Distributed Cloud Forensic System with Decentralization and Multi-participation

Xuanyu Liu[1], Xiao Fu[1(✉)], Bin Luo[1], and Xiaojiang Du[2]

[1] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
dzl532002@smail.nju.edu.cn,
{fuxiao,luobin}@nju.edu.cn
[2] Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA
dxj@ieee.org

Abstract. A considerable number of cloud forensic systems and tools have been proposed in recent years. Trust issue of digital evidence, a significant security topic, is indispensable for cloud forensics systems. In this paper, we propose a different cloud forensic system—Distributed Cloud Forensic System with Decentralization and Multi-participation (DCFS). The DCFS is set in an untrusted and multi-tenancy cloud environment, and it is assumed that cloud users, cloud employees, or forensic investigators can be dishonest. The DCFS, which is different from existing centralized cloud forensic systems, is a distributed and decentralized system that does not rely on any single node or any third party to obtain credible evidence from the cloud. Trust is divided into all participants in the DCFS, and these participants supervise each other. A distributed public ledger is maintained in the DCFS, and this ledger records all the proofs of forensic evidence along with other useful information. This ledger can enhance the credibility and integrity of forensic evidence to some degree and complete the chain of custody in forensic investigation. The forensic evidence, which are provided by the cloud employees, presented to the court of law using the DCFS will be more trustful.

Keywords: Cloud forensics · Data provenance · Byzantine faults
Distributed systems · Decentralization · Multi-participation

1 Introduction

Cloud forensics is a cross discipline of cloud computing and digital forensics. Digital forensics is the application of computer science principles to recover electronic evidence for presentation in a court of law. Most of the existing cloud forensic systems and tools are set in non-adversarial environment, and they precisely consider the

This work is supported by the National Natural Science Foundation of China (61100198/F0207, 61100197/F0207).

C. Li and S. Mao (Eds.): WiCON 2017, LNICST 230, pp. 181–196, 2018.
https://doi.org/10.1007/978-3-319-90802-1_16

external threats. That is, they trust the cloud service providers completely. In return, this brings the honesty issue of the cloud service providers. Forensic investigators and court authorities may also doubt about the credibility and validity of evidence. It is undeniable that a few studies have tried to reveal the trust issues in cloud forensics. They prefer to trust the cloud service providers partially or reduce their trusted computing base by introducing some trusted components such as monitor, hardware, or database into their methods. In a multi-tenancy environment, they may attempt to pick a trusted third party [1] as an intermediary to store or verify forensic evidence. In short, these solutions are centralized so that their functions rely on a single point. Moreover, the trust of forensic evidence is established on the basis of this point. Unfortunately, neither a single component nor a third party is invariantly trustworthy. Components may exhibit loopholes or bugs and could be compromised by an adversary. A third party can also collude with malicious individuals to hide their crimes for the purpose of illegal income. Both of them can become the single failure point and can cause performance and security issues easily. Existing cloud forensic systems are unsatisfactory to some extent. It is necessary to have alternative forensic systems or tools that do not require this type of trust.

In this paper, we propose a different cloud forensic system—Distributed Cloud Forensic System with Decentralization and Multi-participation (DCFS) from another perspective. The DCFS is set in an untrusted and adversarial environment. It does not trust any single node or any single person and considers both internal and external threats. The DCFS is a distributed cloud forensic system. It does not precisely operate for a single node but for a large-scale network with numerous nodes. These nodes act together to build a more secure and more robust cloud forensic ecosystem. Decentralization indicates that the DCFS does not rely on a trusted component or a trusted third party. Trust is divided among the nodes in the DCFS, and these nodes supervise each other. Multi-participation indicates that the DCFS acts in a multi-tenancy environment with various stakeholders, including users, cloud employees, forensic investigators, and court authorities. It has the assumption that cloud customers, cloud service providers, or forensic investigators can be malicious or dishonest; they may collude with each other to provide faked forensic evidence or may frame innocent people.

Data provenance is selected as the primary raw data of forensic evidence in the DCFS. Data provenance determines and describes the lifecycle history of data sets from original resources to destruction endpoints. It is helpful for analyzing what happened to certain dataset and estimating its scope of influence among the systems. The accountability of the cloud can be enhanced using data provenance. To make data provenance more available and more credible, a public data provenance ledger is introduced into the DCFS. This ledger has something in common with the public ledger maintained in Blockchain [2] systems. A Blockchain system is essentially a distributed ledger of all transactions or digital events executed and shared among all participants in accounting systems. Each transaction in the public ledger is verified by consensus of a majority of the participants. Once recorded, these transactions can never be erased. With same purposes, the public data provenance ledger in the DCFS is a distributed ledger of all proofs of data provenance and other valuable information useful for further forensic investigation. The participants in the DCFS involve in the maintenance of this ledger and reach a consensus on its entries. Every data provenance recorded in the

