RepCloud: Attesting to Cloud Service Dependency

Anbang Ruan and Andrew Martin

Abstract—Security enhancements to the emerging IaaS (Infrastructure as a Service) cloud computing systems have become the focus of much research, but little of this targets the underlying infrastructure. Trusted cloud systems are proposed to integrate trusted computing infrastructure with cloud systems. With remote attestations, cloud customers are able to determine the genuine behaviors of their applications' hosts; and therefore they establish trust to the cloud. However, the current trusted clouds have difficulties in effectively attesting to the cloud service dependency for customers' applications, due to the cloud's complexity, heterogeneity and dynamism. In this paper, we present RepCloud, a decentralized cloud trust management framework, inspired by the reputation systems from the research in peer-to-peer systems. With RepCloud, cloud customers are able to determine the properties of the exact nodes that may affect the genuine functionalities of their applications, without obtaining much internal information of the cloud. Experiments showed that besides achieving fine-grained cloud service dependency attestation, RepCloud incurred lower trust management overhead than the existing trusted cloud systems.

Index Terms—Security and privacy protection, security kernels

1 Introduction

Substantial security threats to the rapidly-growing IaaS (Infrastructure as a Service) cloud computing service model are attracting a great deal of attention. Massive data loss has drawn cloud users to pay serious attention to the confidentiality and integrity of their data and computation when these are outsourced to a cloud provider. Various cloud security mechanisms have been proposed to protect users' benefits with different perspectives. Authorization and authentication mechanisms protect users' assets in the cloud from unauthorized accesses. Auditing modules record actions performed in the cloud for evaluation at a future point in time. IDS and firewall protect internal cloud resource from attacks, and VMI (virtual machine introspection) interfaces [1] enable fine-grained inner-VM (virtual machine) protections from security services provided by the cloud, e.g., antivirus and rootkit detector. A full line of work for secure data outsourcing has also been proposed. Furthermore, Cloud Security Alliance [2] defines a comprehensive cloud security reference model, with 12 security domains covering different level of security mechanisms in the cloud.

However, most of these cloud security mechanisms still rely on the integrity of their lower layer software dependency. Therefore, their genuine enforcements are still under questions. For example, in the virtualization environment, security modules, such as audit or access control, may be easily circumvented when their underlying virtualization layer has been tampered with [1]. Moreover, in the cloud computing paradigm, the infrastructure no longer belongs to the cloud customers. It is hence necessary for the cloud providers to generate evidence for their customers to testify that the advertised services and security mechanisms are effectively enforced. A critical step to achieve this is to prove the trustworthiness of the cloud service dependency for the cloud applications. As a party cannot simply claim its own trustworthiness, a trusted third party (TTP) should be introduced for generating the evidence.

The TCG (Trusted Computing Group) [3] proposes to attach a *platform* (i.e., physical machine) with a TPM (trusted platform module) [4]. The tamper-proof nature of the TPM and its cryptographic protocols empower it to act as a TTP dedicated to this platform. TPM genuinely records the platform's configuration. *Remote Attestations* are further implemented to allow platform users to reliably interrogate the TPM and determine the configuration. The returned *attestation tickets* serve as the trust evidence to vouch for the integrity of the configuration. This configuration usually includes the detailed list of the genuine identity of all the platform's loaded software, which in turn testifies the platform's *properties* (or *behaviors*).

Accordingly, several authors have proposed to integrate cloud systems with the trusted computing infrastructure [5], [6]. Cloud *nodes* (i.e., physical servers) are equipped with built-in TPMs. Centralized *attestation delegate* is implemented to gather the TPM-generated trust evidence from each node to attest to the properties of its hosted services. Currently, these schemes mostly assume homogeneous property distributions among the cloud nodes [7]. For example, whether the entire cloud management nodes enforce ubiquitous non-discriminative VM scheduling policies, or whether all the compute nodes have the capabilities to enforce strong VM-isolation. Besides, they require customers to rely on this central delegate to perform the attestations to the cloud internals [5], [6]. Customers then attest

• *The authors are with the Department of Computer Science, University of Oxford, Oxford OX1 3QD, United Kingdom.*
E-mail: {anbang.ruan, rew.martin}@cs.ox.ac.uk.

Manuscript received 31 May 2015; revised 23 Mar. 2016; accepted 6 Apr. 2016. Date of publication 3 May 2016; date of current version 6 Oct. 2017.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSC.2016.2558513

to this delegate to make sure that the attestation services are genuine. The trustworthiness of the delegate further implies that their assets are secure, and the services they have purchased are trustworthy.

However, we observe that the cloud's nature of *complexity*, *heterogeneity*, and *dynamics* will hamper the effectiveness of these remote attestation schemes, especially when considering the attestations to the cloud service dependency of large-scale cloud applications.

Complexity. Typical IaaS cloud systems have a large quantity of nodes, but only a small fraction serves a customer's cloud application. As we will discuss in Section 4, this set of nodes (i.e., the *dependency* of the cloud application) usually contain not only the VMs' hosts, but also various infrastructural facilities, e.g., the management nodes or the security service modules. Understanding and attesting to these detailed dependency for every cloud application requires tremendous efforts. Some trusted cloud approaches only allow customers to attest to the properties of their VMs' hosts. However, a VM's behaviors depend on not only its host, but also the supporting facilities in a cloud, e.g., the VM's Storage or Networking capabilities are usually implemented on separated nodes [8]. Other approaches hide this dependency complexity by locating the entire application in a separated zone, and vouching for the trustworthiness of all the services in the entire zone. For example, the *nova-verify* service in [9] attests to the properties of an entire Openstack scheduling domain. However, the evidence for this trustworthiness is usually too coarse-grained to provide useful information.

Heterogeneity. To satisfy different security requirements, various security modules are implemented inside different parts of a cloud. Moreover, a same module can also expose different properties in the view of different applications. This diversity therefore introduces the *heterogeneity* in the properties of different cloud applications' cloud dependency, even when they are deployed in a same cloud management domain. For example, the traffic among the VMs of an application are monitored by an internal firewall, while another application uses VMI-based anti-virus module provided by the cloud [10]. However, with the current schemes, it is difficult for customers to determine the exact properties of their application's dependency by merely getting the knowledge of the properties of the entire cloud domain.

Dynamism. Attestation tickets can only indicate the trustworthiness of a platform up to the time when the tickets are generated (Section 3). As the properties of a node are viable to change, attestations should be performed regularly. Moreover, VMs are migrating among different nodes, and these migrations are transparent to customers. Their dependency inside the cloud is therefore dynamic, which intensifies the difficulties of dependency identification and attestation. It is difficult for the existing central attestation delegate to keep track of the dynamism for each cloud application's dependency. It is also infeasible for the cloud to update the detailed locations of the VMs' hosts and all the potential supporting cloud services to customers, in order to allow them to verify their properties. This violates the cloud's flexibility and scalability achieved by the implementation detail hiding.

We observe that the difficulties in effectively managing trust inside a cloud infrastructure come from the opaqueness of the cloud's internal communications to the attesters, e.g., the central attestation delegate. Though a cloud infrastructure is usually organized in a centralized deployment, its internal communication patterns are essentially distributed. Trust relationship among a cloud application's components and the supporting cloud services can be intuitively managed in a decentralized approach, by conforming to the services' interaction semantics. This decentralized trust management can be achieved by integrating the reputation systems [11], which are widely employed for managing and modeling trust for the *peer-to-peer* applications.

