogDrive_c* database file is located on a local drive of a cloud provider P . The first half of the algorithm transfers a *LogDrive_c* database from the local drive to a Hadoop distributed file system (HDFS) [46] and converts the *LogDrive* database into the Hadoop's SequenceFile format. Hadoop's SequenceFile format is used to process binary key-value pairs. The last half of the algorithm creates key-value pairs; the key consists of eight hash values for each 512-byte sector created from a 4096-byte sector, and the value consists of the timestamp, the LBA, and the size

Algorithm 3: CreateHashDB

```

input : A  $LogDrive_c$  database file on a local file
        system, random sampling rate  $r$  (e.g.,
         $r \leftarrow 0.005$  for 0.5%)
output: A  $HashDB$  on a Hadoop Distributed File
        System (HDFS)

1 begin
2    $SECTOR\_SIZE\_FOR\_FILE\_SEARCH \leftarrow 512$ ;
3    $MAX\_LBA \leftarrow \left( \frac{LogDrive_c.disk\_size}{LogDrive_c.sector\_size} - 1 \right)$ ;
4   for  $i \leftarrow 0$  to  $MAX\_LBA$  do
5      $Log_{i,m-1} \leftarrow$ 
       retrieveLog ( $LogDrive_c.index[i]$ );
6     while  $Log_{i,j} \neq null$  do
7       |  $key \leftarrow$  concatenate ( $Log_{i,j}.ts, Log_{i,j}.t_{ns}$ ,
7       |  $i, Log_{i,j}.size$ );
8       |  $value \leftarrow log_{i,j}.s$ ;
9       | writeBinaryPairToHDFS ( $key, value$ );
10      |  $Log_{i,j} \leftarrow$  retrieveLog ( $Log_{i,j}$ .
10      |  $byteOffsetOfPreviousSector$ );
11      end
12    end
13    foreach createHashDBMapper ( $key, value$ )
13    do
14      |  $key\_for\_output \leftarrow null$ ;
14      |  $value\_for\_output \leftarrow null$ ;
15      | for  $k \leftarrow 0$  to
15      |  $\left( \frac{LogDrive.sector\_size}{SECTOR\_SIZE\_FOR\_FILE\_SEARCH} - 1 \right)$  do
16      | |  $sector_k \leftarrow$ 
16      | | splitIntoKthSector ( $value, k$ );
17      | |  $key\_for\_output \leftarrow key\_for\_output +$ 
17      | |  $hash(sector_k) + "$ ";
18      | |  $value\_for\_output \leftarrow key$ ;
18      | | end
19      | | if
19      | | allSectorsFilledWithConstant ( $key$ 
19      | |  $\_for\_output$ ) then
20      | | | continue;
20      | | end
21      | | sendToReducer ( $key\_for\_output$ ,
21      | |  $value\_for\_output$ );
22      | | end
23    end
24    foreach createHashDBReducer ( $key, value$ )
24    do
25      | if random ( $0,1$ )  $< r$  then
25      | | writeBinaryPairToHDFS ( $key$ ,
25      | |  $value$ );
26      | end
27    end
28  end
29 end
30 end
31 end

```

of the sector. For random sampling, the key-value pairs must be stored in evenly distributed orders. The MapReduce framework automatically sorts key-value pairs using keys (i.e., hash values) in the reduce steps. Therefore, LogDrive guarantees evenly distributed orders of *HashDB*. Please note that a 4096-byte sector filled with constant value (e.g., 0x00 or 0xFF) is ignored because the 4096-byte sector cannot be distinct [42].

For searching a large amount of *LogDrive* databases, the system employs a random sampling technique [47]. The sampling rate $r = n/N$ can be calculated using Eq. 1, where N is the total number of sectors (i.e., key-value pairs) in *HashDB*, T is the number of target sectors of a file F , and p is the probability that at least one sector of the file F will be found in the n random samplings. Because of the seriousness of the cases that law enforcement deals with, it is desirable to achieve a 99% confidence level [27]. We employ a 99% confidence interval for the reliability of the random sampling. The system prepares a set of sampled *HashDB* at different sampling rate for some representative file size (e.g., 1 MiB, 10 MiB, 100 MiB, 1 GiB) of search target in advance.

Algorithm 4 shows how the LogDrive framework searches *HashDB* for a file F . The system first splits F into 512-byte sectors and calculates a hash value for each sector. If the last sector is smaller than 512 bytes, the system pads the remainder of the sector with zeros before calculating its hash value. The system ignores a 512-byte sector filled with constant value (e.g., 0x00 or 0xFF) because the 512-byte sector cannot be distinct [42]. The current design employs the MD5 message-digest algorithm [48] that computes a 128-bit hash value. Anti-forensic attacks that use the weakness of the collision resistance in the MD5 algorithm are described in the “Discussion” section.

Restoration function of the LogDrive database

Algorithm 5 shows how the LogDrive framework reads a past sector s_i at the point in time t . The algorithm is used for restoring a past state of a customer C 's $VBD_{c,t}$ for the forensic analysis phase. The $VBD_{c,t}$ can be accessed in read-only mode. For example, an investigator can mount the restored $VBD_{c,t}$, access it via the forensic workstation's file system, make a disk image by using `dd`, and analyze it by using conventional forensic software.

Implementation

We implemented the system model and the algorithms on the hardware and software shown in Tables 1 and 2, respectively. The master server is used for collection, restoration, and the management of the Hadoop slave servers. The Hadoop slave servers are used for the sector-hash-based file search. Table 3 shows the configuration of the Hadoop MapReduce cluster.

Algorithm 4: SearchFile

```

input : A file  $F$ 
output: A list of  $(t_s, t_{ns}, \text{LBA } i, \text{size})$ 
1 begin
2    $\text{SECTOR\_SIZE\_FOR\_FILE\_SEARCH} \leftarrow 512$ ;
3    $\text{FILE\_SIZE} \leftarrow \text{sizeof}(F)$ ;
4    $\text{NUMBER\_OF\_SECTORS} \leftarrow$ 
    $\text{roundup}(\frac{\text{FILE\_SIZE}}{\text{SECTOR\_SIZE}})$ ;
5    $\text{query}[] \leftarrow \text{null}$ ;
6   for  $k \leftarrow 0$  to  $(\text{NUMBER\_OF\_SECTORS} - 1)$  do
7      $\text{sector}_k \leftarrow \text{splitFileIntoKthSector}(F,$ 
8      $k)$ ;
9     if  $\text{filledWithConstant}(\text{sector}_k)$  then
10    | continue;
11  end
12  if  $\text{sizeof}(\text{sector}_k) <$ 
13   $\text{SECTOR\_SIZE\_FOR\_FILE\_SEARCH}$  then
14  |  $\text{zeroPadding}(\text{sector}_k)$ ;
15  end
16   $\text{query}[] \leftarrow \text{hash}(\text{sector}_k)$ ;
17 end
18 foreach  $\text{searchHashDBMapper}(key, value)$ 
19 do
20   for  $k \leftarrow 0$  to
21    $(\frac{\text{LogDrive.sector\_size}}{\text{SECTOR\_SIZE\_FOR\_FILE\_SEARCH}} - 1)$  do
22   for  $l \leftarrow 0$  to  $\text{NUMBER\_OF\_SECTORS}$  do
23   | if  $\text{KthHash}(key) == \text{query}[l]$  then
24   | |  $\text{writeSearchResultToHDFS}$ 
25   | |  $(value, \text{null})$ ;
26   | end
27   end
28 end

```

Figure 6 shows the implementation of the proactive data collection function. We implemented Algorithms 1 and 2 in the LogDrive collection driver of the Xen hypervisor using the `blkmap2` mechanism [49–51]. The gray region in Fig. 6 is protected against customers C and attackers A . The isolation capability between a customer C 's virtual machine and a provider P 's host operating system is provided by the Xen hypervisor [52]. This paper assumes that the customers C or attackers A cannot access the gray region as long as the underlying hypervisor software is not compromised.

We have confirmed that the current implementation of the LogDrive drivers works on the specific minor version of the kernel and Xen hypervisor shown in Table 2 because the `blkmap2` software is tightly integrated with the specific kernel driver for the Xen hypervisor.

Algorithm 5: ReadPastSectorFromLog

input : An LBA i , a UNIX time in seconds t_s , a UNIX time in nanoseconds t_{ns} , a $LogDrive_c$ database

output: A 4 KiB sector s_{ij} , where j satisfies $Log_{i,j}.(t_s.t_{ns}) \leq t_s.t_{ns} < Log_{i,j+1}.(t_s.t_{ns})$

```

1 begin
2   foreach read request on  $VBD_{c,t}$  do
3      $Log_{i,m-1} \leftarrow$ 
       retrieveLog ( $LogDrive_c.index[i]$ );
4     while  $Log_{i,j} \neq null$  do
5       if  $Log_{i,j}.(t_s.t_{ns}) \leq (t_s, t_{ns})$  then
6         return  $Log_{i,j}.s$ ;
7       end
8        $Log_{i,j} \leftarrow$  retrieveLog ( $Log_{i,j}.byteOffset$ 
       OfPreviousSector);
9     end
10    return
      SECTOR_PADDED_WITH_ALL_ZEROS;
11  end
12 end

```

Figure 7 shows how Algorithms 3 and 4 are implemented using the Hadoop cluster. The first half of Algorithm 3 converts a $LogDrive$ database on a local drive into a SequenceFile on a Hadoop cluster. The last half of Algorithm 3 creates a $HashDB$. Algorithm 4 is executed by using the combination of the sequential search algorithm and random sampling. Both algorithms were implemented in the MapReduce programs.

Figure 8 shows the implementation of Algorithm 5. We implemented Algorithm 5 in the $LogDrive$ restoration driver by using the blktp2 mechanism of the Xen hypervisor.

Evaluation

This section presents the performance evaluation. We discuss how $LogDrive$ can be adapted to larger cloud environments in the Discussion section.