ledger is verifiable, accountable, and immutable. Any misbehavior, which can be Byzantine [3], performed to the ledger is not hidden and is rejected. With this, we develop a democratic open and scalable forensic system from a centralized one. To the best of our knowledge, the DCFS is the first decentralized cloud forensic system that do not rely on any single point or third party as well as no single point or third party has absolute power to affect the forensic process. The DCFS can enhance the credibility and integrity of forensic evidence, enhance the accountability and robustness of cloud systems, and complete the chain of custody in forensic investigations. To demonstrate the practicality and security of the DCFS, we have implemented a prototype on OpenStack, which is an open source cloud computing platform. Our evaluation demonstrates the ability of the DCFS to solve the real world forensic problems, and the results reveal that the costs of the DCFS (CPU load, network latency and storage) are sufficiently low to be practical.

Motivation. Most of the existing cloud forensic systems trust their cloud service providers completely or partially and trust forensic investigators acquiescently. The DCFS, which is different from those forensic systems, does not believe in cloud employees and forensic investigators. The goals of the DCFS are as follows:

- Make forensic evidence from cloud more available and accessible.
- Enhance the credibility and integrity of forensic evidence.
- Ensure that any evidence entry presented in front of the court is verifiable.
- No one can tamper with any evidence entry and any misbehavior performed to evidence entries will be discovered and prevented.
- Malicious individuals can never repudiate evidence indicating them.
- No one can recover any valuable information from the DCFS so that the privacy of the user remains protected.

Contributions. The contributions of this paper are as follows:

- We proposed the DCFS. The DCFS is set in untrusted and adversarial multi-tenancy environment and does not rely on any single trusted node or third party. The DCFS overcomes the drawbacks of centralized cloud forensic systems and makes the cloud more accountable and robust.
- We design a public data provenance ledger for the DCFS. This ledger helps the forensic investigators and court authorities to obtain valid and credible evidence. Moreover, it can prevent malicious individuals from tampering or denying the forensic evidence after the fact.
- The DCFS is implemented and evaluated on OpenStack.

Organization. The structure of the paper is organized as follows: Sect. 2 discusses some related studies. Section 3 introduces the design of the DCFS in detail. Then, Sect. 4 provides the security analysis of the DCFS. Section 5 provides the implementation and evaluation of the DCFS, and Sect. 6 concludes this paper.

2 Related Work

There have been several studies on collecting and providing data provenance. PASS [13] is a typical provenance system in the system level. It modifies the Linux kernel and intercepts system call in the VFS level. Based on PASS, a provenance system for XEN [14] is established. SPADE [15] focuses on the related primitive operation on data input and output, and detects system calls of files and processes. LineageFS [16] associates the process ID with file descriptors and creates lineage information for files. Hi-Fi [17] includes various types of nonpersistent data into data provenance. LPM [18] is deployed in Linux kernel and is designed to collect system-wide data provenance, including process, IPC, network, and system call. Based on LSM, the LPM set provenance hooks along with LSM hooks, ensuring that the data provenance represents the proper actions of the system. DPAPI [19], CPL [20], and IPAPI [21] are set in the application level. They provide specific provenance API so that the application developers can invoke these APIs to make their applications provenance aware. HadoopProv [22] realizes a provenance system in Hadoop. In Android platform, Quire [23] extracts the provenance data from IPC and RPC to construct invocation chain. By analyzing this invocation chain, the potential attacks such as excess of authority can be discovered. Similar to Quire, Scippa [24] expands the Binder module in Android system to construct an invocation chain.

However, these systems or tools still lack in providing trustworthy data provenance. They do not consider the internal threats, and they depend only on their host systems. Some researchers tried to solve these issues. The SNP [25] is a network provenance system running in an adversarial setting. Every node manages its own tamper-evident network logs. By querying the relevant information from other nodes, fault nodes can be discovered. However, the SNP relies on some types of behaviors on the network to be observed by at least one correct node, and the error nodes can escape easily from the detection of the SNP by colluding with each other. Moreover, the efficiency of SNP will drop drastically with a reduction in the correct nodes available in the system. Another system, SecLaaS [26], assumes that users, clouds, and investigators can be malicious individually or can collude with each other. It creates the proofs of logs and publishes these proofs on the web for further verification. The drawback of this system is that the proofs are considerably coarse grained, and the proofs remain unprotected after being published. Other researchers attempt to adopt cryptography, trusted computing, or mathematics to enhance the integrity and confidentiality of data provenance, such as ABE [27] and bilinear pairing [28]. The scope of their application is small and may cause considerable resource consumption. Several papers (e.g., [29–34]) have studied related security and networking issues.

Different from them, the DCFS runs in an adversarial multi-tenancy cloud environment. It is a distributed and decentralized forensic system, and it does not rely on any single trusted node or third party. The participants in the DCFS cooperate with each other to make the evidence entries more trustworthy.