In this paper, we aim to build a cloud trust management framework, the *RepCloud*, which will identify and attest to the cloud service dependency for a large-scale IaaS cloud application. RepCloud integrates trusted computing framework with the reputation systems. In RepCloud, *decentralized attestations (DAs)* are performed among nodes inside the cloud, based on their interactions. Attestation tickets gathered by nodes are *disseminated* among logical related nodes to reduce redundant attestations. The attestation relationship represented by the tickets are further *aggregated* to form a global *web-of-trust*. This *web-of-trust* allows the attestation delegate to understand each node's dependencies inside the cloud. Accordingly, by querying and tracing this *web-of-trust*, the central attestation delegate is able to identify the cloud service dependency of a cloud application, and gather necessary tickets to vouch for the properties of all the depending services. In particular, we made the following contributions:

1) *Decentralized attestation scheme.* We integrated the reputation systems in the trusted computing framework. We managed *TCG Trust* (i.e., trust relationship defined with the Trusted Computing Group) in the cloud in a decentralized and autonomous approach, and proposed to aggregate and disseminate the attestation tickets. Our experiments showed that, with reputation systems, the *TCG Trust* is more effectively and efficiently transmitted among interconnecting nodes. Besides, redundant attestations are significantly reduced, while a low level of *properties-change-detection* delay is still maintained.
2) *Hierarchical cloud trusted computing base definitions.* We defined the concept of *Cloud Trusted Computing Base (cTCB)* to illustrate the cloud service dependency for a target cloud application. We defined the *cTCB* in a hierarchical structure, with the scale of the concerning nodes minimized. These definitions help to better illustrate the dependency relationship among cloud components. Moreover, we observed that by monitoring the direct communication among nodes and performing decentralized attestation, the *cTCB* is easier to determine and continuously attest to.
3) *Fine-grained cloud trusted computing base attestations.* We designed a new paradigm for implementing the *cTCB* attestation, taking into consideration the interaction patterns among related nodes. *cTCB* attestations help cloud customers to determine the security properties of the cloud dependency nodes that support or may affect the genuine functionalities of their VMs. It implements an effective and practical way for establishing trust from the customers to the cloud systems and the cloud service providers.

In the following text: Section 3 introduces the background and our motivations. Section 4 revisits a general cloud model, and presents the hierarchical *cTCB* definitions accordingly. In Section 5, we introduce the RepCloud framework for aggregating and disseminating the *TCG Trust*, according to the *cTCB* definitions. Section 6 presents the *cTCB* attestation method built on RepCloud. Our experiments and implementation are presented in Sections 7 and 8 respectively. Section 2 summarizes related work. Finally, we conclude our paper and discuss the future work in Section 9.

2 Related Work

Trusted Virtual Datacenter (TVDc) [12] provides strong isolation between workloads by enforcing a Mandatory Access Control (MAC) policy throughout a datacenter. It also provides integrity guarantees to each workload by leveraging a hardware root of trust in each platform to determine the identity and integrity of every piece of software running on a platform. Trusted cloud computing platform (TCCP) [5] enables users to attest to the IaaS provider and determine whether or not the service is secure before they launch their virtual machines. Cloud verifier (CV) [6] generates integrity proofs for customers to verify the integrity and access control enforcement abilities of the cloud platform that protect the integrity of customer's application VMs in IaaS cloud.

These trusted cloud solutions share the similar centralized structure. They attest the trust state of the entire cloud altogether. As we discussed in Section 1, deficiencies limit their scalability. Therefore, when implementing cloud application attestations, the existing schemes could only provide a coarse-grained results, such as whether the entire cloud infrastructure is enforcing a certain security mechanism. RepCloud, on the other hand, implements a decentralized peer-to-peer attestation structure and exposes the trust state of the cloud taking considerations of the peer attestation relationships. The semantics of communications are preserved, which can enable a fine-grained attestation, with the security concerned nodes been regularly attested to. It therefore enables cloud users to determine the exact properties for each of their VMs inside the cloud.

Santos et al. proposed a new abstraction to let data be sealed and unsealed only by nodes whose configurations match a predefined policy [13]. However, the genuinely enforcements of this architecture still needs to be attested, which is still implemented by the centralized delegated scheme. Abbadi [7] also identifies that the cloud dynamics prohibit the practical trusted cloud implementation. He suggested that the cloud infrastructure is actually homogeneous, and proposed the *combined chain-of-trust*, which attests a cluster of nodes who have the exactly same configuration together. However, we observed that when considering the supporting services as the TCB of the cloud application [14], the homogeneous assumptions are broken. In this case the detailed TCB's properties have to be obtained in order to achieve cloud application attestation. This will be facilitated by the RepCloud framework.

A framework for evaluating the CSPs' trustworthiness by using the trusted computing infrastructure was proposed in [15], [16]. In this work, the CSPs' claims on the SLAs are translated to a set of properties and will be examined by the property-based attestation schemes [17], [18]. *Uncertainties* are also introduced to model subjective trust semantics such as the trustworthiness of the *Certificate Authorities (CAs)* issuing the property certificates. The *dynamic trust model* is designed to integrate these uncertainties when evaluating the properties of a cloud service. This work is a good complement to the RepCloud framework, as RepCloud focuses on identifying the fine-grained attestation target for a cloud customer, and this framework focuses on translating the properties. We will further investigate the integration with this framework. Uncertainties in RepCloud's distributed trust model will be more complicated to model, as RepCloud gathers the evaluations from multiple nodes to form a consensus of a target node's properties. This requires the aggregation of uncertainties towards a target properties. We will also investigate this model in our future work.

The *Service Dependency Discovery* systems [19] determine the service dependency for a large-scale distributed environment by analyzing the timings of network packets' transmission and reception. This helps determining which service hosted on the node is responsible to the interactions with a remote service. Accordingly, by implementing the Constellation methods in each RepVisor, trust dissemination and aggregation can be implemented according to the finer-grained dependency relationship. Therefore, the delegate is able to understand whether a *dynamic dependency* of a node is actually interacting with a VM or a cloud service hosted on the node. We leave the integrations of these schemes in our future work.

3 Background and Motivations

In this section, we first present the basic concepts of the trusted computing. We further introduce important ideas from the reputation systems, which are widely used in the peer-to-peer systems. We finally discuss how reputation systems will improve the scalability of the trusted computing technology. This forms the foundation for the RepCloud framework, introduced in the following sections.

3.1 Trusted Computing

Trusted Computing Group [3] specifies a hardware module, the trusted platform module, to serve as a root of trust for a platform. The attestation architecture designed by the TCG aims at *measuring* all loaded executables by hashing each piece of software into the TPM before loading it. The platform is bootstrapped by the CRTM (the Core Root of Trust for Measurement), which is trusted by default and will initiate measurements to the BIOS and the boot loader to construct a *chain of trust*. This chain is then extended through every operating system components up to the applications and their configuration files. Once the executables are measured into the TPM, the TPM can reliably attest to the hash values by signing them with a TPM protected key, i.e., the *AIK* (Attestation Identity Key). The signed values (i.e., the *attestation tickets*) are sent to the challenger (or verifier), who can then decide whether to trust the target platform. The tickets generated by the trusted computing framework are *deterministic*, i.e., they can reveal the genuine state of the target platform, as they are based on the tamper-proof registers protected by the TPM.

