SIV: A Structural Integrity Verification Approach of Cloud Components with Enhanced Privacy

Bo Zhao, Peiru Fan*, Pengyuan Zhao, Mingtao Ni, and Jinhui Liu

Abstract: Private data leakage is a threat to current integrity verification schemes of cloud components. To address this issue, this work proposes a privacy-enhancing Structural Integrity Verification (SIV) approach. It is made up of three processes: proof organization, proof transformation, and integrity judgement. By introducing a Merkle tree technique, the integrity of a constituent part of a cloud component on a node is represented by a root value. The value is then masked to cipher texts in proof transformation. With the masked proofs, a structural feature is extracted and validated in an integrity judgement by a third-party verification provider. The integrity of the cloud component is visually displayed in the output result matrix. If there are abnormalities, the corrupted constituent parts can be located. Integrity is verified through the encrypted masked proofs. All raw proofs containing sensitive information stay on their original nodes, thus minimizing the attack surface of the proof data, and eliminating the risk of leaking private data at the source. Although some computations are added, the experimental results show that the time overhead is within acceptable bounds.

Key words: integrity verification; cloud components; structural feature; privacy

1 Introduction

Nowadays, a significant part of our lives depends on various cloud services. The beneficial characteristics of the cloud service paradigm, e.g., low cost, fast deployment, and high flexibility, facilitate its wide adoption. Security is essential because of the sensitivity of cloud data and resources. However, due to the complexity of cloud architecture and operations, security evaluation remains an ongoing research objective[1].

Integrity verification is an important aspect of securing cloud services. Integrity verification is able to detect violations of software or predefined security policies, and also play an integral role in many other security techniques, such as remote attestation, intrusion detection, and trust evaluation. Integrity verification research has led to many improvements, but security defects remain. Most existing solutions are designed for stand-alone systems, performing a one-time verification of a single object. The cloud presents a different situation with new challenges; a cloud component usually contains multiple constituent parts distributed on separate nodes and cooperating towards the common function that the component is designed to accomplish. For testing the integrity of a cloud component, all of its constituent parts and dependencies need to be evaluated.

There are two crucial points in the process of integrity verification: the expression and checking of integrity proofs, and the protection from tampering of these proofs in storage and transmission. Many researchers
have sought the clearest and most accurate forms of proof that can describe the integrity status of an object; for example, raw logs/records\textsuperscript{[2]}, test reports\textsuperscript{[3]}, process contexts\textsuperscript{[4]}, properties\textsuperscript{[5]}, and Platform Configuration Registers (PCRs)\textsuperscript{[6]}. Forms of raw proof, such as records and reports, can provide a solid basis for integrity verification, but present the risk of leaking private data. The direct use of informative proofs like these creates a large attack surface which, if abused, can reveal private details of the objects. Other proofs are simple transformations, like properties and PCRs, but the use of these does not eliminate privacy leakage risks. With PCRs, platform configuration information can be deduced with collision analysis\textsuperscript{[7]}. The privacy threats are further heightened in the cloud. Cross-referencing these proofs with data from other sources (e.g., reports about vulnerabilities and patches) can increase the amount of sensitive information that can be derived about the cloud. Armed with these details, further attacks can be initiated, such as locating victim virtual machines\textsuperscript{[8]} or finding the weakest link to inject malicious codes\textsuperscript{[9]}. Thus privacy issues with the cloud components need to be considered in the integrity verification.

The protection of integrity proofs is another research focus. Representative schemes attempt to establish dedicated secure channels for secure transmission. These schemes are efficient when there is only one object to be verified, but less efficient when the objects are numerous and distributed on separate cloud nodes. Building independent secure channels for each object adds a heavy workload, causing a considerable latency and degrading performance.

We therefore seek to balance proof protection and performance in integrity verification in this paper. Accordingly, we propose a Structural Integrity Verification (SIV) approach for cloud components. Analyzing the deployment and operations of structural cloud components reveals an implicit behavioral pattern, on the foundations of which we present a structural feature to comprehensively express the integrity status of all constituent parts of a cloud component. The feature exists not only in raw proofs but also in encrypted masked proofs, thus the SIV approach is designed to enhance privacy in the integrity verification of a structural cloud component. Our contributions are summarized as follows:

1. We present a structural feature to comprehensively express the integrity of a cloud component. A cloud component is composed of many constituent parts distributed on separate nodes; by observing their deployment and operations, an implicit behavioral pattern is detected. On the basis of this pattern, the structural feature expresses the integrity status of the cloud component in a collective way and can further facilitate the integrity verification of all its constituent parts in a batch.

2. We put forward an effective integrity analysis method that supports encrypted masked proofs. Sensitive information is often carried in raw proofs, such that their use may lead to the leakage of private data. The structural feature presented in our paper is present in both raw and masked proofs, therefore a Third-Party Verification (TPV) service can analyze the integrity of a cloud component based on encrypted masked proofs. The raw proofs can remain on their birth nodes, minimizing the attack service and helping to preserving privacy.

3. We begin to pave the way for verifying the integrity of cloud components in a structural manner. We hope our work can lead to similar evaluations of different cloud environments and inspire more research in the area.

Our paper is organized as follows: Section 2 reviews the related work on integrity verification in cloud environments. Section 3 discusses the challenges, system model, and threat model in this paper. Section 4 states the preliminaries of cloud components and the structural feature. The design of SIV is detailed in Section 5, followed by a security analysis in Section 6 and a performance analysis in Section 7. Section 8 gives the experimental results. We conclude the paper with a discussion of limitations and potential future work in Section 9.

2 Related Work

In general, an integrity verification process consists of several procedures: proof generation, transmission, processing, and analysis. During the whole process, the two crucial points are the expression and protection of proofs.

Raw forms of proof, such as logs, records, and reports, are commonly used for integrity verification. For example, Khan et al.\textsuperscript{[2]} discussed using existing forensic techniques with cloud logs to identify malicious behaviors by attackers. Saibharath and Geethakumari\textsuperscript{[3]} collected virtual machine disk images, logs, etc., to make a similar analysis. Tan et al.\textsuperscript{[4]} collected information on the dynamic running
environment, including memory, processes, CPU, network ports, disk files, and configuration data to prepare for further trustworthiness evaluation. Watson et al.\cite{10} developed an online malware detection approach in cloud computing infrastructures and analyzed both system and network level data directly obtained from virtual machines. However, such direct use of informative raw proofs gives rise to the risk of leakages of private data, as the details of objects can be derived. Thus simple transformations of raw proofs, like properties and PCRs, became popular. For instance, Zhang and Lee\cite{11} mapped the measurements of virtual machines to security properties to monitor their security health. PCRs are also classic proofs used in Integrity Measurement Architecture (IMA) to express the integrity of a platform, but the details of the platform can still be exposed\cite{12}. Leakage risks remain with simple transformation techniques, and the issue is even more serious in cloud environments. With detailed raw proofs, the internal architecture and operating mode of the cloud can be derived, making it easier for an attacker to cause damage. Therefore the risks of leaking private data in the verification of integrity proofs in cloud components deserve special attention.