3 System Design

3.1 System Structure

In the DCFS, participants are scattered in different cloud systems. A small and smart process named DCFS Peer Agent (DPA) is allocated to every participant. The DPAs act as agents for participants and enable them to join in the DCFS membership network, involve in the DCFS cloud forensic system, manage their personal data, and conduct operation requests. Figure 1 shows the system structure of DCFS, which can be divided into three layers:

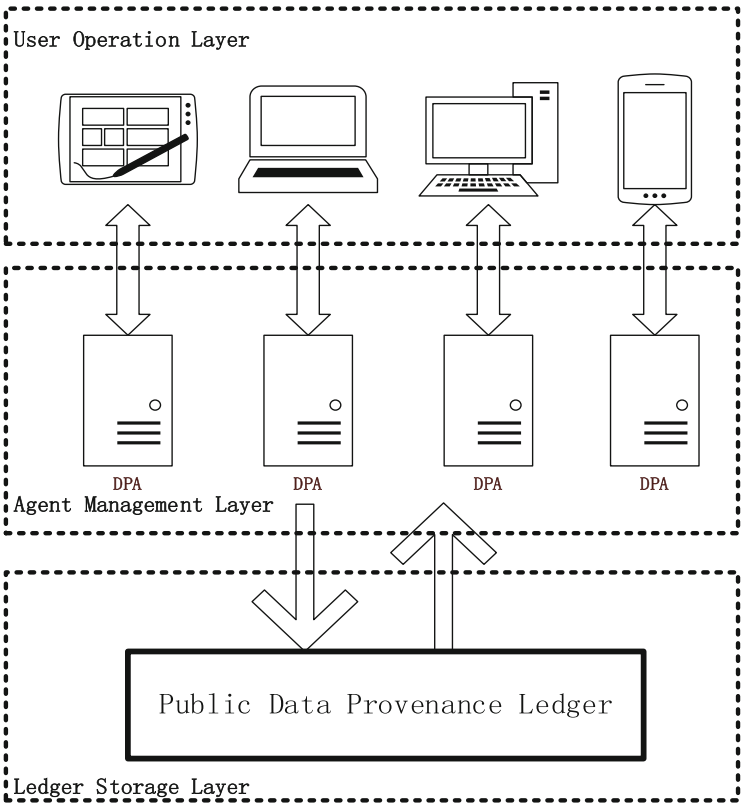

Fig. 1. System structure of the DCFS

User Operation Layer. This layer consists of all participants along with their terminal applications installed in their PCs, smart phones, or even in a third-party cloud platform. These terminals are management consoles for participants to communicate with their DPAs, commit their operation requests, and manage their personal data. They can also manage their individual accounts, check the DCFS network topology, and examine the DCFS runtime information. Investigators and court authorities can use customized terminals to verify the integrity and validity of forensic evidence.

Agent Management Layer. The DPAs are mainly located in this layer. They are independent, and they connect with each other to establish a point-to-point network. Certain consensus algorithm is adopted to ensure that these DPAs are in the same final state and reach a consensus on the public data provenance ledger. Moreover, the DPAs are responsible for managing data entries in the public ledger, acting as a gateway that evaluates all data accesses, and communicating with terminal applications to handle the received operation requests.

Ledger Storage Layer. In this layer, a public provenance ledger offers a scalable, highly available, secure, and independent storage service for data provenance against confidentiality and integrity attacks. This public ledger is not precisely located in a single node or stored on the database of a third party. Instead, it is a distributed ledger and all participants own a full copy of this ledger in their database. The DPAs of the participants reach a consensus on the ledger and ensure the contents of each ledger stored in different participants' database are consistent. Any malicious activity performed to the ledger is unacceptable to the other correct DPAs unless majority of DPAs are under control by an attacker. It is not invariantly easy to control majority of DPAs in a large-scale cloud system. This design guarantees that any data entry recorded in the ledger remains unchanged once generated. No one, including users, cloud employees, or investigators can tamper with the ledger.

3.2 DCFS Peer Agent

As Fig. 2 shows, a DPA mainly consists of three modules: Data Manager, Operation Handler, and Membership Service.