Table 1 Hardware specification

	Master	Slave
CPU	Xeon E5-2630v3 x2	Core i7 5820K
(No. of cores)	(16)	or 7800X (6)
RAM	DDR4 64 GiB	DDR4 64 GiB
NIC	10GBASE-T	10GBASE-T
Storage	RAID0 consists of three SATA3 SSDs (Crucial CT512)	M.2 SSD (Samsung 960EVO)
Usage	Algorithms 1, 2, and 5	Algorithms 3 and 4
No. of servers	1	7

Table 2 Software specification

	Master	Slave
Kernel	2.6.32.57	2.6.32-573.el6
OS	CentOS 6.8 64bit	CentOS 6.8 64bit
Hypervisor	Xen-4.1.2	N/A
Hadoop	Version 2.7.1	Version 2.7.1
Compiler	gcc 4.4.7	OpenJDK 1.7.0
Usage	Algorithms 1, 2, and 5	Algorithms 3 and 4
No. of servers	1	7

Proactive data collection function

Figure 9 shows the throughput of the block write and block read of the $LogDrive$ collection driver and of the normal Xen hypervisor. The experiments were conducted using the Bonnie++ benchmark software on up to 16 virtual machines. The master server shown in Table 1 was used for the experiments. Table 4 shows the configuration of the virtual machines.

The write throughput of the proposed $LogDrive$ collection driver was higher than that of the normal Xen hypervisor without $LogDrive$ when two or more virtual machines were executed simultaneously. For example, the write throughput of $LogDrive$ was 6.9 times faster than that of the normal Xen hypervisor when 16 virtual machines were executed simultaneously. The average write throughput of $LogDrive$ was 4.1 times faster than that of the normal Xen hypervisor without $LogDrive$.

The read throughput of the proposed $LogDrive$ collection driver was lower than that of the normal Xen hypervisor in all cases. The decreasing rate of the worst read throughput of four virtual machines was 54%. The average decreasing rate of the read throughput of $LogDrive$ was 27%. We confirmed that the log-structured design mainly offers users the advantage of write throughput.

File search function of the $LogDrive$ database

We created a $LogDrive$ database using the following procedures. First, a virtual machine with the configuration shown in Table 4 was booted. CentOS 5.11 Linux as the

Table 3 Configuration of the Hadoop MapReduce cluster

Parameter	Value
mapreduce.map.memory.mb	25000
mapreduce.map.java.opts	-Xmx20000m
mapreduce.map.cpu.vcores	2
mapreduce.reduce.memory.mb	25000
mapreduce.reduce.java.opts	-Xmx20000m
mapreduce.reduce.cpu.vcores	2
yarn.app.mapreduce.am.resource.mb	25000
yarn.app.mapreduce.am.command-opts	-Xmx20000m
yarn.app.mapreduce.am.resource.cpu-vcores	2

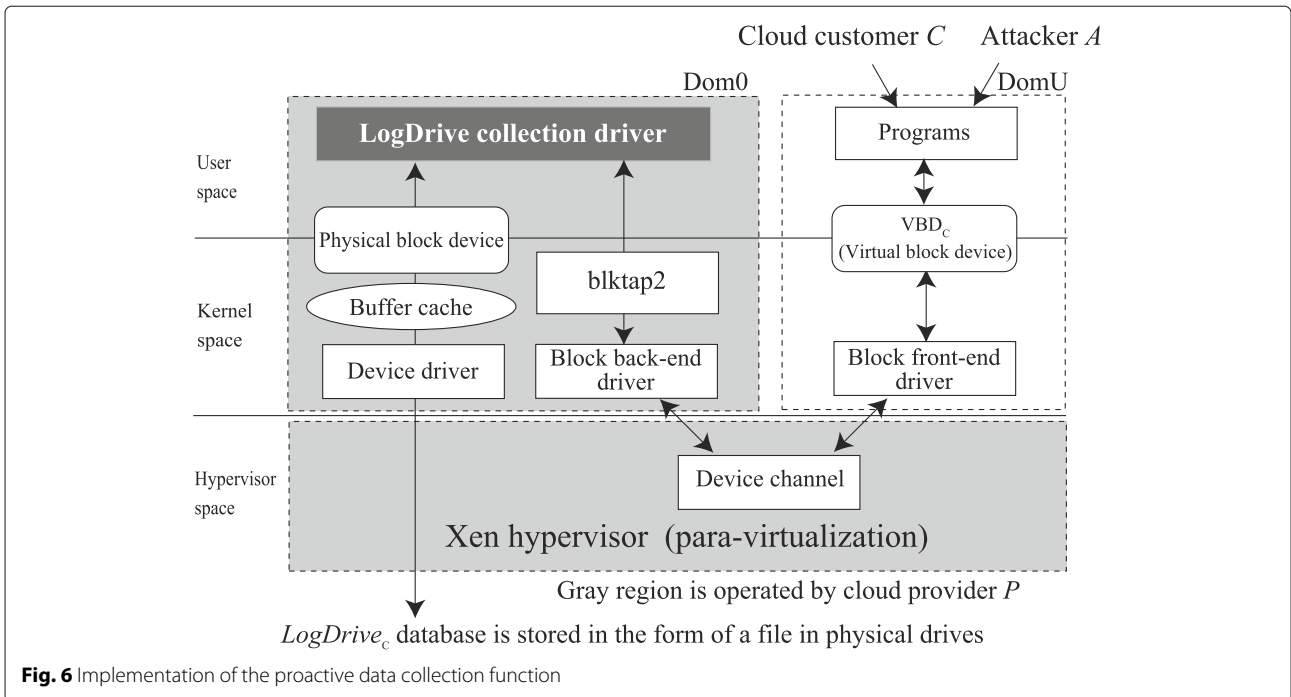


Fig. 6 Implementation of the proactive data collection function

guest OS was installed on the virtual machine. Then, files of the Govdoc dataset [44, 53] were written on the virtual machine for generating *LogDrive* databases. The Govdoc dataset consists of one million files that were collected from web servers of the .gov domain. The five *LogDrive* databases shown in Table 5 were created for the following experiments.

In the experiment, we used CentOS 6.8 with ext4 file system for the host operating system and CentOS 5.11 with ext3 file system for guest operating system. In our setup, we could not boot guest operating systems with ext4 file system because of the limited support of Xen 4.1.2 we used. However, the design of *LogDrive* supports any file system (e.g., an operating system with ext4 file