Protecting proofs remains a focus on the security of storage and collection. In this context, trusted hardware units (such as Trusted Platform Modules (TPM))\cite{13}, Intels Software Guard eXtensions (SGX)\cite{14}, and ARM TrustZone\cite{15}) are commonly used. Sailer et al.\cite{6} secured all the measurements of executable contents in TPMs. Perez et al.\cite{16} presented a vTPM solution, implementing TPM functions for virtual machines. Chen et al.\cite{17} presented a cloud-enhanced design called cTPM to enable integrity verification for mobile applications. Schuster et al.\cite{18} employed SGX processors to realize trustworthy data analytics in isolated memory regions on nodes. Brito et al.\cite{19} leveraged TrustZone technology for secure image processing. These hardware-assisted methods offer good tamper-resistance for proofs, but again most of them are designed for single systems. They are efficient when there is only one object to be verified, but less efficient when the objects are numerous and distributed on separate cloud nodes. Building independent secure channels for each object adds a heavy workload, causing a considerable latency and degrading performance. A balance needs to be struck between proof protection and performance in the integrity verification of cloud components.

3 Model

3.1 Challenges

The deployment and operation of cloud components has a distinctive characteristic, which is that the constituent parts on separate nodes have identical dependent subcomponents in general. This characteristic can assist with the integrity verification of cloud components, as will be illustrated in the next section. First, however, we present the new challenges arising the privacy and performance concerns surrounding proofs that were discussed above:

1. There is a need to strike a balance between the effectiveness of verification and the preservation of privacy. From the point of view of enhancing the accountability and credibility of integrity evaluation, the proofs should be as meticulous and accurate as possible. But this requirement comes into conflict with privacy concerns, given that for any object, the more details that are described, the more information will be revealed. This problem is exacerbated in cloud environments, in which not only will knowledge about the cloud components be revealed, but additional information about the cloud architecture and functionality can also be derived. Thus there is a major challenge involved in expressing the integrity status of a cloud component effectively in a proof while upholding standards of privacy preservation.

2. There is a need to strike a balance between strong protection of proofs and appropriate levels of performance. The proofs are of great use for detecting attack traces, making them a rather appealing target. In existing solutions, individual protection measures are applied to proofs as single objects, as in the case of using a special secure channel for data transmission. This provides a good level of secrecy and integrity, but impedes the batch processing of proofs. For a cloud component with many objects to be verified, protection at the level of individual objects will add significant workload and generate considerable latency. Therefore, we seek a new integrity verification approach that can deal with protected proofs in a batch and achieve appropriate levels of performance.

3.2 System model

Due to the complexity of cloud service models (IaaS, PaaS, and SaaS), professional skills are required to verify the integrity of cloud components. With a lack of necessary resources, it is extremely difficult for individual users to achieve the goals of verification.
A common architecture is to employ a neutral TPV to accomplish the work\cite{20–23}. To enhance universality, our system model is built on this TPV architecture, under which three types of entities are involved: Cloud Service Users (CSU), Cloud Service Providers (CSP), and TPVs. These are depicted in Fig. 1.

The three entities can be described as follows:

1) Cloud Service Users (CSUs) are individuals or organizations making use of cloud services. Their data and applications are outsourced to the cloud. They hope to obtain a fair integrity verification result from an independent TPV to evaluate the trustworthiness of cloud components related to the desired service.

2) Cloud Service Providers (CSPs) include the multiple parties that are involved in the deploying, hosting, managing, and maintaining cloud components to provide services to CSUs. We treat them here as a single entity for simplicity.

3) Third Party Verification (TPV) providers are individuals or organizations with the professional skills to execute the integrity verification of cloud components. As with CSPs, there may be multiple parties included in TPV, but we treat them here as a single entity for simplicity.

The integrity of proofs directly impacts on the fairness of verification results. But the proofs face various threats, as described in the next section.

3.3 Assumptions

We are concerned with the privacy problem in existing integrity verification of cloud components, which comes from the direct use of raw proofs. Sensitive information is contained in these data, and if the proofs are accessed they can reveal details of cloud components and their security status. An attacker can use the raw proof data to identify the weakest links among the cloud security measures and to launch further targeted attacks.

Therefore security holes inside the cloud are outside of the scope of this paper. We assume the proofs are credible, and that they reflect the integrity status of an object faithfully. Our focus is on risks emerging from the TPV providers. Threats of abuse can arise from malicious insiders; the TPV provider may apply data mining or AI techniques to the integrity proofs, gaining additional information and selling it for financial benefit. Inevitably, there are infiltration attacks to cloud nodes, like the VM escape. However, real-world practice shows that though easy to gain control of one or two cloud nodes, it is extremely difficult to gain the most. The costs are enormous. The goal is almost impossible\cite{24}. Thus, we assume that some cloud nodes might be compromised, but the spoilage percentage is in a minority, less than 50%.

TPV providers can also conspire with the CSP to generate inaccurate proofs and deliberately present false integrity verification results to users. This is risky behavior, however. If caught engaging in such unfaithful acts the TPV providers reputation will be damaged to the point that the costs will outweigh any gains. Thus this type of attack is also not considered in our research.

There are other threats, such as eavesdropping and sabotage. They are similar to those faced by common carriers. While they are not detailed here, these threats are taken into consideration in our design. Wherever they may come from, all attacks are following the basic rules of mathematics. A message cannot be decrypted without a correct key, neither can a signature be forged in a strictly limited time.

4 Preliminaries

In this section, we first illustrate the concept of a cloud component. We then present a structural feature based on a pattern implicit in its deployment and mode of operation. The proof is the basis for our SIV approach.

4.1 Cloud component

The cloud component is a collective concept in this paper. It refers to a set of dependent subcomponents distributed on separate cloud nodes to cooperate on a common function. Each group of subcomponents on a node is regarded as a constituent part of the cloud component. The KVM/Xen hypervisor for virtualization, the Apache Hadoop framework for
distributed processing, the OpenStack architecture for cloud management, and the OpenSSL tool for communications are four examples of this kind of cloud component.

Generally, the dependent subcomponents of all constituent parts tend to share the same names, codes, and binaries. Sometimes even the configurations are the same, when they are working for the same higher-level cloud service. This is due to the need for quick deployment and simple management of software. In fact, for simplified sake, all constituent parts of a cloud component tend to come from the same source code repository. Especially due to the large scale adoption of VM cloning techniques and automated deployment in batches, this feature, which forms a realistic foundation for our approach, is common among cloud computing components.

Take the toolkit OpenSSL as an example. To support the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols, a cryptography library must be installed on every cloud node. OpenSSL can then be classified within the scope of cloud components as defined in this paper. In the integrity verification of OpenSSL, all of its constituent parts distributed on the cloud nodes should also be validated. In this situation, the individual verification method of single objects that is offered by existing solutions is not appropriate; we need an approach that executes integrity verification in a batch.

4.2 Structural feature

Since the constituent parts of a cloud component tend to be identical for the sake of simple deployment and operation, their integrity measurements also tend to be identical. An implicit pattern is concealed in the measurements, and based on this we can extract a structural feature to express the integrity of the cloud component.

Again take the OpenSSL to illustrate. If all of the cryptography libraries on cloud nodes match the source taken from a trustworthy code repository selected by the CSP, the cloud component OpenSSL is claimed to be integral.

In an integrity verification, when the OpenSSL is integral, the measurements of all constituent parts (in this case, the separate cryptography libraries) are identical. The equality pattern inside the measurements can be used as a feature to make a judgement on the integrity. If there are minor malicious nodes, violations to the equality pattern will be obvious, because when a library has been tampered with, the measurements will be different from others that are trustworthy. The difference showing in the equality pattern between an trustworthy and a compromised cloud component could therefore be used as the proof to verify integrity. Moreover, the constituent parts can be reviewed in a batch. We call this a structural proof and design our SIV approach on this basis.

We use only one metric, the measurements of constituent parts, in our presentation of the SIV approach for the sake of clarity; our SIV method is extensible and additional metrics can be added to increase the evaluation certainty.