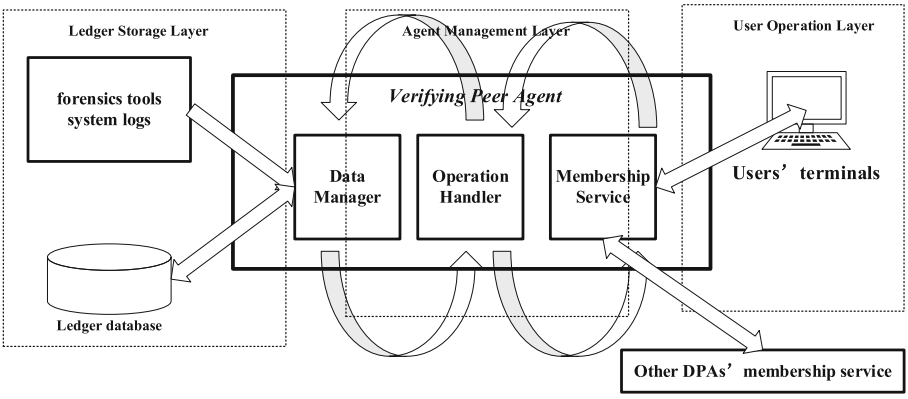

Fig. 2. Modules of the DPA. Data Manager is the medium between Ledger Storage Layer and Agent Management Layer. Membership Service is the medium between Agent Manager Layer and User Operation Layer.

Data Manager. The primary responsibility of Data Manager is to create raw proofs of data provenance using data provenance metadata from forensic tools and other useful system logs. These raw proofs of data provenance will be sent to Operation Handler for

further encapsulation and then sent to Membership Service for consensus process. Data Manager is also in charge of communicating with the database for public ledger.

Operation Handler. Operation Manager handles operation requests such as query and verification. When receiving an operation from a user, Data Manager searches satisfactory data from the public provenance ledger, verifies its validity, and packages the data in a proper manner.

Membership Service. The main duty of Membership Service module is network communication. It communicates with the terminals of the participants to receive operation requests from users and respond to them. It also connects with the other Membership Service modules of the DPAs to build the distributed forensic network. Moreover, it manages and evaluates data access policy for each participant.

3.3 Public Data Provenance Ledger

The public data provenance ledger is the core of the DCFS. Only the proofs of data provenance are included in the ledger for the following reasons:

- Raw data provide huge storage and network consumption while proofs are more lightweight.
- The main goal of the DCFS is to offer the ability to verify the correctness of evidence. The proofs of data provenance are sufficient.
- Malicious individuals may try to learn about some crucial information of other users from the ledger. If only proofs are included, nearly no valuable information can be recovered from them. The privacy of the users will remain protected.

The process from raw data to entries in the ledger includes four steps: First, a data provenance graph is generated using the raw data provided by forensic tools and system logs. Second, a new proof is created based on the data provenance graph. Third, with necessary encapsulation, the proof is added into the ledger. Finally, a consensus process is started between DPAs to ensure that all DPAs agree and accept this proof.

Data Provenance Graph. Forensic tools for cloud computing should be compatible with the cloud's characteristics of on-demand self-service, rapid elasticity, and scalability. This indicates that the data provenance model should be easy to generate and manage, be open, extensible, and scalable, and be compatible with existing forensic formats and follow existing practices and standards. Based on the above consideration, Open Provenance Model (OPM) [4] is selected as the standard data provenance format in the DCFS. The OPM defines data provenance in a precise and technology-agnostic manner and allows multiple levels of data description to coexist. It allows provenance information to be exchanged easily between systems based on a shared provenance model.

The provenance graph is a directed acyclic graph (DAG), and it implies causal relationships between states and operations. To capture transitive provenance, we can define, for any execution e, a provenance graph $G(e) = (V(e), E(e))$, in which each vertex $v \in V(e)$ represents a state or operation, and each edge (v1, v2) represents that v1 causes v2. It also indicates that the data provenance of v1 is a part of that of v2.

From the provenance graph, we can easily identify the problem's origin and estimate the scope of influence it had created to the entire system. It is helpful for forensic investigations.

The entire system may face the problem of provenance explosion. To reduce the amount and complexity of data provenance and relieve the burden of the system, methods from [5, 6] are adopted. The data provenance entries, having little or no impact on forensic analysis, will be recycled. Therefore, the size of the data provenance graph is reduced, and the proofs can be easily generated.

Proof of Data Provenance. Proofs should be easy to use and hard to forge. Many systems use Merkle trees [7] as their basic data structure for verification, such as Bitcoin and P2P network. Mostly, it is a binary tree, but it can also be a multi-way tree. The value of leaf nodes in the tree is the cell data in the dataset or its hash value. The value of a nonleaf node is calculated by hashing all its child nodes' values. Recursively, the hash value of the tree root is generated, and the entire tree is completed. Merkle tree stores the summary information about a large dataset to make the verification more efficient. It is unnecessary to reveal or transmit the entire tree, and it natively enables a user to validate the integrity of any subset of data.

The DCFS selects the Merkle tree as its data format for proofs. The steps from the data provenance graph to the Merkle tree are shown in Fig. 3. First, a topological sort of all nodes in the data provenance graph is generated. Because the topological sort of a directed acyclic graph is not unique, this step uses timestamps as another parameter. For nodes in the same topological layer, they are sorted again in ascending order by their timestamps. Second, leaf nodes in the Merkle tree are filled with topologically sorted nodes' hash value, and the other part of the tree is calculated based on these leaf nodes. Subsequently, the value at each branch node is calculated by concatenating the values of its children and computing the hash of that aggregation. Finally, the value of the root node is selected as the proof of the data provenance graph.