Meanwhile, the tickets are themselves *tamper-proof*, as they are signed by the TPM protected key. However, these tickets are *transient* as they can only reflect the genuine state of the platform up to the time when they are generated. Hence, attestations should be performed regularly for updating the trust states.

As with the limited computation capability of a TPM chip, a complete attestation procedure between two nodes may require several seconds, the latency of which is unacceptable when attestation is performed regularly. Stumpf et al. [20] proposed the Timestamped Hashchain-based Attestation to compensate this deficiency. To avoid performing expensive TPM operations for every attestation request, the server (the node to be attested to) uses a nonce issued by a trusted third party binding to the timestamp of the current time to perform the TPM_Quote instruction regularly and generate an attestation ticket each time. As long as the time have been synchronized, the client can determine the trustworthiness of the server from the ticket and deduce the freshness of the attestation from the nonce. As the nonce is generated from a global time instead of randomly chosen by the client, attestation tickets generated by this scheme are hence *disseminable* and can be reused by a third party, e.g., different clients.

3.2 Reputation System

In many collaborative applications such as P2P systems, trust only represents a "personal" view of a peer, and reputation systems [11], [21] are used for a peer to infer trust towards a stranger by consulting other peers it trusts or by aggregating all the others' views, in assumption that most peers will "tell the truth". By harnessing the community knowledge in the form of feedback, reputation-based trust systems help participants decide whom to trust, encourage trustworthy behavior, and deter dishonest participation by providing a means through which reputation and ultimately trust can be quantified and disseminated. A typical reputation system usually defines following components [11]: 1) *formulation*, the mathematical presentation of the reputation metric, together with the sources of input to that formulation; 2) *calculation*, the algorithm to calculate the formulation for a given set of constraints; 3) *dissemination*, the mechanism for storing and disseminating the reputation value. As this "personal" view is very easy to forge, various attacks to the reputation systems exist, while several countermeasures have also been proposed [11].

3.3 Motivations

As a single attestation ticket can undeniably reveal the genuine trust state of a node, the *TCG Trust* can be effectively and efficiently disseminated in the cloud with reputation systems. Hence, by performing attestations among nodes inside the cloud based on their interactions, and then disseminating the attestation tickets, redundant attestation efforts can be saved. With adaptive specification of re-attest timing, a low average for repeatedly attestations interval can still be achieved. Moreover, from the mutual-attestation relationship among nodes, a global *web-of-trust* can be constructed. We observe that in the web with nodes regularly attesting to each other and exchanging newly generated tickets, corrupted nodes can be quickly identified. With the communication semantics maintained by the web, the dependency of an application can be easily identified. With additional attestations from a third party to this web, malicious collaborators [11] can also be discovered. Hence, by constructing and modeling this web, we can achieve fine-grained cloud TCB attestation with high confidence for trust.

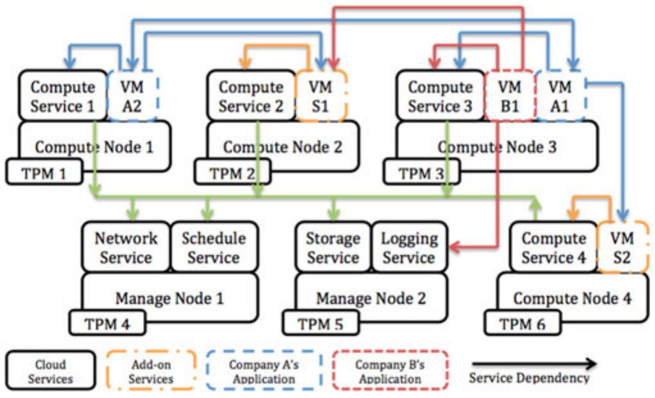

Fig. 1. Trusted cloud expectations.

4 Cloud TCB Definitions

We first visit a general IaaS cloud scenario. In an IaaS cloud, cloud applications are implemented as a set of virtual machines, which run on shared physical machines, under the supervision of the hypervisors, such as XEN or KVM. Supporting facilities are implemented to organize all the virtualized platforms into shared management domains. These facilities constitute the cloud management infrastructure. For example, in the OpenStack cloud infrastructure [8], these facilities usually include: the *Compute* services, which are installed on each virtualization platform to manage hypervisor and the VMs; the *Schedule* services, which schedule VMs among the *Computes*, to balance load or other economical elements; the *Network* or *Storage* services, which control the networking or storage capabilities for the VMs. Besides, cloud customers may also acquire additional cloud services for their specific requirements, e.g., *Logging Services*, or other service VMs deployed by third party providers [14].

In Fig. 1, company A deploys a cloud application, which contains VM_A_1 and VM_A_2. Company B's application only has one VM, VM_B_1. In order to determine the properties of the A's cloud application and the acquired cloud services, A will attest to: 1) the *chain-of-trust* inside its VMs; 2) the *chain-of-trust* rooted from the hardware TPMs of the VMs' hosts to support the virtualization platforms; 3) all the services deployed on their VMs' hosts, in order to organize the hosts in a cloud, e.g., the *Compute*, and the client side services of the *Storage* and *Network*, etc.; 4) all the services deployed independently, but provide crucial services for hosting the cloud, e.g., the *Schedule*, *Compute*, and *Storage*; and 5) all the additional cloud services, deployed to service the target cloud applications' specific needs.

According to [22], the TCB (Trusted Computing Base) of a computer system is *the set of all hardware, firmware, and/or software components that are critical to its security, in the sense that bugs or vulnerabilities occurring inside the TCB might jeopardize the security properties of the entire system*. When regarding a cloud as a large-scale operation system, the

TCB for A's application should include all the components deployed in the cloud to support its functionalities, i.e., components 2)-5). To help effectively identifying all these components, we define the *cloud TCB* at three granularities:

Core TCB of a node. A VM relies on the virtualization layer, including the hypervisor, the management console and other privileged software modules. Components in this layer also have their own dependencies. Accordingly, as discussed in Section 3, all the components forming the chain-of-trust [3] up to the virtualization layer are crucial for implementing a VM. On the other hand, each cloud node implements essential services for constituting a cloud infrastructure. We define these services as the *Core Service* of the node, e.g., the *Compute* service on each VM host, or the centralized *Schedule* service. They also effect the genuine behaviors of a target VM.

Accordingly, we define the *Core TCB of a node* (*coreTCB*) to contain the *Core Services*, along with all the software components in the node's *chain-of-trust* [3] to bootstrap the platform into an expected state to host the VMs and the Core Services. Generally, these include the CRTM, BIOS, the bootloader, Operating System Kernel or hypervisor, and other critical modules, etc. On the other hand, collocated VMs may also affect the integrity of a target VM, e.g., by enforcing side-channel attacks, or exploiting the hypervisor's vulnerabilities. However, attesting to these VMs is both impractical and infeasible. Alternatively, cloud customers may determine the risks from these collocated VMs by examining the properties of the hypervisor and additional supporting services, e.g., whether strong isolation mechanisms are implemented, or violation precautions are enforced, etc.