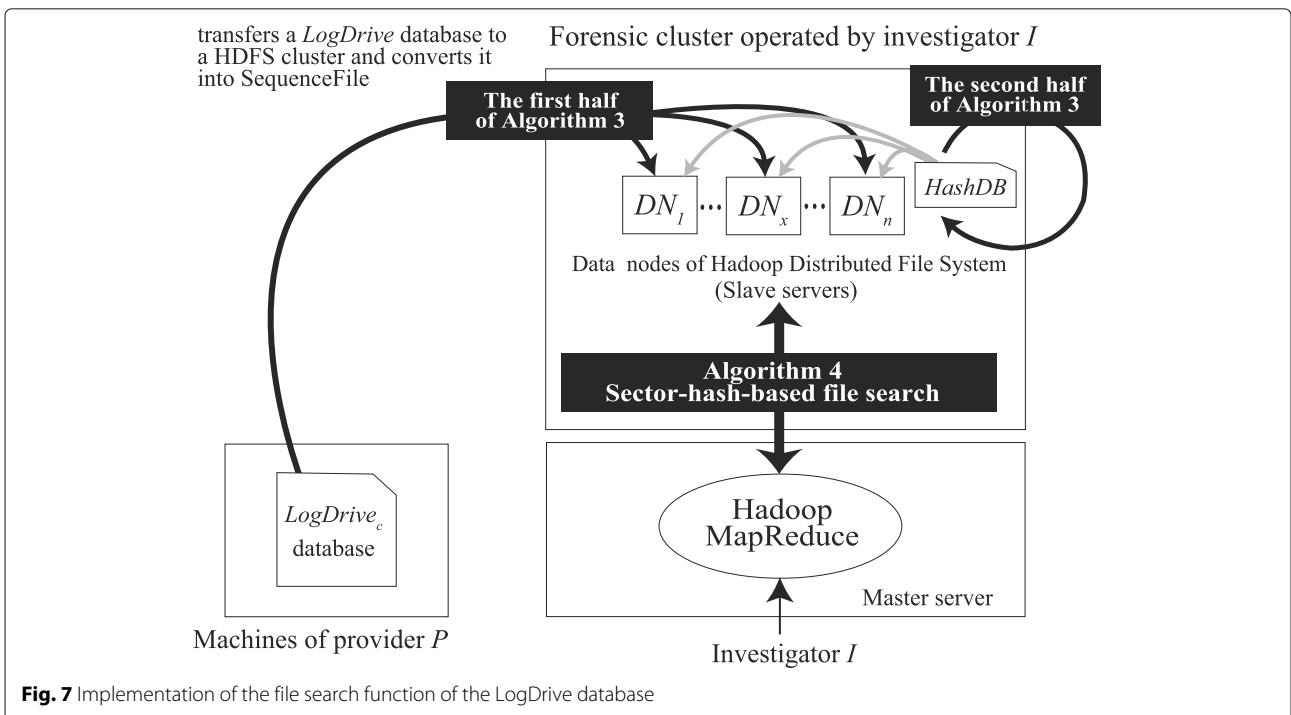


Fig. 7 Implementation of the file search function of the LogDrive database

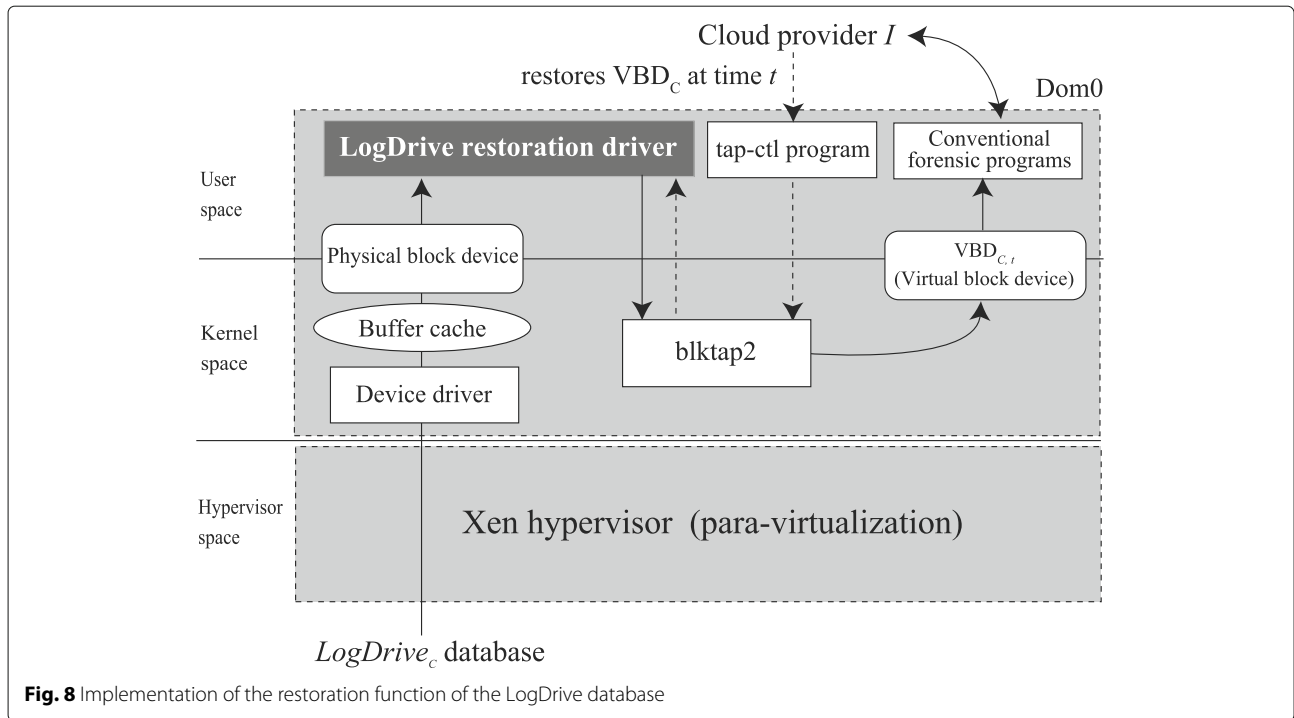


Fig. 8 Implementation of the restoration function of the LogDrive database

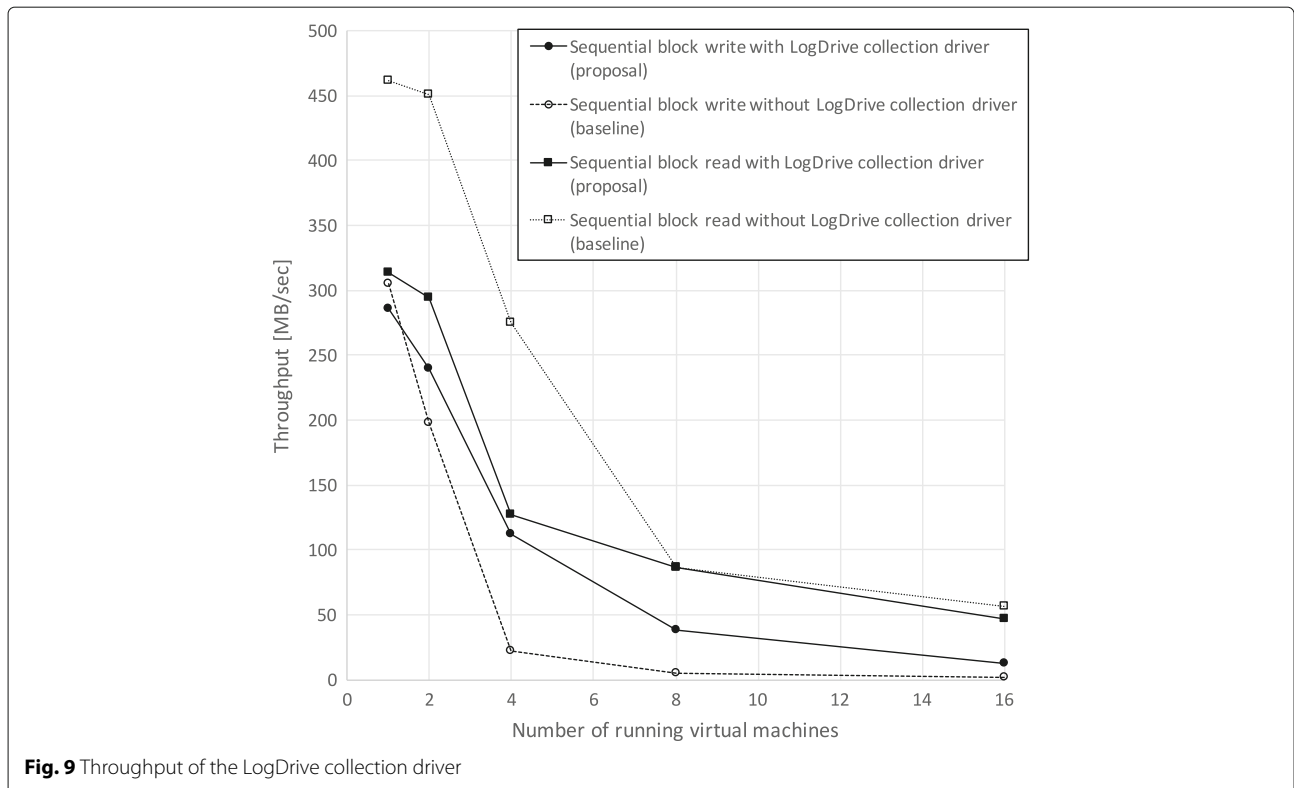


Fig. 9 Throughput of the LogDrive collection driver

Table 4 Configuration of the virtual machines

Virtual CPU	1 core
Virtual RAM	2 GiB
Virtual block device	<i>LogDrive.disk_size</i> = 1 TiB <i>LogDrive.sector_size</i> = 4096
Guest OS	CentOS 5.11 32bit
File system	ext3
Benchmark software	Bonnie++ 1.9.7

system) because LogDrive does not need to process any specific information of underlying file system. As shown in Algorithms 1 and 2, LogDrive processes a 4 KiB sector, a logical block address (LBA), and a UNIX timestamp as an input. A UNIX timestamp is obtained from a host operating system and the timestamp is independent of file systems of guest operating systems. A 4 KiB sector and an LBA are also independent of file systems. These data do not include any file system related information such as file system's metadata. In addition, Algorithm 5 processes a logical block address and a UNIX timestamp as an input. These data do not include any file system related information. Therefore, the design of LogDrive is independent of underlying file systems in guest operating systems.

Figure 10 shows the time for converting the *LogDrive* database into SequenceFile and for creating *HashDB*. The Hadoop cluster that consists of the seven slave servers shown in Table 1 was used. The average throughput of the first half of Algorithm 3 was 28.4 MiB/s and of the second half of Algorithm 3 was 239 MiB/s. As the size of the *LogDrive* database increased, the processing time increased linearly. Therefore, the time complexity of Algorithm 3 is $\mathcal{O}(n)$, where n is the number of written sectors in a *LogDrive* database.

Figure 11 shows the response time of Algorithm 4 without random sampling. The experiments of a *LogDrive*

Table 5 Size of the *LogDrive* databases for the experiments

Size of <i>LogDrive</i> database [GiB]	No. of written directories from (Each directory holds 1,000 files) Govdoc
7.7	0
178.3	250
309.3	500
452.1	750
542.4	1,000

The first row shows the size of the *LogDrive* database after a clean installation of the guest OS

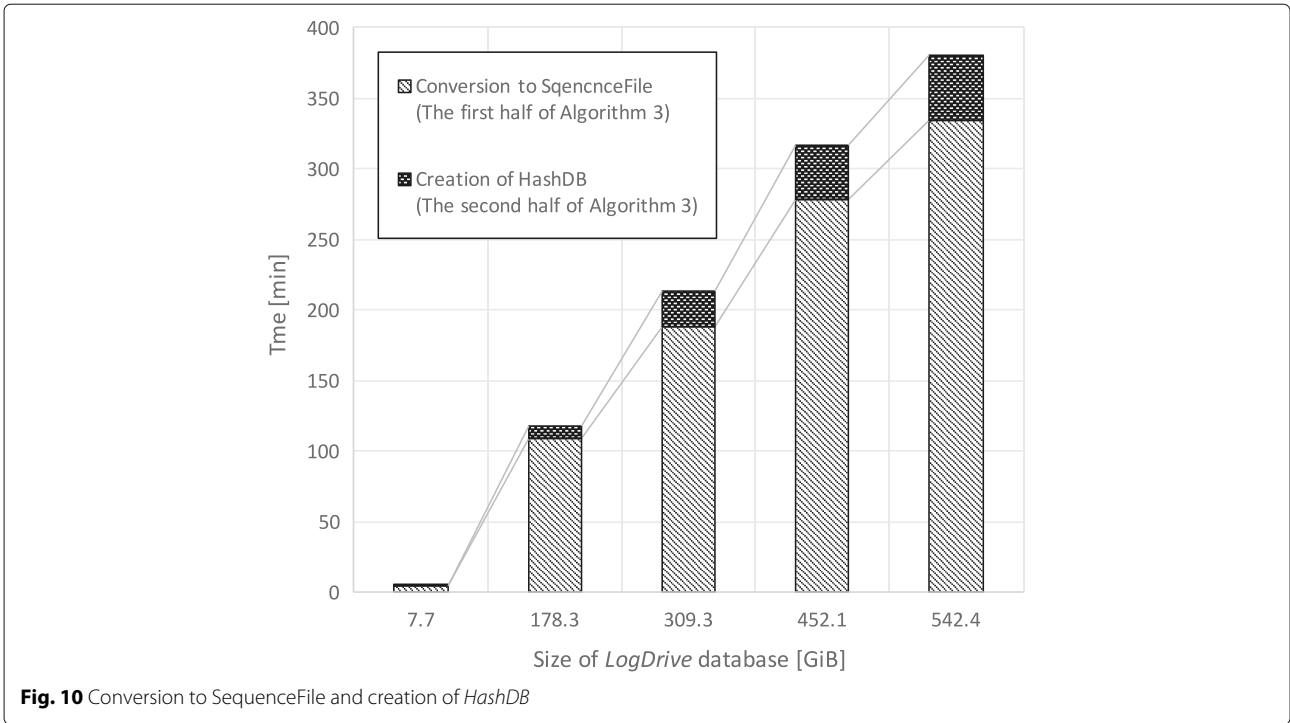
database larger than 542.4 GiB were conducted using multiple 542.4 GiB *LogDrive* databases. As the number of sectors in a *LogDrive* database increased, the response time increased linearly. Table 6 shows the average throughput of the search queries (i.e., the slopes of the lines shown in Fig. 11). While our setup searched for a file smaller than 100 KiB at about 2.86 GiB/s, the query throughput was dropped to 1.69 GiB/s when the system searched for the 1 MiB file. Evaluating the average throughput of an investigator's setup is useful when the investigator has to estimate the processing time and to determine the appropriate specification of the forensic cluster (e.g., the number of computing nodes). For example, our setup will be able to search for a 100 KiB file in a 100 TiB *LogDrive* database without random sampling in about 10.3 h.