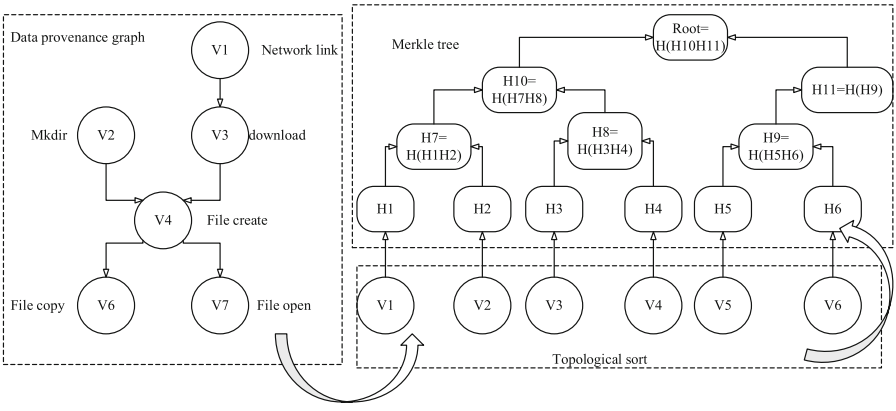

Fig. 3. Steps from data provenance graph to Merkle tree

Proof Chain. The public ledger is a chain of proof entries, and each entry contains the proof of data provenance graph and other essential information to ensure its irreversibility.

To preserve the correct order of the entries in the ledger, an Entry Chain (EC) is introduced. This Entry Chain is a hash chain. Entry Chain will be generated as follows:

$$\mathrm{EC} = <\mathrm{Hash}(\mathrm{Proof}, \mathrm{ECprev}) > \tag{1}$$

where ECprev is the Entry Chain of the previous entry in the ledger, and Proof is the proof information in this entry.

A Proof Entry consists of EC, proofs of data provenance, user id, domain id, and timestamp:

$$\mathrm{Proof\,Entry} = <\mathrm{EC},\ \mathrm{Proof},\ \mathrm{UID},\ \mathrm{DID},\ \mathrm{Timestamp} > \tag{2}$$

Consensus Process. The DCFS is set in an adversarial setting and any misbehavior happened could be Byzantine. To reach a consensus on the public data provenance ledger between DPAs, the DCFS realizes its consensus process based on the PBFT [8] algorithm.

For efficiency and better management, the DPAs will be classified into different domains. For example, the DPAs in the same host machine or having the same cloud administrator will be categorized under the same domain. In each time interval, a domain itself will select a leader DPA to lead affairs. This leader DPA will convey the operation requests, guide the consensus process, and communicate with other domains. When a user desires to perform an operation request, the processing procedure is as follows: First, the DPA of this user receives the operation request and transports it to the leader DPA. Then, the leader DPA will transport the operation request to the other DPAs in its domain and the leader DPAs of the other domains. Next, all the DPAs will handle this operation request and return the results to the DPA of the user. Finally, this DPA handles the received results and return them to the user's terminal.

The pseudocode description of the consensus algorithm is showed in the Appendix.

4 Security Analysis

The cloud has full control over generating the data provenance. The DCFS functions based on an assumption that the process of generating the data provenance is trusted. It focuses on helping the investigators and court authorities to verify the evidence using the proofs included in the DCFS ledger. The participants reach a consensus on this ledger, and trustful proofs are invariantly available. The DCFS guarantees that any violation of ledger's integrity will be prevented and evidence tampered with will finally be detected during the verification process.

In the DCFS, three entities are involved: users, cloud employees, and investigators. All of them can be malicious individually or can collude with each other. However, at the verification stage, any misbehavior can be detected using the public provenance ledger. Figure 4 illustrates the detailed flow of evidence verification. Based on the evidence entries provided by a cloud employee, an investigator can calculate its proofs and fetch the corresponding proofs from the public ledger in the DCFS. If the two proofs are equal, the investigator may trust these evidence entries and present them to the court. Otherwise, he can reject them and doubt the honesty of the cloud employee. When the court receives the evidence, it first checks the ownership information from the ledger to judge whether an innocent is framed. Then, it fetches the corresponding proofs from the public ledger again to compare with the proofs of the received evidence. If they are equal, the court will accept the evidence. Otherwise, the evidence will be rejected, and the court doubts the honesty of the investigator.