5 Design

We present an SIV approach in this section. An overview is given first, followed by details of the three processes of proof organization, transformation, and integrity judgement. We expect to achieve the goal of raising a fair verification result with an enhanced level of privacy and performance.

5.1 Overview

The SIV approach in a TPV-based verification model is depicted in Fig. 2. There are three entities involved: CSU, CSP, and TPV. A CSU initiates a verification request. The TPV responds to the request and cooperates with the CSP to deploy agents on cloud nodes; the agents are responsible for the proof collection work. The TPV analyzes the proofs and raises results for the CSU. Our work makes some changes to existing interactions. The purple rectangle covers three processes: proof organization, transformation, and analysis. They are the major reforms, and make up the core of the SIV approach.

In SIV, both the organization and transformation of proofs are done inside the cloud. A set of masked proofs are sent to a TPV provider. A structural feature is extracted from the proofs to make an integrity judgement. By transforming raw proofs on the cloud nodes where they were created, the details of the verified objects can be concealed. This eliminates the possibility of leaking private data at the source. Also the use of the structural feature means that the constituent parts, including many dependent subcomponents, of a cloud component can be verified in a batch, which enhances the performance of the SIV approach. The
three processes in SIV are specified in the following sections.

5.2 Proof organization

The function of a cloud component relies on the cooperation of all of its constituent parts on different nodes. A constituent part often consists of many dependent sub components, such as libraries or binary executables. All of these sub components should be considered in an integrity verification. Thus for a constituent part on a node, a set of measurements is usually generated to express its integrity.

We leverage a Merkle hash tree[25] to organize the set of measurements for all subcomponents on a node. Every value in the set is hashed using a one-way cryptographic function. A hash value is arranged as a leaf in the tree. Each pair of the leaves is hashed together to derive an upper node. The process is repeated until only one node remains, which is called the Merkle root. Each measurement in the set relates to the integrity of a subcomponent. If one is tampered with, the corresponding leaf will change, which will change the hash of its upper node, and so on, eventually resulting in a change to the Merkle root. Thus, with this root value the comprehensive integrity status of a constituent part can be rapidly determined.

We illustrate the application of a Merkle hash tree on the measurements set through a constituent part example of OpenSSL 0.9.8k on a Windows system. It relies on two static libraries: libeay32.lib and ssleay32.lib, and four dynamic ones: libeay32.dll, libeay32.lib, ssleay 32.dll, and ssleay32.lib[26]. A Merkle tree is constructed as shown in Fig. 3, made up of four layers. The hashes of the subcomponent measurements are the six leaves on Layer 1 (H00 – H05 on L1). The second tree layer (L2) has an odd number of hashes. Then, to complete the pair hashing process, the last node (H12) is concatenated with itself to form a new upper node in Layer 3 (L3). On merging it with the other node (H20), the Merkle root (H30) can be derived.

Fig. 3 A Merkle hash tree of an OpenSSL constituent part.
For a constituent part of a cloud component on a node, its measurements set is organized in this way to get a Merkle root value. The root value is regarded as a raw proof. The direct use of raw proofs may bring leakage risks, and they need to be transformed before transmission due to privacy concerns. This transformation process is detailed in the next section.

5.3 Proof transformation

A Merkle root value reflects the comprehensive integrity of a constituent part. If the part is unchanged and honest, the root value will be fixed. This means that, with repeated verifications, the frequent use of the root value poses a major threat, leading to a higher potential for successful forgery attacks. Besides which, routine updating or upgrading operations on the cloud component will give rise to uniform changes in the root values of all constituent parts. Cross-referencing this phenomenon with information from other sources, such as the distribution of new component versions or reports on vulnerabilities, allows for the derivation of further details and thus constitutes a threat to cloud privacy. To resolve this issue, we propose a proof transformation process in this section.

We first prepare some notations. The cloud nodes set is denoted as \( D = \{D_0, \ldots, D_n\}, i \in [0, n − 1] \). The total number of nodes is \( n \). The identifier of a node \( D_i \) is \( \text{Id}_i \). The constituent part on node \( D_i \) is denoted as \( \text{MRV}_i \). A masked proof of the root value \( \text{MMRV}_i \) is produced after the transformation; this is the data to be sent to the TPV provider. Each cloud node uses a public-private key-pair \( (PL_i, SK_i) \) to establish identity; the TPV provider also uses a public-private key-pair \( (PL_T, SK_T) \) for the same purpose.

As depicted in Fig. 2, there are two parts to the proof transformation process: message construction and message encryption. In message construction, the body, denoted as \( CP_i \), consists of five parts: a masked proof \( \text{MMRV}_i \), a node identifier \( \text{Id}_i \), a random value \( N \), a timestamp \( T \), and a data signature \( \text{Sig}_i \). The \( \text{MMRV}_i \) are cipher texts from a one-time encryption of \( \text{MRV}_i \). We use a symmetric algorithm \( \text{SEC}() \) and a randomly selected key \( K_{sec} \) provided by the CSP for the encryption. The \( \text{Id}_i \) represents the node identity number; \( N \) is specified in the verification request from the TPV provider, and \( T \) is the current time at the moment when the message is constructed. The \( \text{Sig}_i \) is signed by the data source node and attached to the end of the message. In one-time symmetric encryption the keys are known only to cloud nodes, and should be changed with every verification. In this way, all raw proofs are transformed inside the cloud and cannot be accessed by any other parties, thereby cloud privacy is preserved to its ultimate limit. \( \text{Id}_i \) distinguishes the data source, and \( N \) and \( T \) maintain data freshness. \( \text{Sig}_i \) prevents message counterfeiting and repudiation. For message encryption, we use an asymmetric algorithm \( \text{AEC}() \). The key is \( PK_T \), the public key of the TPV provider. After the asymmetric encryption, the final output of the transformation is denoted as \( M_i \). We illustrate the proof transformation in Algorithm 1.

In proof transformation we use an asymmetric algorithm to encrypt the clear data. Although computation speed is lower than symmetric encryption, the Merkle root \( \text{MRV}_i \) is a hash value with a small and constant size, so the cost is not excessive. When all of the masked proofs have been transferred to the TPV, we are ready for integrity judgement.

5.4 Integrity judgement

The process of integrity judgement has two phases. In Phase 1 we use the structural feature extracted from the masked proofs to make a preliminary integrity judgement. If the cloud component proves to be integral, the verification ends. If violations are detected, a list of compromised constituent parts is made and Phase 2 begins. In Phase 2 we drill down into the compromised constituent parts to locate the compromised subcomponents. Using this two-phase judgement procedure, we can rapidly attain a fine-grained integrity verification result.

When a masked proof message \( M_i \) arrives at the TPV provider and is decrypted, the random value, timestamp, and node signature are validated. If the message is valid, the data \( \text{MRV}_i \) is then arranged as the \( i \)-th element of a

**Algorithm 1 Proof transformation**

**Input:** A Merkle root value \( \text{MRV}_i, i \in [0, n − 1] \)

**Output:** A message \( M_i, i \in [0, n − 1] \)

1. Select a symmetric encryption algorithm \( \text{SEC}() \) and a one-time key \( K_{sec} \) by the CSP. They are known to all cloud nodes. Thus for an \( \text{MRV}_i \), its masked proof is expressed as \( \text{MMRV}_i = \text{SEC}() \times K_{sec} \).
2. Construct the message with the cipher texts \( \text{MMRV}_i \), the node identifier \( \text{Id}_i \), the random value \( N \), and the timestamp \( T \). It is expressed as \( CP_i = \text{MMRV}_i \| \text{Id}_i \| N \| T \| \text{Sig}_i \).
3. Encrypt and sign the message. It is expressed as \( M_i = \text{AEC}() \times PK_T \).
list denoted as $SF = MMRV_i, i \in [0, n - 1]$. Phase 1 then begins in Algorithm 2.