Cloud TCB of a node. Cloud is a distributed system with a large amount of collaborating nodes. The functionalities of one node usually depend on the behaviors of the others. For example, a *Compute* node relies on the *Schedule* node to assign it with VMs; and the *Schedule* node further needs the *Image Service* node for managing the VM images. We therefore define all those nodes supporting a target node's *Core Services* as its *static dependency*. The static dependency relationship is usually defined by the cloud administrator during the deployment of the cloud infrastructure.

On the other hand, nodes in a cloud interact with each other due to the communications among their upper layer services. We define these nodes as a target node's *neighbors*. For example, VMs on a node may communicate to the VMs on the others, which will result in the communications among their hosting nodes. Moreover, we assume attacks require direct interactions: malicious behaviors in the neighbors may bring risks to the target node. Therefore, the behaviors of a node's *neighbors* may affect the node's functionalities. We define these nodes as the target node's *dynamic dependency*. In addition, *static dependency* nodes also need to communicate with the target to implement their responsibilities. We denote the nodes, serving both as *dynamic* and *static* dependency, as *active static dependency*. This prevents dishonest CSPs (cloud service providers) to deliberately specify service nodes as static dependency, but configure them to not providing services.

We now define the *cloud TCB of a node* ($cTCB_{Node}$) to include the node's own coreTCB, the coreTCBs of all its dynamic dependency node, and the $cTCB_{Node}$ of all its active static dependency node. As static dependency nodes can be considered as the delegate to implement parts of the target node's functionality, their $cTCB_{Node}$ should be considered as part of the target node's $cTCB_{Node}$. This recursive definition will ultimately build an active static dependency tree, containing all the nodes to support the target node's functionality, and all the dynamic dependency nodes which will affect the genuine behaviors of target node and all its active static dependency.

Cloud TCB of an application. An application in an IaaS cloud contains a set of VMs and possibly several supporting cloud services. The *cloud TCB of an Application* ($cTCB_{App}$) is hence comprised of the $cTCB_{Node}$ of all the nodes hosting these VMs or services. On the other hand, as VMs can be migrated and the location of the supporting services can also be changed accordingly, this TCB dynamically changes during the lifecycle of the application. In this paper, we only concern with the attestation to this TCB at a specific time, which has a static set of nodes. By integrating trusted migration protocols [5], attestation to the entire lifecycle can be achieved.

5 RepCloud Framework

According to the definitions in Section 4, the key to effectively attesting to the $cTCB_{App}$ for a cloud application is to identify the $cTCB_{Node}$ for all the cloud nodes hosting and supporting the application's components. This can be achieved by analyzing the communication patterns among the nodes inside the cloud. However, as discussed in Section 1, the *complexity*, *heterogeneity* and *dynamics* of a cloud inhibit the central attestation delegate to effectively understand the cloud's internal activities. In this section, we present the *RepCloud* framework, which preserves the interaction semantics among cloud nodes by enforcing *decentralized attestation*.

The core idea of *DA* is that attestations to cloud nodes are performed on demand by other nodes based on their communication needs. As the communication patterns generally reflect the service dependency relationship, each node possesses the properties of its depending nodes. Moreover, as nodes may have same dependency, each node disseminates the attestation tickets it collects to its logical neighbors. This reduces the redundant attestation efforts. Moreover, a *web-of-trust* is constructed, which allows nodes to react faster to other nodes' property changes. The *web-of-trust* also helps the *attestation delegate* to understand the dependency relationship among cloud nodes. This facilitates effective and fine-grained *cTCB* attestations.

In RepCloud, each node maintains a *Local Trust Vector (LTV)* recording the attestation tickets it gathered by attesting to its communication peers, which we define as its *neighbors*. These tickets are then disseminated and aggregated iteratively among the neighbors to construct the global *web-of-trust*. A partial *web-of-trust* is maintained by each node in its *Global Trust Metric (GTM)*, as a node only concerns the properties of its *neighbors*. We will examine the formulation, calculation and dissemination of trust semantics in RepCloud respectively.

5.1 Notations

A node in the cloud is denoted as N_i. The attestation ticket generated by the attestation from N_i towards N_j (performed by N_i) is denoted as:

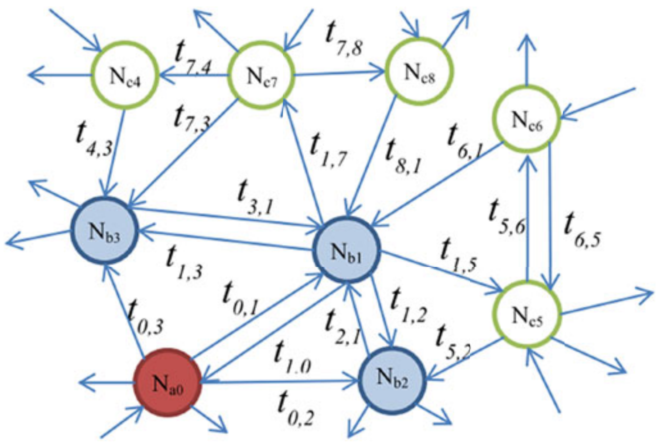

Fig. 2. Simplified *web-of-trust* topology.

$$t_{i,j} = (PCR_{i,j}, ts_{i,j}).$$

It represents the trust N_i places upon N_j. $PCR_{i,j}$ is the PCR value representing the trust state of N_j, and $ts_{i,j}$ is the timestamp for the time the attestation was performed. The *Local Trust Vector* of N_i is hence denoted as:

$$L^i = \vec{t}_i.$$

We further define the *Global Trust Metric* as follows. GTM maintained by N_a is denoted as G^a. In the following text, superscripts are used only when necessary to denote that the data structure is maintained by the particular node,

$$G = [t_{i,j}].$$

As each node only maintains the trust information of its neighbors, only a part of its GTM has valid values. We define $t_{i,j} = (0,0)$ when no attestation is performed by N_i to N_j. The set Nbr^i denotes the neighbors of N_i:

$$Nbr^i = \{N_j \mid \forall j(t_{i,j} \neq 0)\}.$$

We define relationship "$>$" for comparing the freshness of the attestation tickets as below. "max" and "min" are hence self-defined

$$t_{i,j} > t'_{i,j} \Leftrightarrow ts_{i,j} > ts'_{i,j}.$$

Fig. 2 illustrates a simplified *web-of-trust* topology. Directed edges represent attestations, with their endpoints as the target nodes. Attestation tickets are specified as the weights. For better presentation, in the figure, nodes attested to by N_{a_0} are referred to as N_{b_i}. They are the neighbors of N_{a_0}, comprising the set Nbr^{a_0}. Nodes interacting with N_{b_i} but have no connection to N_{a_0} are depicted as N_{c_j}. With the two-level trust aggregation in RepCloud (Section 5.3), all these attestation tickets can be located in the GTM of N_{a_0}: G^{a_0}.