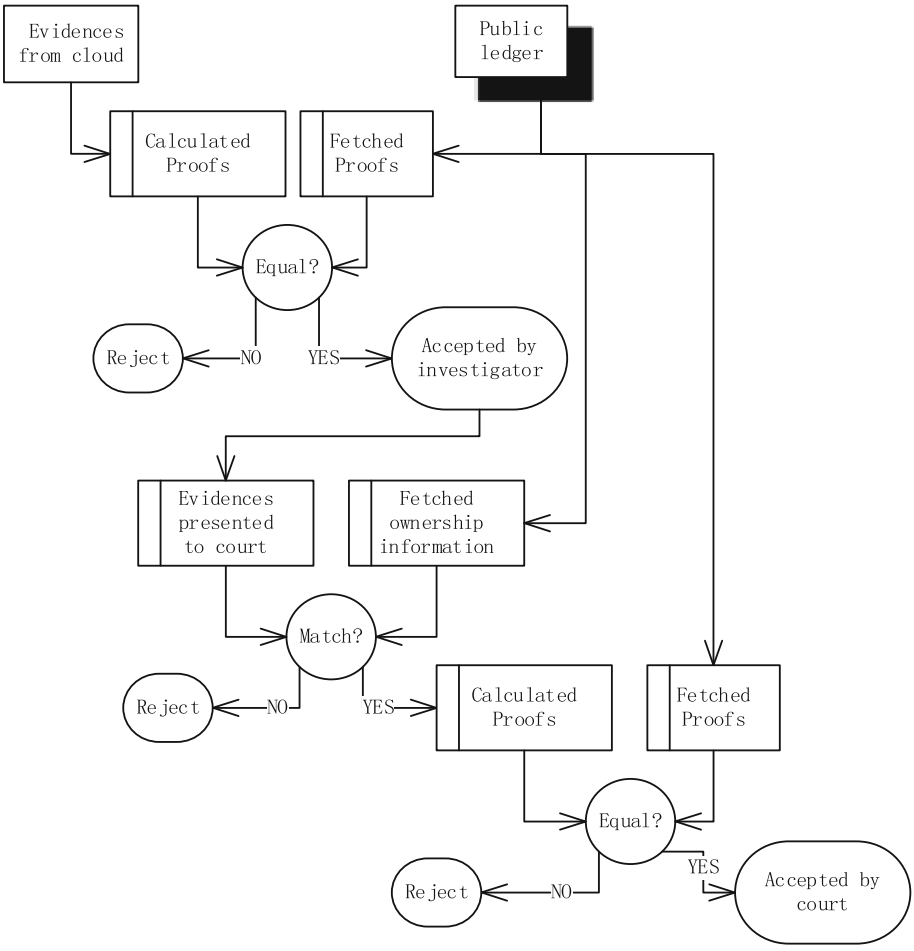

Fig. 4. Process flow of evidence verification

The DCFS is a distributed and decentralized cloud forensic system, and all participants reach consensus on a distributed public ledger. Few malicious participants have limited impact on the entire system unless they can control the majority of the DPAs. We assume that any misbehavior happened in the DCFS can be Byzantine, and we design our consensus process based on the PBFT algorithm. This algorithm guarantees liveness and safety under the premise that provides a fault tolerance of $(n - 1)/3$, where n is the total number of participants. This indicates that malicious participants should not be more than one third of all participants for safety. With a greater number of participants, the DCFS will become more secure and stable. In other words, the DCFS will be more useful in a large-scale system.

A brief proof is as follows: Define n as the number of all nodes and f as the number of faulty nodes. An assumption for asynchronous Byzantine agreement is that correct nodes will send precisely one correct message to others in each phase while the faulty nodes may send more than one message to confuse others. A minimum of $(n + f)/2$ received messages are essential to reach an agreement on a message [9]. It is invariantly possible for a node to accept $n - f$ messages. Consider a correct node n in phase p, where n has already sent a message to all the other nodes. As there exists a minimum of $n - f$ correct nodes, the n's buffer will receive a minimum of $n - k$ reply messages. The $n - k$ should outnumber $(n + f)/2$, i.e., $n - f > (n + f)/2$. Therefore, $f < n/3$.

As only hash results and some necessary identification information are included in the public ledger, the malicious individuals can barely recover any valuable information from the ledger.

5 Implementation and Evaluation

We implemented our prototype system on three desktop computers with Intel(R) Core (TM) i5-3330 3.00 GHz CPU, 8 GB main memory, and 256 KB L2 cache. Ubuntu 14.04 LTS 64-bit was used as Host Operating System, and Openstack was selected for implementation and evaluation. Virtual environment was created with XEN, and each desktop computer runs five VM instances so that there were 15 participants in total. We used SHA-2 (SHA-256) hash function for hashing. A containerization runtime environment for DPAs was established using Docker [10]. Using FROST [11], we obtained API logs from Nova nodes as metadata for generating data provenance.