We perform XOR operations on $SF$ to verify whether the structural feature is satisfied. A flag $F$ is used to represent the verification result. If $F$ is true, an integral proof will be raised; if $F$ is false, a list of compromised constituent parts is made and both the CSP and CSU are informed.

In Phase 1, a two-dimensional calculation matrix $CM$ and a one-dimensional result matrix $RM$ are introduced. The former is used to save the binary XOR operation values, and the latter is to intuitively show the normalized results. If all of the values in $RM$ are zero, then all elements in the list $SF$ are known to be identical; we can then infer that all constituent parts meet the equality pattern, the structural feature is satisfied, and therefore the integrity of the cloud component is verified. If some values in $RM$ are not zero, then some elements in list $SF$ are different, and we can infer that the equality pattern is somehow violated. According to our threat model the worst case scenario would be that all compromised constituent parts are tampered with in some way. When the XOR is used to compare the masked proof from compromised parts with that of trustworthy parts in list $SF$, “1” is placed in $CM$. Since the compromised parts are in the minority, the ones in a $CM$ row will be in the majority.

In $RM$ a normalized value greater than 0.5 will point to a compromised constituent part. This forms the basis of our judgement in Phase 1. When there are violations, the second phase begins. This is a targeted verification towards the compromised parts, with the aim of figuring out the specific subcomponents of a part that have been compromised. A set of compromised constituent parts $CCP$ has been devised in Phase 1. A known trustworthy part is added to the set at the beginning of Phase 2, so it can be used for contrast. Additional trustworthy nodes can also be added; there is no upper limit to the number, but one certain credible node is sufficient to establish a baseline.

In this phase the assumption model discussed in Section 3.3 is no longer valid that the compromised cloud nodes are in a minority, so the structural feature fails. However, using the masked proofs of all leaf measurements for constituent parts in the set $CCP$, the specific subcomponents that are compromised can be accurately identified.

$CCP$ is made up of $p$ parts. One is $Id_z, z \in [0, p - 1]$. The symbol $z$ represents the node holding the compromised part. The leaves on $Id_z$ are arranged in a list $H_z = H_{z_d}, d \in [0, q - 1]$. A constituent part has $q$ subcomponents. The masked leaves in an $H_z$ are arranged in a list $MH_z = MH_{z_d}, d \in [0, q - 1]$. Therefore, all masked proofs collected in Phase 2 are denoted as $MH_{CCP} = MH_z, z \in [0, p - 1], d \in [0, q - 1]$. After making the judgements, all compromised subcomponents are recorded in a list, denoted as $QSS = S_{z_d}$. $S_{z_d}$ is a tuple $(z, d)$ telling us exactly which subcomponent of which constituent part is compromised.

Phase 2 is also illustrated in Algorithm 3.

With the two-phase integrity judgement process, a comprehensive status of cloud components can be rapidly provided. When there are violations, all compromised subcomponents can be located accurately.
Algorithm 3  Integrity judgement — Phase 2

Input:  CCP
Output:  QSS
1. Select a known honest node randomly and add it to the CCP.
   There are $p$ nodes in total.
2. Execute the algorithm proof transformation on all leaf measurements. This is expressed as:
   For $z$ in range $[0, p - 1]$
   For $d$ in range $[0, q - 1]$
   $$MH_{zd} = \text{Proof Transformation Algorithm}(H_{zd}).$$
   produce a set $MH_{CCP} = MH_{zd}, z \in [0, p - 1], d \in [0, q - 1]$.  
3. Initialize a blank list QSS.
4. The node $z_0$ is honest, its corresponding masked forms of those leaves are used as contrast.
   For $z$ in range $[1, p - 1]$
   For $d$ in range $[0, q - 1]$
   If $MH_{0d} == MH_{zd}$, then continue.
   Else add a tuple $S_{zd} = (z, d)$ into the QSS.
5. Identify the specific questionable subcomponents on different nodes according to the identifiers in the list QSS.

Throughout the judgement, only masked proofs are required by the TPV provider, with raw proofs remaining where they were generated. In this way the risk of leaking private data during integrity proofs is significantly reduced. Furthermore, instead of verifying the subcomponents as individual constituent parts, they are evaluated in a batch, thereby greatly improving the performance of verifying the integrity of cloud components.

6  Security Analysis

Privacy preservation of a cloud component is a focus of our SIV approach, but equally important are the unforgeability of integrity proofs and the accuracy of locating questionable subcomponents. Hence our security analysis is reported from these three aspects: privacy, unforgeability, and accuracy.

Property 1 (Cloud privacy) The TPV provider can never derive any details from the masked proofs other than the comprehensive integrity of a cloud component, no matter how many times the verifications are repeated.

Proof For a constituent part of a cloud component, only a Merkle root value is produced after the organization process to reflect the parts integrity. In this way all of the measurements of subcomponents are compressed into a hash value. The Merkle root is then transformed into masked proofs-cipher texts generated from one-time symmetric encryption algorithms. Other than extracting the structural feature from the masked proofs for integrity judgement, the TPV provider has no means for decrypting them, let alone deriving cloud details. Besides which, the cipher texts are encrypted with a randomly selected key by the CSP in every verification. For a trustworthy constituent part the masked proof changes in every verification. Cloud privacy is thereby enhanced in our SIV approach, and Property 1 is proved.

Property 2 (Unforgeability) A masked proof coming from a trustworthy cloud node can never be tampered with or forged maliciously.

Proof A masked proof data message is constructed in the proof transformation process. To ensure unforgeability, the message should be sensitive to any violations and, if it is tampered with or forged, the TPV should be able to easily identify the breach. The messages containing masked proofs are unforgeable. They are encrypted by an asymmetric algorithm with the TPV providers public key $PK_t$. Provided that the TPV providers private key $SK_t$ is secured, malicious nodes can never successfully intercept, decrypt, intimate, forge or manipulate any message to deceive the TPV provider. Forging a message before encryption is also impossible, because a random value $N$, specified in the initial verification request, and a timestamp $T$ are attached to the data to ensure freshness. A signature $Sig_i$ issued from its source node is also added to the data to ensure authenticity. Provided that the nodes private key is secured, the signatures are unforgeable. Thus, with these guarantees, Property 2 is proved.

Property 3 (Accuracy) The SIV approach can provide an accurate integrity verification result for a cloud component. If it is integral, SIV can present the proof; if it is not, SIV is able to specify exactly what subcomponents of what constituent parts are compromised.

Proof The TPV provider makes integrity judgements on the masked proofs in a two-phase process. The overall integrity is determined by satisfying a structural feature extracted from the masked proofs. It is inherent from an equality pattern arising from the deployment and operation of the constituent parts of a cloud component. A one-dimensional result matrix $RM$ is produced after Phase 1. If the cloud component is integral, all values in $RM$ are zeros, which serves as the proof. If there are some violations, Phase 2 is initiated. A set of all compromised parts $CCP$ is prepared in Phase 1. A known trustworthy part is added
to the CCP to serve as contrast. The measurements of all the subcomponents of a constituent part are masked and transmitted to the TPV provider in a new cycle. Those masked values from the trustworthy part are used to establish a baseline, with any differences from this baseline added to the list QSS. We can then exactly locate the compromised sub-components from QSS. Hence Property 3 is proved.