Figure 12 shows the response time of Algorithm 4 with random sampling in logarithmic scale. The experiments of a *LogDrive* database larger than 542.4 GiB were conducted by using multiple 542.4 GiB *LogDrive* databases. The sampling rate $r = n/N$ was calculated using Eq. 1, where $p > 0.99$, N is the number of sectors in a *LogDrive* database, and T is the number of sectors in a file F . Table 7 shows the sampling rate r for each file size. The sampled *HashDB* files were created using Algorithm 3 with these sampling rates. For example, the search for the 10 MiB file (017804.pdf) in a 100 TiB *LogDrive* database required 2.9 min. Table 8 shows the average throughput of Algorithm 4 with random sampling. While our setup searched for a file smaller than 10 MiB at about 398 GiB/s, the query throughput was dropped to 103 GiB/s when the system searched for the 100 MiB file.

Restoration function of the *LogDrive* database

First, a virtual block device *VBD* with 12 GiB capacity formatted with ext4 file system was created. Then, a *LogDrive* database was created by overwriting an 8 GiB test file 16 times. The test file was generated by executing the command `dd if=/dev/urandom of=test.dat count=8 bs=`echo 2^30|bc`. After creating the LogDrive database, the read throughput of the LogDrive restoration driver was measured.`

Figure 13 shows the throughput of the implementation of Algorithm 5. In the experiment, we restored *VBD* at 16 points in time at which the test file was overwritten. The throughput of the sequential block read was measured by the command `hdparam -t` on *VBD*. The throughput of the file system read was measured by mounting *VBD* as an ext4 file system and by executing the command `dd if=/mnt/test.dat of=/dev/null bs=1M count=10` after dropping the kernel's buffer cache



both in the guest OS and in the host OS. The page cache read was measured by the same method instead of dropping the page cache in the host OS. As the number of overwrites increased, the read throughput decreased except for the page cache read. The

file system read was slower than the sequential block read. Once the file was loaded into the buffer cache of the host OS, the proposed system could access the restored file system at a throughput of 2 GB/s.

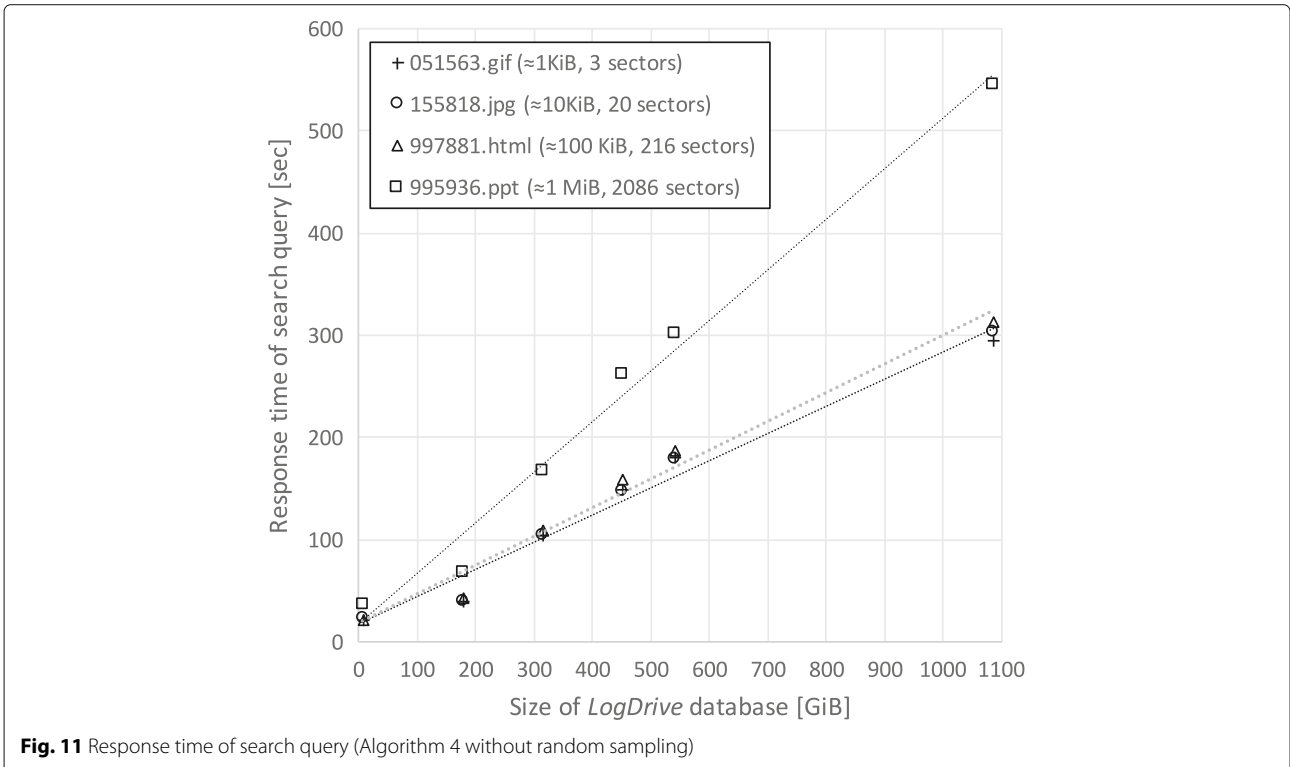


Table 6 Average query throughput of Algorithm 4 without random sampling

Test file of Govdoc dataset [44]	Throughput [MiB/s]
051563.gif (≈1 KiB, 3 sectors)	2,988
155818.jpg (≈10 KiB, 20 sectors)	2,988
997881.html (≈100 KiB, 216 sectors)	2,811
995936.ppt (≈1 MiB, 2,086 sectors)	1,738

Average throughput = size of LogDrive / response time

Discussion

This section presents a discussion about security, performance, and size of the logs.

Security

We classify possible attacks on the LogDrive framework into six categories based on the STRIDE threats model [4]. The STRIDE threats consist of spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privilege. Table 9 shows the summary of security threats against the LogDrive framework.

Spoofing. The system cannot detect identity spoofing in virtual machines. If someone pretends to be a genuine customer *C* by using remote control malware and commits crimes in virtual machines, the LogDrive framework cannot identify the person who commits crimes over the Internet. The customer *C* is responsible for preventing identity spoofing by employing strict authentication

mechanisms and regular software updates on the virtual machines.

Tampering. The current standard Linux kernel periodically flushes the page cache if it is older than 30 s. If attackers can set the interval of flushing the page cache to be longer, the LogDrive collection driver cannot record precise timestamps of each sector. However, a longer interval affects normal operations of applications on the virtual machines; therefore, the attack has only a limited effect on timestamps. This limitation can be mitigated by using memory forensics in addition to storage forensics. Please note that the timestamps of the guest operating system have no effect on LogDrive because the timestamps of LogDrive are managed by the host operating system.

Tampering with *LogDrive* databases outside the virtual machine is discussed in the “Related work” section.

Repudiation. Even if an attacker *A* creates a modified file *F'* by inserting an extra 1 byte near the start of an original file *F* to bypass sector hashing, LogDrive preserves both the original file *F* and the modified file *F'*. In this case, similarity digest hash algorithms, such as sdhash [54], are needed to find *F'* using the original file *F*.

The current implementation employs MD5 to generate hash values of sectors. MD5 and SHA-1 are vulnerable against collision attacks [55, 56]. If the attackers can make a false file or a malicious program that has hash values of sectors identical to those of known-good files (e.g., known-good system files) and if the LogDrive framework is used for excluding fragments of known-good files by