For easy deployment and management, the DPAs were deployed in Docker containers and were isolated. The gRPC [12] was used for establishing point-to-point network connections between DPAs. Certainly, DPAs can be deployed anywhere as long as there are network capabilities. For example, DPAs can also be embedded into the VM instances of a user or even be located in a third party cloud platform.

The DCFS operates under a normal environment in our experiments. In order to evaluate the impact on the performance of the DCFS, we compared the CPU load, network latency, and storage consumption under the condition of running our forensic system (Table 1).

Table 1. Performance impact

	Performance impact
CPU load	Increasing 2.5% on average
Network latency	Increasing 10% on average
Storage consumption	10 MB a day for each DPA

The system delay was within an acceptable range compared with SNP [25] and SecLaaS [26]. However, at the peak of system operation, the DCFS may have a remarkable effect on the host system. To be practical in normal commercial environment, some optimizations are essential. We have left this for future research.

6 Conclusion and Future Work

Collecting forensic evidence from cloud is a challenging task because forensic investigators have considerably little control over cloud systems. Currently, forensic investigators still depend on cloud service providers to obtain forensic evidence. To the best of our knowledge, there is no procedure to verify whether the cloud service providers have provided the correct evidence to the investigators. Forensic investigators may also present invalid evidence to the court. In this paper, we proposed a different cloud forensic system DCFS from another perspective. The DCFS considers internal threats and provides the ability to securely obtain trustful data provenance for forensic purpose. It can also solve some issues existed in traditional centralized forensic systems. From our experiment, we observed that it is practically feasible to combine the DCFS with the cloud infrastructure.

One limitation of the DCFS is that it still requires some trust in the cloud. In particular, we have to trust that the generation of data provenance from the cloud is correct. This can be relieved by using tamper-evident logging such as Peer Review [35]. In future, we will investigate to make our system more efficient, practical, and expansible so that it can be compatible with more cloud architectures and based on DCFS, we will try to bring cooperative forensics and shared security into more distributed systems or IoT systems.

Appendix

The algorithm of a consensus process on an operation request is as follows:

```
if leader DPA then
   n=deployIdToOperation(received operation request)
   sendStepOneMessageToAll (leader ID, n, checksum, re-
quest)
else if not leader DPA then
   WaitForStepOneMessage()
end if
m1=receivedStepOneMessage
if m1.leaderID is right&&m1.checksum is right&&m1.n is
never used&& m1.n is within minimum and maximum then
    AcceptStepOneMessage()
    sendStepTwoMessageToOthers(leader ID, this DPA's
own ID, n, checksum, request)
    waitForStepTwoMeaasgeFromOthers()
else
    doNothing()
end if
Message[ ] m2=all receivedStepTwoMessage
na2=numOfAcceptedStepTwoMessage
define F as the tolerable maximum of fault DPAs
for all m in m2 do
    if m.leaderID is right&&m. DPA's own ID is tight&&
m.checksum is right&&m.n is never used&&m.n is within
minimum and maximum&&m.StepOneMessage is accepted then
     AcceptStepTwoMessage()
     na2++
   end if
end for
If na2>2F then
    sendStepThreeMessageToOthers(leader ID, this DPA's
own ID, n, checksum, request)
    waitForStepThreeMeaasgeFromOthers()
else
```

```
    doNothing()
end if
Message[ ] m3=all receivedStepThreeMessage
Na3=numOfAcceptedStepThreeMessage
for all m in m3 do
   if m.leaderID is right&&m. DPA's own ID is tight&&
m.checksum is right&&m.n is never used&&m.n is within
minimum and maximum&&m.StepTwoMessage is accepted then
     AcceptStepThreeMessage()
     na3++
   end if
end for
If na3>2F then
recordOperationRequestWithIDLocally(n)
else
    doNothing()
end if
```

References

1. Santos, N., Gummadi, K.P., Rodrigues, R.: Towards trusted cloud computing. HotCloud **9** (9), 3 (2009)
2. Pilkington, M.: Blockchain technology: principles and applications (2015)
3. Lamport, L., Shostak, R., Pease, M.: The Byzantine general problem. ACM Trans. Program. Lang. Syst. (TOPLAS) **4**(3), 382–401 (1982)
4. Moreau, L., Clifford, B., Freire, J., et al.: The open provenance model core specification (v1.1). Future Gener. Comput. Syst. **27**(6), 743–756 (2011)
5. Lee, K.H., Zhang, X., Xu, D.: High accuracy attack provenance via binary-based execution partition. In: NDSS (2013)
6. Lee, K.H., Zhang, X., Xu, D.: LogGC: garbage collecting audit log. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, pp. 1005–1016. ACM (2013)
7. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 369–378. Springer, Heidelberg (1988). https://doi.org/10.1007/3-540-48184-2_32
8. Castro, M., Liskov, B.: Practical Byzantine fault tolerance. In: OSDI, vol. 99, pp. 173–186 (1999)
9. Bracha, G., Toueg, S.: Asynchronous consensus and broadcast protocols. J. ACM (JACM) **32**(4), 824–840 (1985)