7 Performance Analysis

This section analyzes the computation overhead and proof data size of our SIV approach. We compare it with two other common integrity verification approaches, denoted as A1 and A2. A1 is from Ref. [6], which uses raw measurements, while A2 is from Ref. [4], which uses original reports. The symbols are the same as used above. There are $n$ nodes in a cloud, correspondingly there are $n$ constituent parts in a cloud component. Each constituent part has $q$ subcomponents. There are $p - 1$ malicious nodes in the cloud; the proportion $(p - 1)/n$ is less than 0.5.

7.1 Computation

The extra computations added by the SIV approach stem from three processes: proof organization, transformation, and integrity judgement. We define the following notations for the operations: Hash denotes one hash operation, $SEnc$ denotes one symmetric encryption, $ASEnc$ denotes one asymmetric encryption, $Sig$ denotes one signature operation, $X$ denotes one XOR operation, $SUM$ denotes one addition, $SUB$ denotes one subtraction, and $DIV$ denotes one division.

The organization and transformation of proofs are executed on each cloud node individually. Taking a constituent part with $q$ measurements, a Merkle tree is constructed with overhead of $n(q + 2q - 1)Hash$. For a component with $p - 1$ compromised parts, the proof transformation is executed in two rounds. In Round 1 only a Merkle root is transformed, with computation overhead of $SEnc + ASEnc + Sig$. In Round 2 all Merkle tree leaves are transformed, with overhead of $(p - 1)(SEnc + ASEnc + Sig)$. The total computation for transformation is then $(n + p - 1)(SEnc + ASEnc + Sig)$. If we produce a signature with an asymmetric algorithm, the computation will be $(n + p - 1)(SEnc + 2ASEnc)$. The integrity judgement of the TPV includes two phases; in Phase 1 the computation is $n(nX + SUM + DIV + SUB)$, and in Phase 2 it is $(p - 1)(qX)$. The total computation is then $n(nX + SUM + DIV + SUB) + (p - 1)(qX)$.

In the A1 approach all measurements are transmitted to the TPV provider. A common means of security protection is to negotiate a symmetric key for encryption, in which case the computation would be denoted as $n(qSEnc + 2ASEnc)$. The values are inspected individually against the prepared baselines. The integrity verification computation is $n(qSUB)$.

In the A2 approach the verification follows a similar method and the differences are in the size of proof data, which does not affect the computation. A comparison of computation overhead and complexity of A1, A2, and SIV is shown in Table 1.

From Table 1 we see that the complexity of SIV is $O(n^2)$, and that of both A1 and A2 are $O(nq)$. The SIV approach has a higher computation overhead and complexity, but the reduction of proof data size balances the time overhead. Moreover, the cloud bears part of the proof processing, which is easy for it to handle considering the high levels of cloud computing power. This considered, and in view of the urgent need to preserve privacy, the increased computation is acceptable.

7.2 Proof data size

The SIV approach makes primary use of masked proofs. We define the following notations for illustration. The hash function used to generate the raw measurements of subcomponents is denoted as $HASH_1$, and the hash value size is $l_1$ (Bytes). The hash function used to generate the Merkle tree is denoted as $HASH_2$, and the hash value size is $l_2$ (Bytes). It follows that in the integrity judgement process, the size of a masked proof for a constituent part in Phase 1 is $l_1$, and that of a masked proof for a compromised part is $ql_1$. For a cloud the proof data size is $n l_2 + (p - 1) q l_1$. When $l_1 = l_2$, the size is $(n + pq - q) l_1$.

Table 1 Comparison of computation.

| SIV | Cloud | TPV |
|-----|-------|-----|
|     | Computation overhead | Complexity |
|     | Proof organization     | Proof transformation |
| A1 and A2 | $n(3q - 1)Hash$         | $(n + p - 1)(SEnc + 2ASEnc)$ |
|     |                          | $n(nX + SUM + DIV + SUB) + (p - 1)(qX)$ |
|     | Integrity judgement     | $O(n^2)$        |
|     |                          | $O(nq)$         |
In A1 the proofs are raw measurements, and the data size is $nql_1$. In A2 the proofs are original reports; if the size of a report is denoted as $c$ (KB), then the proof data size is $nqc$.

A comparison of the proof data sizes of A1, A2, and SIV is shown in Table 2.

The unit of hash values is generally Byte, whereas reports is KB due to the presence of natural language. Since the difference is of several orders of magnitude, proof data size is reduced greatly, even though SIV adds computations to both the cloud and the TPV provision. This means that the overall performance is balanced out, or even improved, as discussed in the following section.

8 Experiments

We first describe the experimental setup used to evaluate the SIV approach, and then present the results of our effectiveness and performance evaluations.

8.1 Setup

We test the SIV approach on three datasets: D1, D2, and D3. D1 is a small set of raw measurements collected from a private cloud. It has nine subsets (S1 – S9). The cloud is built on multiple VMs, which are used as nodes and host the constituent parts of a component. The cloud runs on three computers, the hardware and software details of which are shown in Table 3. Limited by physical resources, the number of VMs ranges in [6, 30] with a step of six. D2 is a large set of raw measurements. It simulates clouds with different numbers of total and malicious nodes, controlled artificially for research purposes. It has twelve subsets (S10 – S21). D3 is a set of cloud components with different numbers of subcomponents. It has one subset S22 with a component P1 and 23 simulative elements. The number of subcomponents numbers is changed artificially for fine-grained research. D1 and D2 are presented in Table 4, and D3 is presented in Table 5.

The three computers shown in Table 1 are also used as cloud nodes and TPV providers alternately to perform the tests. The VMs are suspended after the generation of measurements, so as to reduce the impact of SIV evaluation.

We choose a common communication tool, OpenSSL 0.9.8gz[27], as the component to be verified in our experiments, because it is one of the most popular software applications in cloud setups. The hash function used in SIV is SHA256; the symmetric algorithm is AES128, and the asymmetric algorithm is RSA. The programming language is Python 2.7.13[28]. The experimental results follow.

8.2 Effectiveness

To evaluate the effectiveness of SIV we focus on whether it can accurately raise an integrity verification result and locate the compromised subcomponents. The tests are performed on computer C1 with subsets S1 – S9. The first five tests are to demonstrate how SIV works, while the last four are to test the accuracy. The experimental results for integrity verification are shown in Fig. 4.

In Fig. 4 we use an arrow with two properties, direction and color, to represent the values in a result matrix $RM$. Figures 4a – 4e demonstrate the appearance of the structural feature in the first five subsets with all trustworthy nodes. When all of the constituent parts making up OpenSSL are integral, their masked proofs will satisfy the structural feature, and the arrows corresponding to the values in $RM$ will have the same direction and color. Figures 4f – 4j show violations of malicious nodes. Taking Fig. 4f as an example, there are two malicious and 28 trustworthy nodes in the cloud. We therefore see that there are two arrows with a different direction and color from the others, meaning that the structural feature has been violated. The abnormal arrows accurately specify the compromised nodes. In Figs. 4f – 4h the compromised constituent parts are tampered with in the same way, thus the direction and color of the abnormal arrows

| Table 2 | Comparison of proof data size. |
|---------|-------------------------------|
|         | Proof data size | Unit |
| SIV     | $(n + pq - q)/l_1$         | Byte |
| A1      | $nql_1$            | Byte |
| A2      | $nqc$              | KB   |

| Table 3 | Hardware and software information of three computers. |
|---------|------------------------------------------------------|
| Computer | Hardware and software information                     |
| C1      | CPU Intel(R) Pentium G3250@3.20GHz, 4 GB RAM, OS OSX 10.9.5. |
| C2      | CPU Intel(R) XeonE5-2603v4@1.70GHz, 64 GB RAM, OS Ubuntu 16.04.2 LTS. |
| C3      | CPU Intel(R) Core(TM) i7-6700HQ @2.60GHz, 16 GB, OS Windows 10. |
Table 4  Comparison of computation.