5.2 Local Trust Gathering

A node maintains the trust evidence of its $cTCB_{Node}$ in its LTV by performing attestations to its neighbors periodically. The three typical steps of an attestation procedure are considered for gathering this local trust:

Measurement. Every node in the cloud model has an embedded TPM, which records the measurement of every component in its chain-of-trust. The entire core TCB is measured. Different runtime attacks exist to inject malicious codes directly into the memory of the node without being measured. However, as countermeasures [23] are being proposed and the attestation framework is also evolving to mitigate these threats, we currently assume that all executable loaded on a node have been genuinely measured, and with the measurement values extended to the TPM.

Attestation. Three types of communication actions with different assumptions on trust are defined in our context: 1) *Critical communication*. Security-critical communication actions among nodes require immediate attestations, because previously generated attestation tickets may be obsolete. These actions can be specified by users, and will trigger local attestation whenever they are encountered. For example, a user can define a VM storing confidential database as a critical resource, and require all the communications to this VM to trigger immediate attestations. These communications are therefore regarded as the critical communications. 2) *General communication*. For communication actions that do not require immediate attestation, trust evidence can still be gained as proof for service enforcement, e.g., SLA checking. Attestations to a target node in this case can be performed a longer time ago, or by some other nodes that are trusted (attested) by the current node. 3) *Trusted communication*. Specific communication actions should be performed with preassumed trust. Otherwise they may lead to recursive attestations. These actions should be carefully designed and with limited interfaces, e.g., RepCloud protocol communications.

RepCloud uses the Timestamped Hash-chain attestation protocol [20] to increase the throughput of the attestation operations (Section 8). This protocol also generates transmissible attestation tickets. The time of attestation is inferred from the ticket, as the *nonce* is bound to a global time. Every time when an attestation is performed, the *PCR* value and the timestamp will be stored in the LTV of the current node. Every attestation ticket is valid for a specific length of time, after which new attestation should be performed.

Verification. With local attestation, a node is regarded as trustworthy when the measurements of all components in its *coreTCB* are identified in a global expected measurement list, i.e., the *white-list*. Faulty nodes will be reported. As a node is identified with its AIK, a new AIK can be issued to the node after it is reset. It is hence regarded as a new node in RepCloud, and its previous distributed maintained attestation information will be discarded automatically when obsolete. All VMs hosted on a faulty node will be halted and the users will be informed. With the decentralized trust management structure of RepCloud, distributed white-list management can be implemented.

5.3 Global Trust Aggregation

As a node only concerns the trust state of its neighbors, a local part of the web-of-trust is maintained in its GTM. In RepCloud, a two-level trust aggregation is enforced: aggregating the trust relationship among the neighbors' neighbors. For example, in Fig. 2, N_{a_0} should obtain all the trust placed upon N_{b_i} by N_{c_j}. Three methods can be employed to construct and update the GTM.

5.3.1 LTV Aggregation

The GTM of a node is constructed by fetching and merging the LTVs of its neighbors immediately after attestations to them are performed. Aggregation is implemented by overwriting the corresponding entries in the local GTM with gathered LTVs:

$$G^a_{i,*} = L^i, \ \forall i(N_i \in Nbr^a).$$

In most cloud deployment models, nodes tend to interact with their nearby peers, i.e., neighbors of a node also communicate with each other, especially when VMs of a cloud application are more likely to be deployed within a same cluster in the cloud. Hence, the GTM of a node may also contain the trust its neighbors place upon each other. As we can see from Fig. 2, by merging the LTV of N_{b_1}, N_{a_0} also gains trust information placed upon its neighbors N_{b_2} ($t_{1,2}$) and N_{b_3} ($t_{1,3}$). When considering scenarios where nodes have little common in communication peers, multi-level of trust aggregation can be used, instead of the two-level in our case. EigenTrust [21] shows that this multi-level aggregation still has a fast convergence.

5.3.2 GTM Aggregation

To gather more detailed attestation relationship, the GTMs of a node's neighbors can be aggregated. To avoid recursively expanding the GTM, a Sliced GTM (*SGTM*), denoted as $SG^{b,a}$, is returned to N_a to only contain the trust information of all N_a's neighbors maintained by N_b:

$$SG^{b,a} = [(G^b_{i,*})^T]^T \cup [G^b_{*,i}], \ \forall i(N_i \in Nbr^a).$$

$SG^{b,a}$ is then extended into G^a. The max operator is used for updating the freshness of the trust information:

$$G^a_{i,j} = \max(G^a_{i,j}, SG^{b,a}_{i,j}), \ \forall b(N_b \in Nbr^a).$$

In Fig. 2, as N_{a_0} is the neighbors of N_{b_1}, its LTV is maintained in N_{b_1}'s GTM. As N_{b_1} also maintains the LTVs of N_{c_7} and N_{c_5}, $t_{7,3}$ and $t_{5,2}$ are returned in SG^{b_1,a_0}. Moreover, as N_{b_1} is also performing GTM extension, $t_{4,3}$, $t_{8,1}$, and $t_{6,1}$ can also be returned. As the peers are at the same time extending their GTMs, trust is aggregated iteratively.

As defined, $SG^{b,a}$ also contains L^b, when $N_b \in Nbr^a$. Hence SGTM can be returned with each attestation instead of LTV. However, in the bootstrapping stage, when nodes' GTMs are mostly empty, LTV merging will facilitate a faster convergence.

5.3.3 Trust Dissemination

As attestation tickets are transient by nature, trust information in GTM will soon become obsolete. On the other hand, as nodes are attesting to each other due to occurring events, newly generated attestation tickets can be sent to the set of corresponding nodes to update the trust information maintained in their GTMs:

$$I^a_b = \{N_i \mid \forall i(G^a_{i,b} \neq (0,0))\}.$$

I^a_b denotes the set of nodes that have also attested to N_b. As in Fig. 2, a new attestation ticket $t_{0,1}$ from N_{a_0} to N_{b_1} is sent to the nodes that have also attested to N_{b_1}: N_{b_2}, N_{b_3}, N_{c_6}, and N_{c_8}. On receiving this ticket, those nodes first verify the signature of the AIK [3] signing the ticket, and then update $t_{0,1}$ in their GTMs.

6 CLOUD TCB ATTESTATION

In this section, we present how fine-grained cloud TCB attestations are achieved with the RepCloud framework. A *RepVisor* (Section 8) is deployed on each node to enforce the *Decentralized Attestation*. The *attestation delegate* is deployed in the cloud infrastructure in the similar way as the centralized attestation schemes [5], [6], [9]. The delegate receives the attestation requests from cloud customers, and returns attestation reports. With the *web-of-trust* constructed by *DA*, the delegate is able to identify and attest to the properties of the dependency for a large-scale cloud application with affordable effort.

When receiving the attestation request, attestation delegate first locates all the nodes hosting the target application's VMs. It then constructs a partial *web-of-trust* by attesting to the nodes and aggregating their locally maintained trust matrix, i.e., the GTMs. From the *web-of-trust*, it further identifies the *active static dependency*, and initiates attestations to necessary targets. Finally, when the partial *web-of-trust* converges, it aggregates the attestation tickets from the web. The *attestation report* is then generated to contain the properties of the target application's $cTCB_{App}$.