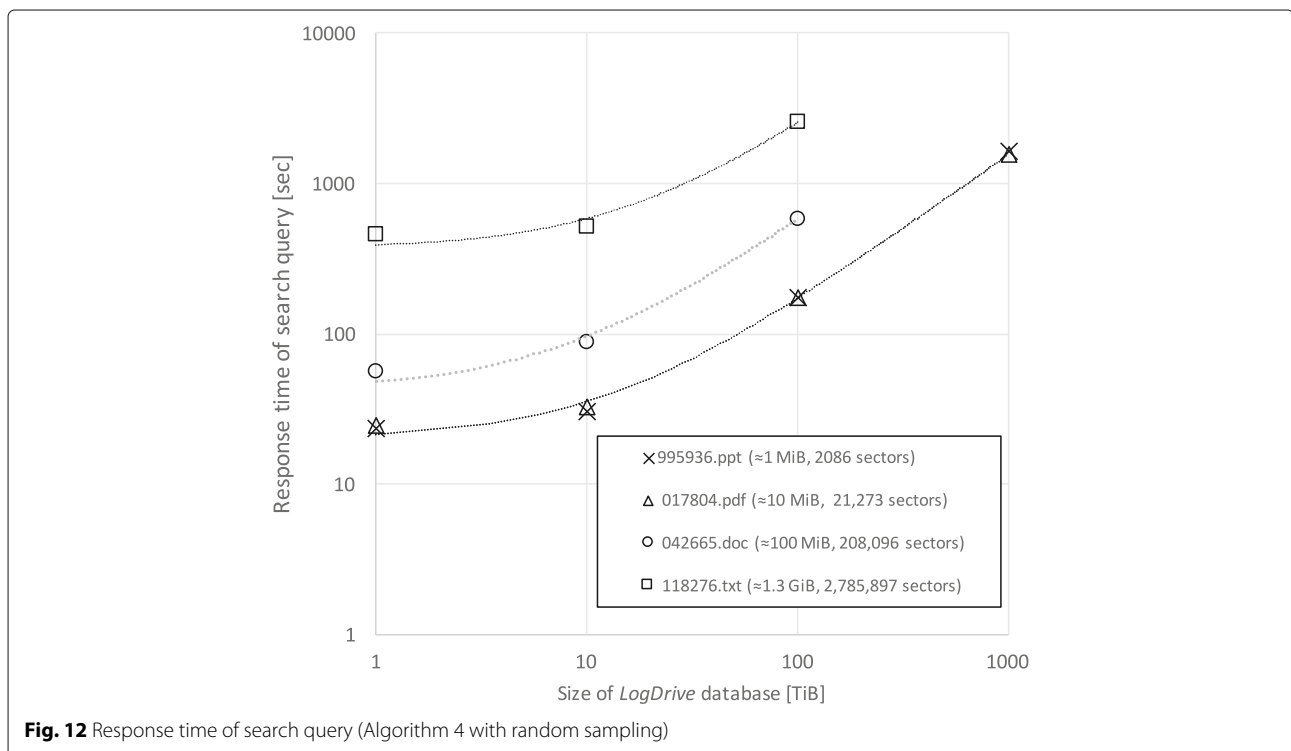


Fig. 12 Response time of search query (Algorithm 4 with random sampling)

Table 7 Sampling rate for each file size

Target file size	Sampling rate for $p > 99\%$
1 MiB	0.00229492
10 MiB	0.000229490
100 MiB	0.000022950
1 GiB	0.0000022400

using white lists (not black lists), the system will not be able to detect the malicious file created by the attacker. To prevent the attack, the MD5 hash algorithm should be replaced with collision-resistant hash algorithms such as SHA-256 or SHA-512.

An attacker can make files unreadable in a virtual machine by using cryptography or steganography to repudiate his or her past incidents. LogDrive cannot recover an encrypted file or a stego file. Therefore, other tools such as data recovery tools or memory forensic tools are needed [57] for recovering the original message.

Information disclosure. If the attackers gains accesses to the cloud management plane on the host operating system, they can steal the *LogDrive* databases of all the customers C of the provider P . LogDrive employs the combination of the Xen hypervisor [52], a host operating system, and the LogDrive framework as a trusted computing base (TCB). LogDrive assumes the integrity of the underlying trusted computing base. Hardware-based attestation mechanisms have the ability to check the integrity of the trusted computing base both in bootstrap step and in runtime [58]. A provider P is responsible for protecting the TCB against attackers.

Denial of service. If attackers issue a large number of write requests on virtual machines (e.g., DDoS attacks on a web server in a virtual machine to overflow the web server’s logs), the LogDrive framework may not be able to save all written sectors on a *LogDrive* database because of the limit of storage capacity. If LogDrive detects denial of service (DoS) attacks, it can intentionally slow down the throughput of the virtual block device *VBD*. The system can also return I/O errors to mitigate the attacks. In addition, the system can mitigate the attacks by transferring old logs of a *LogDrive* database to a Hadoop Distributed File System (HDFS) or other cloud storage periodically to

Table 8 Average query throughput of Algorithm 4 with random sampling ($p > 0.99$)

Test file of Govdoc dataset [44]	Throughput [GiB/s]
995936.ppt (≈ 1 MiB, 2,086 sectors)	396
017804.pdf (≈ 10 MiB, 21,273 sectors)	400
042665.doc (≈ 100 MiB, 208,096 sectors)	103
118276.txt (≈ 1.3 GiB, 2,785,897 sectors)	26

Average throughput = size of LogDrive / response time

Table 9 The STRIDE threats of LogDrive

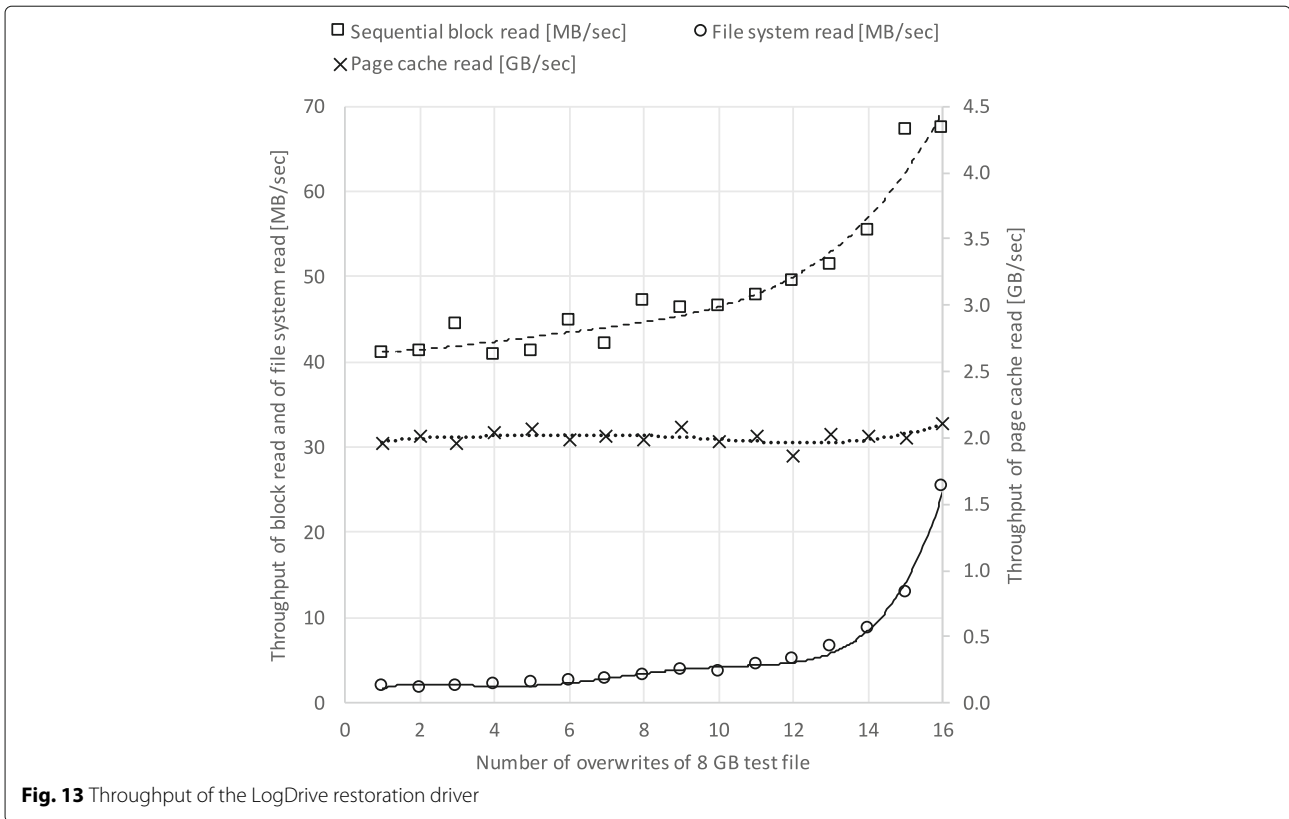
Threat	Attack method	Protection	Anti-forensic attacks	
			type 1	type 2
Spoofing	Pretending someone in VMs	Authentication on guest OS	✓	
Tampering	Incorrect timestamps	Memory forensics	✓	
	Tampering with LogDrive database	Secure audit log		✓
Repudiation	Changing a few bits of a file	Similarity digest hash	✓	
	Collision attacks	Collision resistant hash algorithm	✓	
	Cryptography and steganography	Data recovery tools, memory forensics	✓	
Information disclosure	Stealing LogDrive database	Protection of TCB		✓
Denial of service	Overflowing logs	Throughput control of LogDrive	✓	
Elevation of privilege	Taking control of LogDrive	Protection of TCB		✓

secure sufficient storage space. A log rotation mechanism is needed both for long-term collection and for mitigating the DoS attacks.

Elevation of privilege. Finally, we discuss direct attacks on the LogDrive framework. If an attacker could elevate his or her privilege, he or she might be able to attack LogDrive directly on a host operating system. In addition to hardware-based attestation mechanisms [58], minimizing the code size of the trusted computing base is also important to reduce the complexity of the system and to reduce the vulnerability of LogDrive. For this reason, we have developed the LogDrive framework on a lightweight security-purpose hypervisor called BitVisor [59].

Performance

Algorithms 1 and 2. As shown in Fig. 9, the average write throughput of the LogDrive collection driver was 4.1 times faster than the normal write throughput of the Xen hypervisor without LogDrive. The reason for the better throughput is the combination of an efficient log-structured design and a buffered I/O. The LogDrive collection driver groups multiple $Log_{i,j}$ into a single large I/O and flushes it at the tail of the *LogDrive* database. This design makes it possible to improve the write performance compared to a normal Xen hypervisor that writes each



sector s_i onto scattered positions of each LBA i on the host operating system. On the other hand, the average decreasing rate of the read throughput was 27%. The reason for the decrease in the read throughput is the overhead of the log-structured design. We confirmed that the LogDrive collection driver offers users improved write throughput (i.e., four times faster than the normal throughput without LogDrive) and moderate read throughput.

We believe that LogDrive is the first work to log all write operations on a virtual block device for forensic purpose. Therefore, we cannot compare with other prior work of same purpose. Here, we compare LogDrive with SNPdisk (snapshot disk), a copy-on-write snapshot mechanism. Table 10 shows the performance comparison with SNPdisk [60]. Although SNPdisk degrades the both read throughput and write throughput, LogDrive improves write throughput because of our log-structured design. King et al. presented time-traveling virtual machine

for reverse debugging of operating systems [61]. King et al. implemented their time-traveling virtual machine (TTVM) by using ReVirt [62] and User-Mode Linux. TTVM records the contents of a virtual machine’s CPU registers, virtual memory pages, and virtual disks (i.e., virtual block devices). While TTVM employs a copy-on-write method to save changes in memory pages and in disk blocks, LogDrive employs log-structured design to save changes in a virtual block device. Although TTVM takes checkpoints at a certain interval such as 10 s, LogDrive takes continuous checkpoints without any interval. The performance overhead of TTVM that took checkpoints every 10 s was 16–33%. In contrast, the average decreasing rate of read throughput in LogDrive is 27% and the average increasing rate of write throughput in LogDrive is 410%. LogDrive improved write throughput because of its log-structured design.