| Dataset | Subset | N  | MP  | Description                                                                 |
|---------|--------|----|-----|----------------------------------------------------------------------------|
|         |        |    |     | D1                                                                                       |
|         | S1     | 6  |     | All nodes are honest. This is to demonstrate how SIV works.                           |
|         | S2     | 12 |     |                                                                                            |
|         | S3     | 18 | 0   | There are malicious nodes in private cloud. The proportions are 1/15, 1/5, 2/5, and 2/3. This is to test the accuracy of SIV. |
|         | S4     | 24 |     |                                                                                            |
|         | S5     | 30 |     |                                                                                            |
|         | S6     | 30 | 1/15|                                                                                            |
|         | S7     | 30 | 1/5 |                                                                                            |
|         | S8     | 30 | 2/5 |                                                                                            |
|         | S9     | 30 | 2/3 |                                                                                            |
|         | S10    | 100| 0   | All nodes are honest. This is to test the performance of SIV on different sizes of proofs. |
|         | S11    | 1000|   |                                                                                            |
|         | S12    | 5000|   |                                                                                            |
|         | S13    | 10,000| |                                                                                            |
|         | S14    | 1000| 0.1| There are malicious nodes. This is to test the performance of SIV on different malicious nodes proportions. |
|         | S15    | 1000| 0.2|                                                                                            |
|         | S16    | 1000| 0.3|                                                                                            |
|         | S17    | 1000| 0.4|                                                                                            |
|         | S18    | 1000| 0.1| There are malicious nodes. This is to test the performance of SIV on both different sizes of proofs and malicious nodes proportions. |
|         | S19    | 3000| 0.2|                                                                                            |
|         | S20    | 5000| 0.3|                                                                                            |
|         | S21    | 7000| 0.4|                                                                                            |
|         |        |    |     | D2                                                                                       |
|         | S10    | 100| 0   |                                                                                            |
|         | S11    | 1000|   |                                                                                            |
|         | S12    | 5000|   |                                                                                            |
|         | S13    | 10,000| |                                                                                            |
|         | S14    | 1000| 0.1|                                                                                            |
|         | S15    | 1000| 0.2|                                                                                            |
|         | S16    | 1000| 0.3|                                                                                            |
|         | S17    | 1000| 0.4|                                                                                            |
|         | S18    | 1000| 0.1|                                                                                            |
|         | S19    | 3000| 0.2|                                                                                            |
|         | S20    | 5000| 0.3|                                                                                            |
|         | S21    | 7000| 0.4|                                                                                            |

Table 5  Information of dataset 3.

| Dataset | Subset | Cloud component | SN   | Description                                                                 |
|---------|--------|-----------------|------|----------------------------------------------------------------------------|
| D3      | S22    | P1: OpenSSL0.9.8zg | 6    | The P1 is on windows system.                                               |
|         |        | X0              | 10   |                                                                            |
|         |        | X1              | 30   |                                                                            |
|         |        | X2              | 70   |                                                                            |
|         |        | X3              | 120  |                                                                            |
|         |        | X4–X22          | 200–2000| The SNs are changing in range [200, 2000], the step is 100.                |

are alike. In Fig. 4i, there are two attackers and the compromised constituent parts are tampered with in two different ways, so the direction and color of the abnormal arrows are different (shown as red and grey). In these cases where there is a minority of malicious nodes, our SIV approach can accurately raise an integrity verification result. However, the threat model is invalidated and the structural feature fails when the malicious nodes are in the majority, as shown in Fig. 4g where two-thirds of the nodes are malicious. The assumption that the majority of arrows represent trustworthy nodes does not hold in such cases. When all the constituent parts are tampered with in the same way, the masked proofs from the malicious nodes will form their own structural feature, confusing our SIV into establishing an erroneous baseline and leading to inaccurate integrity judgement results.

After the preliminary verification in Phase 1, Phase 2 works to specify exactly which subcomponents are compromised. In essence it is same as the classic determinations with raw measurements, but makes use of the masked proofs.

In the entire integrity verification process, none of the raw measurements have left their original places. This minimizes the attack surfaces of proofs, and eliminates the risk of leaking of private data at the source. At the same time, the accuracy of verification is guaranteed. Therefore SIV is effective and offers enhanced privacy.

8.3 Performance

To evaluate performance we focus on the time overhead of the three processes of the SIV approach. Performing tests on proof subsets S10–S22, we examine how different parameter values (number of subcomponents, key length, hash functions, number of nodes, and proportion of malicious nodes) effect the time overhead.
The tests are conducted on the three computers described in Table 3 as C1, C2, and C3, with the results represented on histograms in red, green, and blue, respectively. The unit of computation time is microseconds.

The first test studies the time overhead involved in executing a proof organization process. The major time-consuming procedure therein is computing the Merkle root value for all raw measurements of subcomponents. There are two possible influencing...
factors: the computational ability of the cloud nodes and the Subcomponents Number (SN). The process is executed on each node individually, with the three computers C1, C2, and C3 working as nodes. The test is performed on subset S22. All of the hash functions used in this process are SHA-256. The experimental results are shown in Fig. 5.

The time overhead is inversely proportional to the computational ability of the cloud nodes. From the hardware information we know that C1 is the most powerful computer and C2 is the least. Accordingly, from Fig. 5 we observe that in most cases (X7 – X17, and X19), C1 takes the shortest time and C2 the longest. But in the cases of P1 – X6, X18, and X20 – X22, C1 or C3 takes the longest, because although we have tried to minimize concurrent tasks on nodes, they cannot be eliminated entirely and thus can still effect the time overhead. The first influencing factor is thus proved with close relevancy.

The time overhead is proportional to the number of subcomponents SN, but the growth rate is relatively low. From Fig. 5 we can observe that when SN changes from 6 to 2000, the execution time on C1 increases from 0.46 ms to 10.14 ms. This represents a small maximum difference of 9.68 ms. We can thus conclude that although the SN does have an effect on the time overhead of proof organization, it makes only a small impact. The second influencing factor is thus proved with small relevancy.

The second test studies the time overhead involved in executing a proof transformation process. The major time-consuming procedures are the signature and symmetric encryption. There are three possible influencing factors: the computational ability of the cloud nodes, the hash types used in the signature, and the key length used in asymmetric encryption. The process is executed on each cloud node individually, with the three computers C1, C2, and C3 worked as cloud nodes. We use five types of hash functions for the signature: MD5, SHA-1, SHA-256, SHA-384, and SHA-512, and three types of key length for asymmetric encryptions: 1024 bits, 2048 bits, and 4096 bits. The asymmetric algorithm is RSA. We encrypt the Merkle root value with a symmetric AES128 algorithm; the key is randomly generated each time and the length is 16 bits. The experimental results are shown in Fig. 6.

The time overhead is inversely proportional to the computational ability of the cloud nodes. From Figs.
6b and 6c we can observe that C1 takes the shortest time and C2 takes the longest. The relationship is not entirely consistent, as seen in Fig. 6a where C1 takes longer than C3 to perform some of the hash functions for the signature. Time overhead is generally not highly relevant to the hash types used for the signature. From Fig. 6b we observe that the time overhead of the five types of hash functions is relatively stable when the public keys used in the RSA algorithm have a same length of 1024 bits. Although the fluctuations on C1 are obvious, this only represents a small maximum difference in value of 3.62 ms. The second influencing factor is thus proved with little relevancy.