6.1 Constructing Web-of-Trust

As most of the application's hosting nodes have communications with each other, due to the interactions among its VMs, their trust evidence is mostly recorded in each other's GTM. Therefore, gathering and combining a few GTMs from selected nodes will be sufficient to aggregate the attestation tickets for the needed nodes. To facilitate this process, customers may first specify a set of entrance VMs according to its application's business logic. For example, they can specify the smallest subset of VMs that have altogether interacted with all the rest VMs in the application. This helps the delegate to assemble necessary *web-of-trust* with less effort.

Obtaining the initial set of VMs, the delegate constructs the *web-of-trust* iteratively (Fig. 3):

1) The delegate attests to the nodes hosting the initial VM set, and fetches their GTMs (Step 1). These GTMs are merged to form the initial *web-of-trust*.
2) From this web, the delegate determines whether all the VMs' hosts are identified. Additional attestations are performed to the missing nodes, with their GTMs merged (Step 2). Now the updated *web-of-trust* contains all the hosts of the target application, and part of the hosts' dynamic dependency.
3) All the hosts are added to the critical nodes set C. The rest steps identify the dynamic and static dependency for the nodes in C.
4) The delegate now searches the *active static dependency* for all the nodes in C (Step 3). In our cloud model, the static dependency relationship is defined by the administrators, and managed in a central

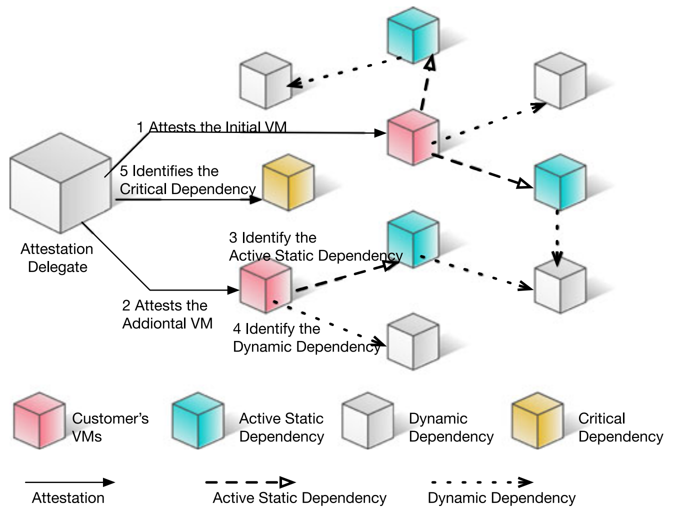

Fig. 3. Simplified repcloud web-of-trust construction.

database (Section 8). For example, the *Scheduler* node is marked as the static dependency of the *Compute* node, which is managed by the *Scheduler*. Therefore, every node marked as the static dependency of the critical nodes in C and also appears in the *web-of-trust*, is iteratively added to the critical node set C.

5) The delegate further searches dynamic dependency nodes, based on the attestation relationship from the *web-of-trust* (Step 4). For example, for the nodes that have interactions with most of the other nodes, but have not been attested to by the inspector. Or the nodes that have frequently interactions to the node in C, but have only had little interactions with other nodes on the web.

6) Various reputation calculations can be applied to identify critical nodes from the given *web-of-trust*. Further investigations are out-of-scope for this paper. We will discuss related topics in our future work. Accordingly, further attestations are performed to the identified semantic critical dependency nodes (Step 5). These nodes are added to set C.

7) With the updated C, the dynamic and static dependency discovery procedures are iterated until the delegate is convinced that most of the relevant trust information for attesting to the target application's *cTCB* has been gathered. For example, a new iteration only introduces a very few new nodes to the *web-of-trust*.

More attestation strategies can be developed to satisfy various application scenarios. Our current framework only serves as an example to demonstrate how the attestation delegate can better understand the trust dynamic inside the cloud, and perform fine-grained *cloud TCB* attestations. On the other hand, as the delegates are deployed on cloud nodes, their hosts participate in *Decentralized Attestations*. This greatly reduces repeatedly attestations' overheads, especially when customers require regularly attestations. Moreover, by strategically installing multiple delegates in different locations inside the cloud, and allowing them to share attestation tickets, more efficient attestation schemes can be achieved. We leave this to our future work.

TABLE 1
Simplified Aggregated Attestation Report

VM Identity	Host Identity	Measurement Values	Property List
VM_A	$Host_1$	$Hash_1$	$\{Properties_{A1}\}$
	$Host_2$	$Hash_2$	$\{Properties_{A2}\}$
	...	...	...
VM_B	$Host_1$	$Hash_1$	$\{Properties_{B1}\}$
	...	...	...
	...	...	...

6.2 Aggregating Attestation Tickets

To generate the *Attestation Report* for a target cloud application and its $cTCB_{App}$, the delegate first aggregates and parses the attestation tickets it gathers from the *web-of-trust*. For each VM in the application, the delegate identifies its host, and searches the host's dependencies by tracing the attestation relationship from the *web-of-trust*. All the dependencies' attestation tickets altogether represent the properties of the VM's $cTCB_{Node}$. Different report generation scheme can be implemented for different cloud's usage scenario. In this paper, we only focus on how relevant attestation tickets are collected, i.e., how the *web-of-trust* is constructed. We leave more designs to the different schemes to our future work.

Table 1 presents an example *Aggregated Attestation Report* for a target application. Each entry contains the *measurement value* for the *coreTCB* of a cloud node, which is identified as the dependency of a specific VM. Each measurement is translated to a list of *property descriptions*, which represents the expected behaviors of the node's *coreTCB*. Therefore, by aggregating all the properties for the relevant entries for a VM, the $cTCB_{Node}$ for the VM's host is attested to. All VMs' $cTCB_{Node}$ therefore construct the application's $cTCB_{App}$.

6.3 Generating Attestation Reports

To hide the host identity and dependency cloud service composition, in the final *Attention Report*, only the *properties* are included. Table 2 presents the exemplar properties. These are usually defined in the Service Level Agreements (SLAs) by the CSPs and certified by trusted authorities. We leave the property determination and certification to our future work. Redundancy exists when aggregating the properties. This is because different nodes may have the same *coreTCB*. This is due to the homogeneity of the cloud's basic infrastructure services. For example, most *Computes* in a same cluster have the exact same platform configuration. Therefore most of the dynamic dependency nodes of the target host may have the same measurement value. Consequently, the properties of nodes with the same *coreTCB* are listed only once. On the other hand, as VMs are usually deployed in the same clusters, redundancy also exists among the supporting services. For example, VMs' hosts may share the

TABLE 2
Simplified Cloud TCB Attestation Report

VM Identity	Property List
VM_A	$\{VM\ Memory\ Encryption, Storge\ Encryption,\}$
VM_B	$\{High\ Availability\ Storage, Redundant\ Instance,\}$
...	...

same management node, or have management node exhibiting the same trust evidence. Therefore, the report can maintain a list of all the different properties, and have each entry record references to the target ones.