Algorithm 3. Algorithm 3 consists of two parts. The first part is to transfer a *LogDrive* database to a Hadoop cluster and to convert it into a SequenceFile. The second half is to create *HashDB* from the SequenceFile. Although the setup employs 10GBASE-T Ethernet, the average throughput of the first half of Algorithm 3 was 28.4 MiB/s and of the second half of Algorithm 3 was 239 MiB/s as shown in Fig. 10, respectively. For example, our setup needs to transfer write logs to a Hadoop cluster at 305 MiB/s when the single virtual machine is

Table 10 Performance comparison with copy-on-write snapshot

	LogDrive (Log-structured)	SNPdisk (Copy-on-write)
Read throughput	73%	86%
Write throughput	410%	86%

The above values are the percentage of read throughput and write throughput compared with non-snapshot. The performance of SNPdisk is based on scheme 1 of the experiment in [60]

executed. The throughput of the first half of Algorithm 3 could be improved by introducing parallel processing of each part of *LogDrive* database. The throughput of the second half of Algorithm 3 could be improved by increasing the number of slave machines. For mitigating the low throughput of Algorithm 3, *LogDrive* has to perform Algorithm 3 periodically in the background. This paper presented the evaluation of the proof-of-concept of the *LogDrive* framework. Further studies of more efficient transfer mechanism of write logs to a cluster is required.

Algorithm 4. If *LogDrive* monitors k virtual machines that writes data at an average throughput of T_{VM} byte/s during D days, *LogDrive* needs to process *LogDrive* databases of S_{total} bytes. The database size S_{total} is calculated using Eq. 2.

$$S_{total} = k * T_{VM} * D * (60 * 60 * 24) \text{ [bytes]} \quad (2)$$

For example, if we have to monitor 10 virtual machines of the average throughput $T_{VM} = 1,048,576$ (i.e., 1,024 KiB/s) in 100 days, then the total size of the *LogDrive* databases S_{total} is 82.4 TiB.

The response time of the search query t_{search} can be estimated using Eq. 3, where $T_{experiment}$ is the average throughput obtained from the experiments shown in Table 6 and in Table 8, respectively.

$$t_{search} = \frac{S_{total}}{T_{experiment}} \text{ [sec]} \quad (3)$$

The response time of the search for a 10 MiB file with random sampling ($p > 99\%$) in a 82.4 TiB *LogDrive* database is 210 sec (i.e., 3.5 min) when the parameters are $T_{experiment} = 400 * 2^{30}$ and $S_{total} = 82.4 * 2^{40}$.

For faster search, the number of slave machines have to be increased. Our setup consists of seven slave machines. If we assume that the number of slave machines is proportioned to the query throughput because Algorithm 4 is essentially sequential search algorithm, the following Eq. 4 is derived. T_{single} is the throughput of single slave machine and N_{slave} is the number of the slave machines that derives $T_{experiment}$.

$$T_{single} = \frac{T_{experiment}}{N_{slave}} \text{ [byte/s]} \quad (4)$$

The throughput of the single slave machine that searches for a 10 MiB file with random sampling ($p > 99\%$) is 57 GiB/s. If the total number of slave machine is 100 then the throughput of the cluster is 5.5 TiB/s. In this setup, *LogDrive* can search *LogDrive* databases of $S_{total} = 82.4 * 2^{40}$ bytes (82.4 TiB) for a 10 MiB file in 14.7 sec.

From the perspective of the technique we used, one of the most similar work with ours is statistical random sampling in combination with sector hashing to find a physical drive holding such as child pornography or malware. Taguchi [63] shows that with 15 min of sampling

they can give a 90% confidence that less than 10 MiB of target data is present on a 500 GB hard disk drive (i.e., 556 MB/s). We cannot directly compare *LogDrive* with Taguchi's work because *LogDrive* needs to create HashDB before searching, for example, we need 380 min to make HashDB from about 500 GiB write operations. However, in our setup, once the HashDB is created, we can search any target file of 10 MiB with 99% confidence in 400 GiB/s as many times as needed.

Algorithm 5. As shown in Fig. 13, as the number of writes of the restored virtual block device increased, both the read throughput of sequential block read and file system read increased. The reason of the increase of the read throughput is the number of traversal in the linked list in the *LogDrive* database. The latest log can be accessed at the first traversal of the linked list. Conversely, the oldest log can be accessed at the last traversal of the linked list.

The sequential block read throughput of a single virtual machine without *LogDrive* was around 300 MB/s. In contrast, the lowest sequential block read throughput on the restored virtual block device was around 40 MB/s; therefore, our restoration driver's throughput is about 13% of the read throughput without *LogDrive*.

Time-traveling virtual machine (TTVM) [61] is similar work with *LogDrive* except for that TTVM's purpose is reverse debugging. While TTVM employs a copy-on-write virtual block device to create checkpoints, *LogDrive* employs log-structured virtual block device. In the paper [61], restoration time of a checkpoint in TTVM were between 5 s and 23 s when TTVM took checkpoints every 25 s. In contrast, *LogDrive* does not need to restore a checkpoint prior to analysis because *LogDrive* can dynamically restore the virtual block device at a specified point in time when an investigator requested. The average read throughput of the dynamically restored *LogDrive* is about 13% of the original read throughput without *LogDrive*. Our setup will need 7 h to make a disk image using `dd` command from the restored 1 TiB virtual block device. However, once the disk image was created, the investigator can use conventional forensic tools for analyze the disk image. This paper presented the evaluation of the proof-of-concept of *LogDrive*. Further studies of more efficient restoration mechanisms of a log-structured database are needed.

Size of logs

The *LogDrive* framework collects all written sectors; therefore, it needs a large amount of storage for *LogDrive* databases. If users do not need to restore the past state of virtual block devices, an accumulator such as a bloom filter [64] can be used for holding only proof of previous possession of sectors. In this case, the system stores hash values of written sectors in bloom filters. An investigator I will be able to search an evidence file $F = \{s_0, s_1, \dots,$

s_{n-1} }, where n is the number of sectors of the file F , from the *LogDrive* database that is a set of bloom filters. A log-rotation mechanism, such as the cleaning functions in the design of an original log-structured file system [33], is also needed to recycle the space of old logs.

Related work

This section compares the LogDrive framework with similar systems. Table 11 shows the comparison of different collection methods of virtual block devices in IaaS cloud environments.

Dykstra and Sherman evaluated forensic acquisition tools, including Guidance EnCase and AccessData Forensic Toolkit, in a public IaaS cloud [65]. They successfully returned volatile and non-volatile data from Amazon EC2 by injecting remote data acquisition agent programs (EnCase servlet and FTK agent) into the guest operating systems. These agent programs run on the guest operating systems. In this method, a cloud customer can bypass the collection mechanism or can tamper with collected data because he or she has full control of the guest operating system. Thus, remote data acquisition using agent programs cannot prevent **type 1** anti-forensic attacks.

Dykstra and Sherman presented the design and implementation of FROST [66]. FROST acquires virtual disks from OpenStack cloud platforms. The aim of FROST is to develop a cloud management plane for forensic data collection rather than interacting with guest operating systems in virtual machines. FROST retrieves virtual disks as QCOW2 (QEMU copy-on-write, version 2) [40] images. QCOW2 images support snapshot functions. Investigators can inspect the history of changes between two snapshots. However, the frequency of taking snapshots of copy-on-write disk images is dependent on cloud customers. Even if cloud providers have full control of taking snapshots of customers' virtual machines, they cannot preserve the traces of **type 1** anti-forensic attacks between

two snapshots. For example, if a cloud provider took two snapshots at t_0 and at t_4 in Fig. 3, the evidence file F does not remain on the copy-on-write snapshots. FROST provides an authenticated logging service that guarantees integrity of the logs; however, the service cannot prevent attacks launched on a host operating system (i.e., **type 2** anti-forensic attacks) because FROST assumes that cloud providers are trusted.

Zawoad and Hasan proposed a forensic-enabled cloud (FECloud) [67] designed on top of OpenStack. The advantage of FECloud is protection of logs against collusion attacks between dishonest cloud providers, customers, and investigators. Building secure logging mechanisms on untrusted machines operated by dishonest entities is a challenging problem [31]. The typical solution is secure audit logging using a trusted third party. FECloud includes the function of Secure Logging-as-a-Service (SecLaaS) [32] that publishes proofs of past logs on public web sites or on Rich Site Summary (RSS) feeds periodically. A proof of past logs is created by using a hash-chain of previous logs and proof accumulators (i.e., bloom filters or RSA accumulators). FECloud provides a secure audit of logs only after issuing a proof of logs. If a dishonest provider tampered with the customers' logs before issuing a proof of the logs, FECloud cannot detect the attacks. From the perspective of storage forensics, they proposed a virtual file system (VFS) monitor in the kernel layer. The VFS monitor requires trust in the guest operating system; therefore, the VFS monitor cannot protect against **type 1** attacks from customers who have full control of the guest operating system. FECloud does not provide analysis functions.