The time overhead is proportional to the key length used in asymmetric encryption. From Fig. 6 we observe that when the hash type is MD5 and the node is C1, the durations are 8.92 ms, 31.51 ms, and 208.69 ms for key lengths 1024 bits, 2048 bits, and 4096 bits, respectively. The third influencing factor is thus proved with close relevancy.

The third test studies the time overhead involved in executing an integrity judgment process. The major time-consuming procedure is the calculation of masked proofs. The process is executed on a TPV provider. There are three possible influencing factors: the computational ability of the TPV provider, the number of nodes in the cloud (N), and the proportion of malicious nodes (MP). The three computers C1, C2, and C3 are worked as TPV providers. The tests are performed on proof subsets S10 – S21. We control the values of N and MP artificially to make comparisons. The experimental results are shown in Fig. 7. The time overhead is inversely proportional to the computation ability of cloud nodes. From Fig. 7 we observe little interference in this group of tests. The first factor is again proved with close relevancy to time.

The time overhead is proportional to the number of nodes N. From Fig. 7a we observe that on C1, when N is 100, 1000, 5000, and 10000, with a zero MP, the durations are 2.38 ms, 234.92 ms, 5575.92 ms, and 22334.79 ms, respectively. The time overhead shows large increases by four orders of magnitude. The second factor is thus proved with close relevancy.

The time overhead is not highly relevant to the proportion of malicious nodes MP. From Fig. 7b we observe that on C1, when MP is 0.1 ms, 0.2 ms, 0.3 ms, and 0.4 ms, with a fixed N of 1000, the durations are 221.89 ms, 229.51 ms, 227.48 ms, and 227.61 ms, respectively. There is a small maximum difference in value of 7.62 ms. The third influencing factor is thus proved with little relevancy. Figure 7c confirms the relevancy of N and MP; when both N and MP are increasing, the variation trend of time overhead is similar to Fig. 7a.

By observing the execution time of the three processes, we learn that the third is the longest of the three processes in the SIV approach. The time overheads of the first two are in the tens or hundreds of microseconds, but that of the third is generally in the hundreds or thousands of microseconds. Thus when the cloud component and key functions in SIV have been determined, the number of nodes will be the predominant time factor.

8.4 Comparison
From the performance analysis presented in Section 7 we know that the SIV approach introduces additional computational overhead to enhance privacy. But it also features reductions in proof data size, which might balance out the performance hit. To study whether there are net performance improvements, we perform a quantitative comparison between SIV and two common approaches (A1 and A2) for the aforementioned cloud component[29]. In A1 the proofs are raw measurements. We use an SHA-256 hash function to generate these, thus the size of a raw proof is 256 bits (32 Bytes). In A2 the proofs are original reports. We use a set of reports generated by a popular

![Fig. 7 Time overhead of integrity judgement.](image-url)
integrity analysis software Tripwire 2.4[30], and in our configuration the size of each proof is 2.96 KB. For a cloud component with 30 subcomponents and 1000 nodes, the proof data sizes of the three approaches are compared in Table 6.

From Table 6 we observe that the proof data size in SIV is much smaller than A1 and A2. This is a result of the proofs being organized and transformed on cloud nodes.

We also reconstruct the integrity verification process of A1 and A2 to compare the time overhead of the three approaches. Along with SIV, they are executed on cloud components with 6, 30, and 100 subcomponents involving 1000 and 5000 nodes to measure the overall time overhead. The computing node is C1. The time overheads are listed in Table 7.

From Table 7 we observe that the time overhead of SIV is moderate. For a specific cloud component it takes more time than A1 but less than A2. For a cloud component with 300 subcomponents and 5000 nodes, SIV can produce the verification result in about 13 seconds. Compared to A1 the time latency is about 10 seconds, which remains in an acceptable range. Here we only count the computation time, ignoring communication time. In practice, when the communication time is taken into account, the SIV approach will achieve comparatively greater performance, benefiting from the smaller proof data size. SIV eliminates the leaking of private data at the source by transferring only masked proofs, instead of raw proofs, out of the cloud. Therefore taking into consideration the reduction in proof size, the moderate increment in time overhead and its ability to preserve privacy, SIV outperforms the two common approaches.

### Table 6 Proof data size of three approaches.

|        | Initial generation | Transmission |
|--------|--------------------|--------------|
| SIV    | 32 Byte×30×1000 = 960 KB | 32 KB       |
| A1     | 32 Byte×30×1000 = 960 KB | 960 KB      |
| A2     | 2.96 KB×30×1000 = 88.8 MB | 88.8 MB     |

### Table 7 Time overhead of three approaches.

|        | Time overhead (ms) |
|--------|--------------------|
|        | SIV | A1 | A2 |
| 30     | 1069.240 | 58.449 | 2092.191 |
| 1000   | 1218.220 | 185.225 | 6902.697 |
| 300    | 1921.794 | 557.009 | 20 955.167 |
| 30     | 5774.254 | 314.676 | 10 073.417 |
| 5000   | 6863.079 | 961.278 | 35 794.461 |
| 300    | 12 633.987 | 2823.496 | 106 213.788 |

### 9 Conclusion

The problems of proof expression and protection have been well studied in existing integrity verification approaches. But the private data leakage risks and degradation in performance involved with the verification of a cloud component have not drawn sufficient attention. To address this issue, we first raise the challenges involved in verifying the integrity of cloud components, and present our system model and threat model. We then describe the cloud components and its structural feature. On this basis we present an SIV approach, which includes three processes: proof organization, proof transformation, and integrity judgement. By organizing all of the raw measurements of subcomponents on a node using a Merkle tree method, a root value is computed to represent the comprehensive integrity of a constituent part of a cloud component. This Merkle root is masked in the subsequent proof transformation process. Collision analysis is prevented with a one-time symmetric encryption, and the masked proofs are transferred securely with a TPV public key-based asymmetric encryption. Since the size of the Merkle root is small and constant, the asymmetric encryption time is within a sensible range. From a set of masked proofs from the cloud, a structural feature can be extracted.

The feature is validated in the first phase of the integrity judgement process to raise a verification result. A one-dimensional result matrix \( RM \) is the output. If the cloud component is integral, the \( RM \) could be used as a proof; if it is not, the compromised constituent parts can be accurately located within the \( RM \). In the second phase of integrity judgement a further point of verification can be initiated to precisely locate the compromised subcomponents in a compromised constituent part.

We evaluated the effectiveness and performance of SIV on three datasets, and the experimental results show that the approach has merit. Throughout the process, the raw proofs never leave their original places; only one-time masked proofs are transferred. This minimizes the proof attack surfaces, and eliminates the risk of leaking private data at the source. Although some additional computations are involved, there is a major reduction of proof data size, which balances out the performance hit. SIV thus achieves the goal of verifying the integrity of cloud components with enhanced privacy and good performance.

There are some limitations to our work. SIV makes a necessary assumption that the proportion of malicious
nodes must be less than 0.5. When this assumption fails, SIV is invalid. In addition, the structural feature only exists in cloud components that feature identical constituent parts on separate nodes. When the parts are different, SIV is invalid. In future work, we hope to find solutions to these limitations. We see our work as a first step, paving the way towards protecting cloud privacy in integrity verification. We expect such a work can allow for more meaningful security analysis of existing techniques and will inspire new research in the area.