10. Merkel, D.: Docker: lightweight linux containers for consistent development and deployment. Linux J. **2014**(239), 2 (2014)
11. Dykstra, J., Sherman, A.T.: Design and implementation of FROST: digital forensic tools for the OpenStack cloud computing platform. Digit. Invest. **10**, S87–S95 (2013)
12. gRPC Homepage. http://www.grpc.io/
13. Muniswamy-Reddy, K.K., Holland, D.A., Braun, U., et al.: Provenance-aware storage systems. In: USENIX Annual Technical Conference, General Track, pp. 43–56 (2006)
14. Macko, P., Chiarini, M., Seltzer, M., et al.: Collecting provenance via the Xen Hypervisor. In: TaPP (2011)
15. Gehani, A., Tariq, D.: SPADE: support for provenance auditing in distributed environments. In: Narasimhan, P., Triantafillou, P. (eds.) Middleware 2012. LNCS, vol. 7662, pp. 101–120. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35170-9_6
16. Sar, C., Cao, P.: Lineage file system, pp. 411–414 (2005). http://crypto.stanford.edu/cao/lineage.html
17. Pohly, D.J., McLaughlin, S., McDaniel, P., et al.: Hi-Fi: collecting high-fidelity whole-system provenance. In: Proceedings of the 28th Annual Computer Security Applications Conference, pp. 259–268. ACM (2012)
18. Bates, A.M., Tian, D., Butler, K.R.B., et al.: Trustworthy whole-system provenance for the Linux Kernel. In: Usenix Security, pp. 319–334 (2015)
19. Muniswamy-Reddy, K.K., Braun, U., Holland, D.A., et al.: Layering in provenance systems. In: USENIX Annual Technical Conference (2009)
20. Macko, P., Seltzer, M.A.: General-purpose provenance library. In: TaPP (2012)
21. Carata, L., Sohan, R., Rice, A., et al.: IPAPI: designing an improved provenance API. Presented as Part of the 5th USENIX Workshop on the Theory and Practice of Provenance (2013)
22. Akoush, S., Sohan, R., Hopper, A.: HadoopProv: towards provenance as a first class citizen in MapReduce. In: TaPP (2013)
23. Dietz, M., Shekhar, S., Pisetsky, Y., et al.: QUIRE: lightweight provenance for smart phone operating systems. In: USENIX Security Symposium, vol. 31 (2011)
24. Backes, M., Bugiel, S., Gerling, S., Scippa: system-centric IPC provenance on Android. In: Proceedings of the 30th Annual Computer Security Applications Conference, pp. 36–45. ACM (2014)
25. Zhou, W., Fei, Q., Narayan, A., et al.: Secure network provenance. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, pp. 295–310. ACM (2011)
26. Zawoad, S., Dutta, A.K., Hasan, R.: SecLaaS: secure logging-as-a-service for cloud forensics. In: Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security, pp. 219–230. ACM (2013)
27. Li, J., Chen, X., Huang, Q., et al.: Digital provenance: enabling secure data forensics in cloud computing. Future Gener. Comput. Syst. **37**, 259–266 (2014)
28. Lu, R., Lin, X., Liang, X., et al.: Secure provenance: the essential of bread and butter of data forensics in cloud computing. In: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, pp. 282–292. ACM (2010)
29. Cheng, Y., Fu, X., Du, X., Luo, B., Guizani, M.: A lightweight live memory forensic approach based on hardware virtualization. Inf. Sci. **379**, 23–41 (2017)
30. Fu, X., Du, X., Luo, B.: Data correlation-based analysis method for automatic memory forensics. Secur. Commun. Netw. **8**(18), 4213–4226 (2015)
31. Wu, L., Du, X.: MobiFish: a lightweight anti-phishing scheme for mobile phones. In: Proceedings of the 23rd International Conference on Computer Communications and Networks (ICCCN), Shanghai, China, August 2014

32. Wu, L., Du, X., Fu, X.: Security threats to mobile multimedia applications: camera-based attacks on mobile phones. IEEE Commun. Mag. **52**(3), 80–87 (2014)
33. Du, X., Xiao, Y., Guizani, M., Chen, H.H.: An effective key management scheme for heterogeneous sensor networks. Ad Hoc Netw. **5**(1), 24–34 (2007)
34. Du, X., Guizani, M., Xiao, Y., Chen, H.H.: A routing-driven elliptic curve cryptography based key management scheme for heterogeneous sensor networks. IEEE Trans. Wirel. Commun. **8**(3), 1223–1229 (2009)
35. Haeberlen, A., Kouznetsov, P., Druschel, P.: PeerReview: practical accountability for distributed systems. ACM SIGOPS Oper. Syst. Rev. **41**(6), 175–188 (2007)