The LogDrive framework records all written sectors on virtual block devices as logs; therefore, investigators can search for the evidence file F that was deleted by **type 1** anti-forensic attacks. However, the LogDrive framework does not prevent attacks launched on the host operating system (i.e., **type 2** anti-forensic attacks). To guarantee

Table 11 Comparison of different collection methods of virtual block devices in IaaS clouds

	Trust required	Method	Preserving traces of anti-forensic attacks of type 1	Protection from anti-forensic attacks of type 2	Parallel processing of past logs
Remote data acquisition [65]	OS, HV, Host, HW	Agent programs			
Management plane [66]	HV, Host, HW	CoW snapshots of VMs	Depends on frequency of snapshots		
FECloud [67] with SecLaaS [32]	OS, HV, Host, HW	VFS monitor		✓ (audit logging)	
LogDrive (proposal)	HV, Host, HW	Log-structured virtual storage	✓		✓

The field of "Trust required" shows the trust required in the guest operating system (OS), hypervisor (HV), host operating system (Host), and host hardware (HW)

the integrity of the *LogDrive* database on a host operating system, *LogDrive* needs to be integrated with a secure audit logging mechanism such as *SecLaaS* [32]. *SecLaaS* provides integrity of log files by publishing a proof of logs on public web sites or on RSS feeds periodically. *LogDrive* records all written sectors as logs. Therefore, we believe that *LogDrive* can be integrated with *SecLaaS* without changing fundamental parts of the two schemes. To enforce the *LogDrive* and secure audit of *LogDrive* databases, the integrity of the trusted computing base (i.e., underlying host operating system, hypervisor, and *LogDrive* programs) must be guaranteed by using hardware-based attestation mechanisms [58].

Finally, we compare the abilities of parallel processing in the analysis phase. In the remote data acquisition method [65] and the management plane method [66], conventional forensic tools are used to analyze disk images. Conventional forensic tools can be processed by grouping files on each machine of a cluster [20, 68], as Richard and Roussev presented. *LogDrive* has an advantage of searching files that were overwritten or deleted in the past via parallel distributed processing.

Conclusion

This paper presented the *LogDrive* framework to mitigate the following problems of storage forensics in IaaS cloud environments: (1) volatility of virtual block storage and difficulty in imaging the physical drives in cloud data centers, (2) increasing volume of forensic data, and (3) anti-forensic attacks that hide traces of attacks in virtual machines. The *LogDrive* framework provides a proactive data collection and analysis function for mitigating these three problems. We employed a log-structured design to archive all written sectors to mitigate problems (1) and (3) and the sector-hash-based file detection method with random sampling to mitigate problem (2). This paper presented the problem formulation, the investigative context, the design with five new algorithms, the implementation process, and the performance evaluation.

This paper presented a proactive data collection mechanism to preserve writes history in virtual machines. This paper showed that the average write throughput of the *LogDrive* proactive collection driver was 3.1 times faster than the normal write throughput of the Xen hypervisor without *LogDrive*. Additionally, we integrated a sector-hash-based file detection method with random sampling with *LogDrive*. For example, the average throughput of the search for a 10 MiB file with random sampling was 400 GiB/s and our setup searched a 100 TiB *LogDrive* database for the 10 MiB file (017804.pdf) in 2.9 min.

The design and evaluation of the proof-of-concept of the *LogDrive* framework will help investigators consider more advanced proactive data collection and analysis

mechanisms for time-traveling investigations in volatile IaaS cloud environments. The software of this article is available in the GitHub repository, <https://github.com/manabu-hirano/logdrive>. In future work, we are going to use the logs obtained from the *LogDrive* system in machine learning for anomaly detection and future prediction. The proposed hypervisor-based *LogDrive* system can be used as a common platform to collect and to analyze forensic evidences in cloud environments.

Abbreviations

BLKTAP2: Block tap userspace toolkit, version 2; CPU: Central processing unit; CoW: Copy-on-write; DDoS: Distributed denial of service; DFRWS: Digital forensic research workshop; DN: Data node; DoS: Denial of service; EC2: Elastic compute cloud; EXT3: Third extended file system; EXT4: Fourth extended file system; FECloud: Forensic-enabled cloud; FROST: Forensic OpenStack tools; FTK: Forensic tool kit; FUSE: Filesystem in userspace; HDD: Hard disk drive; HDFS: Hadoop distributed file system; I/O: Input output; IaaS: Infrastructure-as-a-Service; LBA: Logical block address; MD5: Message digest 5; MPI: Message passing interface; NFS: Network file system; NIST: National institute of standards and technology; NSRL: National software reference library; NVM: Non volatile memory; OS: Operating system; QCOW2: Quick emulator copy-on-write, version 2; QEMU: Quick emulator; RAM: Random access memory; RDS: Reference data set; RSA: Rivest, Shamir, and Adleman; RSS: Rich site summary; S4: Self-securing storage server; SHA: Secure hash algorithm; SLA: Service level agreement; SSD: Solid state drive; STRIDE: Spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privilege; *SecLaaS*: Secure logging-as-a-service; TCB: Trusted computing base; VBD: Virtual block device; VFS: Virtual file system; VHD: Virtual hard disk; VM: Virtual machine; ZIP: Zip data compression and archival file format

Acknowledgments

We thank Dr. Suguru Yamaguchi for his longstanding support to our research project. We thank H. Ogawa, H. Takase, and K. Yoshida for their effort in the initial development.

Funding

This work was supported by JSPS KAKENHI Grant Number JP26330168 and JP17K00198.

Availability of data and materials

The software supporting the conclusions of this article is available in the GitHub repository, <https://github.com/manabu-hirano/logdrive>.

Authors' contributions

MH designed and developed the proposed framework as well as analyzed the data from the experiments. NT and SI conducted the experiments of the sector-hash-based file detection function. RK suggested how to analyze the output of the proposed system. All authors approved the final manuscript.

Authors' information

Manabu Hirano is an associate professor in Department of Information and Computer Engineering, National Institute of Technology, Toyota College. He received his Ph.D. degree in Engineering from Nara Institute of Science and Technology (NAIST) in 2008. He studied at University of Kent, UK, as a visiting senior lecturer between June 2012 to January 2013. Natsuki Tsuzuki and Seishiro Ikeda are students of Dr. Hirano's laboratory. Ryotaro Kobayashi is currently an associate professor at Kogakuin University. He received his B.E., M.E., and D.E. degrees from Nagoya University in 1995, 1997, and 2001, respectively. He had been a research assistant at Nagoya University from 2000 to 2008, a lecturer at Toyohashi University from 2008 to 2015, and an associate professor at Toyohashi University in 2016.

Competing interests

The authors declare that they have no competing interests.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Author details

¹National Institute of Technology, Toyota College, 2-1 Eisei, 471-8525 Toyota, Japan. ²Kogakuin University, 1-24-2 Nishi-shinjuku, Shinjuku-ku, 163-8677 Tokyo, Japan.