Acknowledgment
The author would like to thank the anonymous reviewers for their constructive comments and suggestions. This work was supported by the National Key Basic Research and Development (973) Program of China (No. 2014CB340600), Wuhan FRONTIER Program of Application Foundation (No. 2018010401011295), the National Natural Science Foundation of China (No. 61802239), the Fundamental Research Funds for the Central Universities (No. GK201803061), and China Postdoctoral Science Foundation (No. 2018M631121).

References
[1] S. S. Rizvi, T. A. Bolish, and J. R. Pfeffer, Security evaluation of cloud service providers using third party auditors, in Proc. 10th Int. Internet of Things Conf., Exeter, UK, 2017, pp. 1–6.
[2] S. Khan, A. Gani, A. A. Wahab, M. A. Bagiwa, M. Shiraz, S. U. Khan, R. Buyya, and A. Y. Zomaya, Cloud log forensics: Foundations, state of the art, and future directions, ACM Computing Surveys (CSUR), vol. 49, no. 1, pp. 1–42, 2016.
[3] S. Saibharath and G. Geethakumari, Cloud forensics: Evidence collection and preliminary analysis, in Proc. of 2015 IEEE Int. Advance Computing Conf., Bangalore, India, 2015, pp. 464–467.
[4] L. Tan, J. Chen, and M. T. Zhou, Trustworthiness evidence collection mechanism of running dynamic environment of trusted terminal, (in Chinese), Acta Electronica Sinica, vol. 41, no. 1, pp. 77–85, 2013.
[5] L. Chen, R. Landfermann, H. Lohr, M. Rohe, A. Sadeghi, and C. Stuble, A protocol for property-based attestation, in Proc. 1st ACM Workshop on Scalable Trusted Computing, Fairfax, Virginia, USA, 2006, pp. 7–16.
[6] R. Sailer, X. Zhang, T. Jäger, and L. V. Doorn, Design and implementation of a TCG-based integrity measurement architecture, in Proc. 13th Int. Usenix Security Symposium, San Diego, CA, USA, 2004.
[7] V. Varadharajan and U. Tupakula, Counteracting security attacks in virtual machines in the cloud using property based attestation, Journal of Network & Computer Applications, vol. 40, no. 1, pp. 31–45, 2014.
[8] Y. Han, T. Alpcan, J. Chan, and C. Leckie, Security games for virtual machine allocation in cloud computing, in Proc. 4th Int. Decision and Game Theory for Security Conf., Fort Worth, TX, USA, 2013, pp. 99–118.
[9] A. Singh and K. Chatterjee, Cloud security issues and challenges: A survey, Journal of Network and Computer Applications, vol. 79, pp. 88–115, 2017.
[10] M. R. Watson, N. H. Shirazi, A. K. Mamerides, A. Maueth, and D. Hutchison, Malware detection in cloud computing infrastructures, IEEE Transactions on Dependable and Secure Computing, vol. 13, no. 2, pp. 192–205, 2016.
[11] T. Zhang and R. B. Lee, Monitoring and attestation of virtual machine security health in cloud computing, IEEE Micro, vol. 36, no. 5, pp. 28–37, 2016.
[12] Z. H. Ning, W. Jiang, J. Zhan, and P. Liang, Property-based anonymous attestation in trusted cloud computing, Journal of Electrical and Computer Engineering, vol. 17, pp. 1–7, 2014.
[13] G. Proudler, L. Chen, and C. Dalton, Trusted Computing Platforms: TPM 2.0 in Context, Springer, 2015.
[14] V. Costan and S. Devadas, Intel SGX explained, Cryptology ePrint Archive, p.86, 2016.
[15] T. Alves and D. felton, Trustzone: Integrated hardware and software security, ARM, White paper, vol. 3, no. 4, pp. 18–24, 2004.
[16] R. Perez, R. Sailer, and L. V. Doorn, vTPM: Virtualizing the trusted platform module, in Proc. 15th USENIX Security Symposium, Vancouver, Canada, 2006, pp. 305–320.
[17] C. Chen, H. Raj, S. Saroiu, and A. Wolman, cTPM: A cloud TPM for cross-device trusted applications, in Proc. 11th USENIX Symposium on Networked Systems Design and Implementation, Seattle, WA, USA, 2014, pp. 187–201.
[18] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich, VC3: Trustworthy data analytics in the cloud using SGX, in Proc. 36th IEEE Symposium on Security and Privacy, San Jose, CA, USA, 2015, pp. 38–54.
[19] T. Brito, N. O. Duarte, and N. Santos, ARM TrustZone for secure image processing on the cloud, in Proc. 35th Symposium on Reliable Distributed Systems, Budapest, Hungary, 2016, pp. 37–42.
[20] A. H. Aljammal, H. Bani-Salameh, A. Alsarhan, M. Khababsheh, and M. Obiedat, Node verification to join the cloud environment using third party verification server, International Journal of Interactive Mobile Technologies (iJIM), vol. 11, no. 4, pp. 55–65, 2017.
[21] T. S. Khatri and G. B. Jethava, Improving dynamic data integrity verification in cloud computing, in Proc. 4th Int. Computing, Communications and Networking Technologies Conf., Tiruchengode, India, 2013, pp. 1–6.
[22] M. Yi, J. Wei, and L. Song, Efficient integrity verification of replicated data in cloud computing system, Computers and Security, vol. 65, pp. 202–212, 2017.
[23] B. Imene and H. Salima, Verifiable outsourced computation integrity in cloud-assisted big data processing, in Proc. 13th Symposium on Programming and Systems (ISPS), 2018, pp. 1–6.
[24] I. Eyal and E. G. Sirer, Majority is not enough: Bitcoin mining is vulnerable, in Proc. Int. Financial Cryptography and Data Security, Berlin, Germany, 2014, pp. 436–454.

[25] M. Jakobsson, T. Leighton, S. Micali, and M. Szydlo, Fractal Merkle tree representation and traversal, in Proc. Int. Cryptographers’ Track at the RSA Conf., Berlin, Germany, 2003, pp. 314–326.

[26] The installation and usage of OpenSSL on windows system, https://my.oschina.net/vazor/blog/95488, 2012.

[27] OpenSSL, https://www.openssl.org/, 2018.

[28] Python, https://www.python.org/, 2018.

[29] Y. Wang, B. Jin, and J. Dong, Security log with integrity verification support, (in Chinese), Journal of Tsinghua University, vol. 56, no. 3, pp. 237–245, 2016.

[30] Tripwire, https://www.ibm.com/developerworks/cn/aix/library/au-usingtripwire, 2012.

Bo Zhao received the BEng degree from Wuhan Technical University in 1994, and the PhD degree in computer software and theory from Wuhan University in 2006. He is currently the vice-president of the School of Cyber Science and Engineering, Wuhan University. His research interests include information system security, trusted computing, embedded system, and cloud computing security.

Peiru Fan received the BEng degree from Shanxi University in 2011, and the master degree in signal and information processing from Tianjin polytechnic University in 2014. She is currently a PhD candidate in Wuhan University. Her current interests include virtualization, cloud computing, and security.

Pengyuan Zhao received the BEng degree from Hebei University in 2001, and the master degree in computer network from Hebei University in 2010. He is currently a PhD candidate in Wuhan University. His research interests include big data, embedded system, and IoT security.

Mingtao Ni received the BEng degree from Wuhan University of Technology in 1998, and the master degree in software engineering from the University of Electronic Science and Technology in 2008. He is currently a PhD candidate in Wuhan University. His research interests include embedded system, IoT security, and mobile device security.

Jinhui Liu is currently a lecturer at the School of Computer Science, Shaanxi Normal University. She graduated from Wuhan University in 2017 with a PhD degree. Her research interests include post quantum cryptography and digital signature.