On the other hand, the services not serving the target VM but have interacted with the VM's host may be counted as the VM's *dynamic dependency*. For example, a node ($Host_1$) hosts two VMs (VM_A and VM_B). VM_A uses a logging service deployed on node $Host_2$), which results in the interactions between $Host_1$ and $Host_2$. This makes $Host_2$ is also the dependency for VM_B. In this case we assume that the *property* deduction procedure is fine-grained enough to identify whether a service component is configured to serve a target VM. Therefore, when translating properties from the measurement value and the relevant measurement list, the delegate will understand that the logging service on $Host_2$ does not service VM_B. Therefore, the properties of *logging service* for $Host_2$ will be excluded.

VMs having the same property list may indicate that they are hosted on the same node. However, as the *report* hides the real identity of each physical node, this assumption is not assertive. One the other hand, VMs having different trust evidence list imply that they are hosted on different nodes. Although this leaks partial information for VM deployments to the customers, we argue that it is more important for the customer to know that these VMs are consuming different services.

7 Evaluations

We implemented our RepCloud protocols and simulated the cloud environment for evaluation. We counted the number of actions performed, e.g., total interactions and total attestation, and compared them with two kinds of current cloud attestation schemes.

7.1 RepCloud Simulator

The RepCloud simulator is built on top of a P2P simulator, the PeerSim [24]. We use the overlay network in PeerSim to simulate the interactions among nodes in the cloud. We further implemented a new layer of overlay network for the simulation of VM interactions.

Node model. We differentiate two kinds of nodes: *Managers* and *Hosts*, representing cloud managing facilities, e.g., the *Schedule*, *Storage*, and the VM hosting nodes, i.e., the *Compute* node. VMs are grouped into cloud applications, and applications are assigned to hosts in a same cluster with round-robin deployment strategy [25]. Every host can run a number of VMs simultaneously. Each VM runs for a *length* of time, and halts afterwards. A cloud application halts when there is no VM remaining.

The *size* of an application is the number of VMs it contains. We specify a baseline for the size, and a *similarity* value to denote a percentage for the size of each application that can be larger than the baseline, i.e., applications have random sizes within the range from $size$ to $size \times (1 + similarity)$. Similarly, we specify the *VM similarity* as the diversity factor for the VMs inside an application. The length of a VM is also specified according to a base value and this factor.

The integrity state of a host is represented as integer numbers. The changes in state can result from malicious behaviors, as we discussed in our adversary model, or from trustworthy operations, i.e., applying security patches, and they can only be discovered by attestations. However, in our experiment, we do not differentiate the trustworthiness of a state, as our main purpose is to evaluate how the changes in state are discovered. We also do not consider the internal states of VMs.

Network model. A manager connects to a number of hosts to form a cluster. It communicates with them for management tasks, e.g., load-balancing instructions. Managers of the clusters are connected to a root manager to form the cloud, which is regarded as the CC (cloud controller). A VM only communicates to the VMs that belong to the same application as its, and cloud applications usually have different communication patterns. However, the aggregated patterns of a host tend to have a random distribution, especially when considering the multi-tenancy and dynamic nature of a cloud. For simplicity, we hence specify each VM to randomly communicate to others in the same application. We define the *frequency* of a VM as the number of communication actions it performs to another VM within a cycle. Total communications in an application performed within a cycle hence equal to $size \times frequency \times (size - 1)$. Frequency can also relate to VM similarity, in which case the total number should be calculated separately. Communications among VMs trigger communications among hosts, as they are enforced by the hosts.

Simulation execution. In the initialization phase, hosts and managers are generated and linked accordingly. Cloud applications are deployed to occupy the full capacity of the cloud, i.e., the total number of VMs that can be run simultaneously. New applications are deployed regularly, to keep the overall load stable. VMs are also migrating among nodes inside a same cluster for simulating the effects of load balancing.

Simulation is executed in cycles, in each of which every host executes all its VMs by fetching a list of target communication VMs and generating communication events (actions) to the hosts of the targets. We use the event and time model of PeerSim: time proceeds with the occurrence of events. Every event is generated with a timestamp, and is executed in sequence. The global time is set to the timestamp of the currently processing event. We define the time for performing a communication action as our basic time unit, which is mapped to a millisecond for better presentation. Complicated communications can hence be regarded as a sequence of actions. With the time mapping defined, our experiment results can be easily mapped for real system analysis. In the following text, we refer to the time as our simulated time by default. In our simulation, different kinds of events use different random number generators, e.g., cloud app deployment, migration or attacks, to keep their indecency and achieve the same VMs interaction pattern.

7.2 Cloud Attestation Schemes

Three attestation schemes are simulated and evaluated, a *centralized scheme* (CEN), a fully *decentralized scheme* without reputation systems (DECEN) and our *RepCloud scheme* (REP). We further modified CEN and DECEN to implement fine-grained cloud attestation for evaluation, i.e., to enable each node to determine the trust state of its own cloud TCB with these two schemes.

TABLE 3
Basic Simulation Parameters

Simulation	Length of a simulation cycle (minutes)	1
	Total simulation time (hours)	12
	Bootstrap time (minutes)	10
	Attestation freshness (seconds)	10
Network	# of clusters in the cloud	3
	# of compute nodes per cluster	100
	# of VMs per compute node	20
	# of total simultaneous VMs	6,000
Cloud Apps	app generation interval (minutes)	1
	# of app generated per interval	60
	# VMs per app	10
	length per VM (minutes)	10
	VM frequency	300
	App similarity	2
	VM similarity	0.6
Attackers	range of attack intervals (seconds)	1-10
	# of host to attack per interval	1

CEN is used in [5], [26]. Attestations are performed by the central attestation delegates to their connecting hosts regularly. Delegates are attested to by higher-level delegates iteratively. In existing proposals, VMs on each host do not have the knowledge of the properties of the target host it is communicating to. Customers can only attest to top-level delegate, which can then report the properties of all nodes that users' VMs are relying on. In our experiments, in order to achieve fine-grained attestation with this scheme, attestation tickets generated by a delegate are broadcast to all its managed nodes. This allows CEN to achieve the lowest state-change-discovery-delay, as each host will be able to determine whether the state changes have occurred, and take actions accordingly.

DECEN is used [6], with which customers are able to directly attest to their VM's hosts. To attest to the cloud dependency with this scheme, we allow every host to attest to its dependency (its neighbors). However, trust dissemination and aggregation are not used. In RepCloud scheme, before a host is communicating to another, it first searches its LTV and GTM for a valid attestation ticket to the communication target. On a search miss, it will perform the attestation, and disseminate the ticket as we discuss in Section 5.

The detailed configuration of our simulations is presented in Table 3. *Attestation freshness* denotes that attestation tickets can only be valid for 10 seconds. We simulated 600 applications running at the same time in the cloud, each of which contains 10 to 30 fully connected VMs. An application runs for 10 to 16 minutes with a new one deployed after it terminates. The node communication patterns are hence changing from time to time. Each VM generates 300 to 480 communication actions per second to every VMs in the same app.

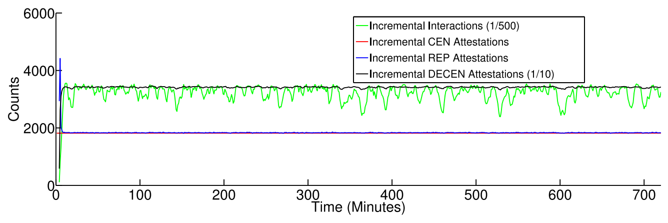

Fig. 4. Attestation counts summary. Interaction counts are scaled to $1/500$. DECEN attestation counts are scaled to $1/10$.