Received: 6 April 2018 Accepted: 13 September 2018

Published online: 03 October 2018

References

- Mell P, Grance T (2011) The NIST definition of cloud computing, SP 800-145. National Institute of Standards and Technology, Gaithersburg. <https://csrc.nist.gov/publications/detail/sp/800-145/final>. Accessed 18 Sept 2018
- Hirano M, Ogawa H (2016) A log-structured block preservation and restoration system for proactive forensic data collection in the cloud. In: Availability, Reliability and Security (ARES), 2016 11th International Conference On. IEEE. pp 355–364. <https://doi.org/10.1109/ARES.2016.8>
- Hirano M, Takase H, Yoshida K (2015) Evaluation of a sector-hash based rapid file detection method for monitoring infrastructure-as-a-service cloud platforms. In: Availability, Reliability and Security (ARES), 2015 10th International Conference On. IEEE. pp 584–591. <https://doi.org/10.1109/ARES.2015.15>
- Shostack A (2014) Threat Modeling: Designing for Security. Wiley, Hoboken
- Palmer G (2001) A road map for digital forensic research. In: First Digital Forensic Research Workshop, Utica. pp 27–30. http://dfrrws.org/sites/default/files/session-files/a_road_map_for_digital_forensic_research.pdf. Accessed 18 Sept 2018
- Kent K, Chevalier S, Grance T, Dang H (2006) SP800-86. Guide to integrating forensic techniques into incident response. National Institute of Standards and Technology, Gaithersburg
- Elyas M, Ahmad A, Maynard SB, Lonie A (2015) Digital forensic readiness: Expert perspectives on a theoretical framework. *Comput Secur* 52:70–89
- Ruan K, Carthy J, Kechadi T, Crosbie M (2011) Cloud forensics. In: IFIP International Conference on Digital Forensics. Springer, Berlin Heidelberg. pp 35–46
- Martini B, Choo K-KR (2012) An integrated conceptual digital forensic framework for cloud computing. *Digit Investig* 9(2):71–80
- Ab Rahman NH, Glisson WB, Yang Y, Choo K-KR (2016) Forensic-by-design framework for cyber-physical cloud systems. *IEEE Cloud Comput* 3(1):50–59
- Garfinkel SL (2010) Digital forensics research: The next 10 years. *Digit Investig* 7:564–573
- Fox A, Griffith R, Joseph A, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I (2009) Above the Clouds: A Berkeley View of Cloud Computing. Electrical Engineering and Computer Sciences. University of California at Berkeley. p. 1–23. Technical Report No UCB/EECS-2009-28
- Reilly D, Wren C, Berry Tq (2011) Cloud computing: Pros and cons for computer forensic investigations. *Int J Multimed Image Process (IJMIP)* 1(1):26–34
- Taylor M, Haggerty J, Gresty D, Lamb D (2011) Forensic investigation of cloud computing systems. *Netw Secur* 2011(3):4–10
- Grispos G, Storer T, Glisson WB (2013) Calm before the storm: the challenges of cloud. *Int J Digit Crime Forensic (IJDCF)* 4(2):28–48
- Ruan K, James J, Carthy J, Kechadi T (2012) Key terms for service level agreements to support cloud forensics. In: IFIP International Conference on Digital Forensics. Springer, Berlin Heidelberg. pp 201–212
- Nanda S, Hansen RA (2016) Forensics as a service: Three-tier architecture for cloud based forensic analysis. In: 2016 15th International Symposium on Parallel and Distributed Computing (ISPD). pp 178–183. <https://doi.org/10.1109/ISPD.2016.31>
- Harnik D, Pinkas B, Shulman-Peleg A (2010) Side channels in cloud services: Deduplication in cloud storage. *IEEE Secur Priv* 8(6):40–47. <https://doi.org/10.1109/MSP.2010.187>
- Bonwick J, Moore B (2007) ZFS: the last word in file systems. https://wiki.illumos.org/download/attachments/1146951/zfs_last.pdf. Accessed 18 Sept 2018
- Roussev V, Richard III GG (2004) Breaking the performance wall: The case for distributed digital forensics. In: Proceedings of the 2004 Digital Forensics Research Workshop, vol. 94, Baltimore
- Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. *Commun ACM* 51(1):107–113
- Roussev V, Wang L, Richard G, Marziale L (2009) A cloud computing platform for large-scale forensic computing. In: IFIP International Conference on Digital Forensics. Springer, Berlin Heidelberg. pp 201–214
- Ayers D (2009) A second generation computer forensic analysis system. *Digit Investig* 6:34–42
- Garfinkel SL, McCarrin M (2015) Hash-based carving: Searching media for complete files and file fragments with sector hashing and hashdb. *Digit Investig* 14:95–105. <https://doi.org/10.1016/j.diin.2015.05.001>
- Garfinkel SL (2013) Digital media triage with bulk data analysis and bulk_extractor. *Comput Secur* 32:56–72
- Young J, Foster K, Garfinkel S, Fairbanks K (2012) Distinct sector hashes for target file detection. *IEEE Computer* 45(12):28–35. <http://doi.ieeeecomputersociety.org/10.1109/MC.2012.327>
- Jones B, Pleno S, Wilkinson M (2012) The use of random sampling in investigations involving child abuse material. *Digit Investig* 9:99–107
- Harris R (2006) Arriving at an anti-forensics consensus: Examining how to define and control the anti-forensics problem. *Digit Investig* 3:44–49
- Garfinkel S (2007) Anti-forensics: techniques, detection and countermeasures. In: 2nd International Conference on i-Warfare and Security, vol. 20087. Edith Cowan University, Perth Western Australia. pp 77–84
- Kessler GC (2007) Anti-forensics and the digital investigator. In: Australian Digital Forensics Conference. Edith Cowan University, Perth Western Australia. p. 1. <https://doi.org/10.4225/75/57ad39ee7ff25>
- Schneider B, Kelsey J (1999) Secure audit logs to support computer forensics. *ACM Trans Inf Syst Secur (TISSEC)* 2(2):159–176
- Zawoad S, Dutta AK, Hasan R (2013) SecLaaS: secure logging-as-a-service for cloud forensics. In: Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security. ACM. pp 219–230
- Rosenblum M, Ousterhout JK (1992) The design and implementation of a log-structured file system. *ACM Trans Comput Syst (TOCS)* 10(1):26–52. <https://doi.org/10.1145/146941.146943>
- Cornell B, Dinda PA, Bustamante FE (2004) Wayback: A user-level versioning file system for linux. In: Proceedings of USENIX Annual Technical Conference, FREENIX Track, Boston. pp 19–28
- Strunk JD, Goodson GR, Scheinholtz ML, Soules CA, Ganger GR (2000) Self-securing storage: protecting data in compromised system. In: Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation (OSDI). Vol. 4. p 12. USENIX Association
- Morrey C, Grunwald D (2003) Peabody: The time travelling disk. In: Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings. 20th IEEE/11th NASA Goddard Conference On. IEEE. pp 241–253
- Xu J, Swanson S (2016) NOVA: a log-structured file system for hybrid volatile/non-volatile main memories. In: 14th USENIX Conference on File and Storage Technologies (FAST '16), Santa Clara. pp 323–338
- Lee C, Sim D, Hwang JY, Cho S (2015) F2FS: a new file system for flash storage. In: 13th USENIX Conference on File and Storage Technologies (FAST '15), Santa Clara. pp 273–286
- Vrable M, Savage S, Voelker GM (2012) Bluesky: A cloud-backed file system for the enterprise. In: Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST '12). USENIX Association, San Jose. p 19. <https://www.usenix.org/system/files/conference/fast12/vrable.pdf>. Accessed 18 Sept 2018
- McLoughlin M, The QCOW2 Image Format. <http://people.gnome.org/~markmc/qcow-image-format.html>. Accessed 1 March 2017
- Microsoft (2006) Virtual Hard Disk Image Format Specification, Version 1.0. http://download.microsoft.com/download/f/f/e/ffef50a5-07dd-4cf8-aaa3-442c0673a029/Virtual%20Hard%20Disk%20Format%20Spec_10_18_06.doc. Accessed 18 Sept 2018
- Garfinkel S, Nelson A, White D, Roussev V (2010) Using purpose-built functions and block hashes to enable small block and sub-file forensics. *Digit Investig* 7:13–23. <https://doi.org/10.1016/j.diin.2010.05.003>
- Garfinkel SL (2007) Carving contiguous and fragmented files with fast object validation. *Digit Investig* 4:2–12. <https://doi.org/10.1016/j.diin.2007.06.017>
- Govdocs1, Digital Corpora. <http://digitalcorpora.org/corpora/govdocs>. Accessed 1 March 2017
- Stevens CE (2008) Information technology - AT Attachment 8 - ATA/ATAPI command set (ATA8-ACS). ANSI, Working Draft Project American National Standard, Revision, 6a
- Borthakur D, et al. (2008) HDFS architecture guide. Hadoop Apache Proj. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.pdf. Accessed 18 Sept 2018

47. Miner D, Shook A (2012) *MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems*. "O'Reilly Media, Inc.", Sebastopol, California
48. Rivest R (1992) The MD5 message-digest algorithm. RFC 1321. <http://www.rfc-editor.org/info/rfc1321>. Accessed 18 Sept 2018
49. Xen Blktap2 Driver. <https://wiki.xen.org/wiki/Blktap2>, Accessed 1 March 2017
50. Meyer DT, Aggarwal G, Cully B, Lefebvre G, Feeley MJ, Hutchinson NC, Warfield A (2008) Parallax: virtual disks for virtual machines. In: ACM SIGOPS Operating Systems Review, vol. 42. ACM, New York. pp 41–54
51. Warfield A, Hand S, Fraser K, Deegan T (2005) Facilitating the development of soft devices. In: USENIX Annual Technical Conference, General Track. pp 379–382. http://usenix.org/publications/library/proceedings/usenix05/tech/general/full_papers/short_papers/warfield/warfield.pdf. Accessed 18 Sept 2018
52. Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A (2003) Xen and the art of virtualization. In: ACM SIGOPS Operating Systems Review, vol. 37. ACM, New York. pp 164–177
53. Garfinkel S, Farrell P, Roussev V, Dinolt G (2009) Bringing science to digital forensics with standardized forensic corpora. *Digit Investig* 6:S2–S11. <https://doi.org/10.1016/j.diin.2009.06.016>
54. Roussev V (2010) Data fingerprinting with similarity digests. In: Chow K-P, Sheno S (eds). *Advances in Digital Forensics VI*. Springer Berlin Heidelberg, Berlin, Heidelberg. pp 207–226
55. Stevens M (2006) Fast collision attack on md5. *IACR Cryptol ePrint Arch* 2006:1–13. <https://eprint.iacr.org/2006/104.pdf>. Accessed 18 Sept 2018
56. Wang X, Yin YL, Yu H (2005) Finding collisions in the full SHA-1. In: *Annual International Cryptology Conference*. Springer, Berlin Heidelberg. pp 17–36
57. Wang H, Wang S (2004) Cyber warfare: steganography vs. steganalysis. *Commun ACM* 47(10):76–82
58. Maene P, Götzfried J, de Clercq R, Müller T, Freiling F, Verbauwhede I (2018) Hardware-based trusted computing architectures for isolation and attestation. *IEEE Trans Comput* 67(3):361–374
59. Shinagawa T, Eiraku H, Tanimoto K, Omote K, Hasegawa S, Horie T, Hirano M, Kourai K, Oyama Y, Kawai E (2009) BitVisor: a thin hypervisor for enforcing I/O device security. In: *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. ACM, New York. pp 121–130. <http://doi.acm.org/10.1145/1508293.1508311>
60. Yu L, Weng C, Li M, Luo Y (2011) SNPdisk: an efficient para-virtualization snapshot mechanism for virtual disks in private clouds. *IEEE Netw* 25(4):20–26. <https://doi.org/10.1109/MNET.2011.5958004>
61. King ST, Dunlap GW, Chen PM (2005) Debugging operating systems with time-traveling virtual machines. In: *Proceedings of the Annual Conference on USENIX Annual Technical Conference, Anaheim*. pp 1–15. <https://www.usenix.org/legacy/events/usenix05/tech/general/king/king.pdf>. Accessed 18 Sept 2018
62. Dunlap GW, King ST, Cinar S, Basrai MA, Chen PM (2002) ReVirt: enabling intrusion analysis through virtual-machine logging and replay. *ACM SIGOPS Operating Systems Review*. In: *OSDI '02: Proceedings of the 5th Symposium on Operating Systems Design and Implementation Vol. 36*. pp 211–224. <https://doi.org/10.1145/844128.844148>
63. Taguchi JK (2013) Optimal sector sampling for drive triage. *NAVAL POSTGRADUATE SCHOOL MONTEREY CA*. <https://calhoun.nps.edu/handle/10945/34750>. Accessed 18 Sept 2018
64. Bloom BH (1970) Space/time trade-offs in hash coding with allowable errors. *Commun ACM* 13(7):422–426. <https://doi.org/10.1145/362686.362692>
65. Dykstra J, Sherman AT (2012) Acquiring forensic evidence from infrastructure-as-a-service cloud computing: Exploring and evaluating tools, trust, and techniques. *Digit Investig* 9:90–98. <https://doi.org/10.1016/j.diin.2012.05.001>
66. Dykstra J, Sherman AT (2013) Design and implementation of FROST: Digital forensic tools for the OpenStack cloud computing platform. *Digit Investig* 10:87–95. <https://doi.org/10.1016/j.diin.2013.06.010>
67. Zawoad S, Hasan R (2015) A TRUSTWORTHY CLOUD FORENSICS ENVIRONMENT, *Advances in Digital Forensics XI*. Springer International Publishing, Cham. pp 271–285
68. Richard III GG, Roussev V (2006) Next-generation digital forensics. *Commun ACM* 49(2):76–80

Submit your manuscript to a SpringerOpen[®] journal and benefit from:

- Convenient online submission
- Rigorous peer review
- Open access: articles freely available online
- High visibility within the field
- Retaining the copyright to your article

Submit your next manuscript at ► springeropen.com