TABLE 4
Total Tampered Interaction Counts

	Interactions	Attestations	Tampered Interactions
REP	$1.15E9$	$1.30E6$	$1.79E6$
CEN	$1.15E9$	$1.30E6$	$1.78E6$
DECEN	$1.15E9$	$2.4E7$	$1.76E6$

7.3 State-Change Detection

We simulate attackers who change the state of a random host in random intervals after the bootstrapping phase. We evaluate RepCloud performance with the state-change detection overhead. Two important criteria are considered: 1) *Total Tampered Interactions:* all communications made towards a host with its presumed state (the state deduced from the latest attestation) different from its current state, as they are relying on trust states that have already changed. 2) *Total Attestations:* the total number of attestations performed for discovering the state changes of nodes.

Fig. 4 depicts an incremental statistics of the state change detection overhead for the three attestation schemes. The data is refreshed every 10 minutes. Attestation counts of the centralized scheme (CEN) are depicted in a strait line, as managers perform attestations to their hosts in a predefined interval (10 seconds). In RepCloud, attestation counts boost to a high level in the first few minutes, as it is in the bootstrapping phase and the LTVs and GTMs are mostly empty. Later data shows that RepCloud still exposes stable attestation patterns while achieving only a slightly higher level of average attestation counts. For the decentralized scheme (DECEN), as it does not share attestation results among nodes, every node will attest to its neighbor when the tickets are regarded as obsoleted (after 10 seconds). Hence the attestation counts are much larger than the others (around 20 times from our results). Table 4 shows that, RepCloud achieves fine-grained cloud TCB attestation with only a slight increase in attestation overhead, and without reputation systems, redundant attestation efforts will boost significantly (as in DECEN).

7.4 Trust Dissemination

Ticket reporting. In RepCloud, every time when a new attestation ticket towards a target node is generated, it is sent to all the node that have also attested to the target, according to the attestation relationship in the GTM. However, in the centralized scheme, the manager does not possess the communication patterns of each host. Hence, it can only broadcast the new ticket to every node as we discussed before. As we can see from Fig. 5, the tick dissemination count is around

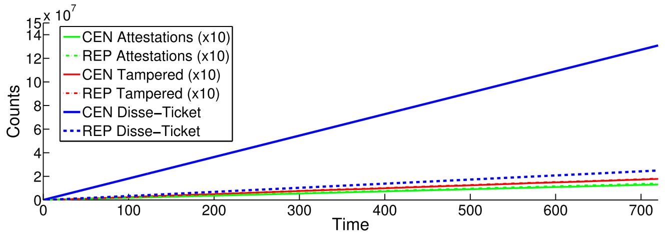

Fig. 5. Total ticket dissemination overhead.

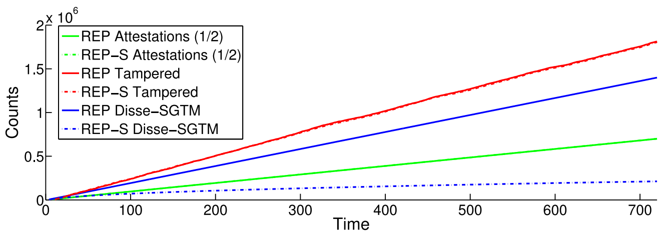

Fig. 6. Total SGTM dissemination overhead.

six times as RepCloud's, while the total attestation and interaction counters remain in the same level. Besides, in centralized scheme, VMs of an application can only be deployed within a same cluster to reduce the broadcast domain. However, in RepCloud, an application can scattered among clusters, while remaining a low level of dissemination overhead, as tickets are disseminated according to nodes communication patterns. More importantly, RepCloud reserves the mutual-attestation relationship, from which a higher level of trust semantic is deduced.

Moreover, as the communication patterns are changing, communication relationship is also transient. For example, N_a may no longer interact with N_b as all related VMs on it are either halt or migrated out. But $PCR_{a,b}$ may still exist in other nodes' GTM, and new tickets towards N_b are still kept sending to N_a. An obsolete interval can be defined to further reduce the ticket disseminate overhead in RepCloud. Attestation tickets generated before the interval will not reveal attestation relationship among the nodes.

SGTM fetching. Besides ticket disseminating, RepCloud incurs an extra overhead: SGTM of a target node is fetched every time when a local attestation towards it is performed. However, as newly generated attestation tickets are disseminated to the corresponding nodes, GTM of each node is kept updating continuously. Hence each SGTM fetching may only contain few updates, especially when the communication patterns of nodes are in a relatively stable phase. To reduce the overhead, SGTM fetching can be delayed for an interval of time (1 minute in our case) when the updated entries of the last time is lower than a threshold. We also specify cultivated delay interval, i.e., the interval will be increased (by 1 minute) every time when the both the current and the last update are lower then the threshold. The interval is reset once the update rate is higher than the threshold.

In our experiment, we specify the threshold as whether the size of GTM is changed (i.e., whether new entry is added), as it indicates the changes in communication targets of a node. All the other changes can be updated by ticket dissemination. Fig. 6 shows that 86 percent of the total SGTM dissemination counts are saved (REP-S), while preserving the same level of total attestation and interaction counts. We scale the total attestation counts to half for better presentation, as it overlaps with the original SGTM dissemination counts.

7.5 Security Analysis

As reputation systems are used in RepCloud to disseminate the *TCG Trust*, existing attacks to them should be considered [11]. However, we argue that as RepCloud transmit tamper-proof attestation tickets instead of raw reputation values, these attacks are avoided by default. For example, in reputation-based P2P trust management systems, nodes can disseminate forged trust information to promote its own reputation value or to degrade others. In RepCloud, the attestation tickets are signed by the AIK [3] of the platform, the private part of which can only be accessed by the TPM. We will present detailed countermeasurements to these known attacks to reputation systems in our future work.

Malicious cooperative [11] is another kind of attack to reputation systems, with which several tampered nodes cooperate to promote the reputation value of each other. In RepCloud, when all the nodes hosting the VMs of an application are tampered, they may report each other as trustworthy without being identified, and hence tamper the target application. However, with the multi-tenancy nature of the cloud, each of these hosts may at the same time host VMs from other applications, which may communicate to nodes in the cloud TCB of the those application. The cooperative should hence also incorporate these nodes. We argue that the cooperative should be sufficiently large to contain a web-of-trust closure for tampering a target application, e.g., by tampering the entire cluster. On the other hand, in the cloud TCB attestation procedure, an entrance node to the web-of-trust is attested to by a trusted third party. Hence, the larger the malicious cooperative is, the higher probability it will be discovered.

Attacks to the trusted computing framework have already been concerned in the Timestamp-based attestation scheme [20], which we used as the local attestation in RepCloud. Nonce in the attestation ticket is used to avoid reply attacks, and session keys are used to counteract man-in-the-middle attack. On the other hand, we do not consider runtime attacks to trusted computing mechanisms. We also do not consider hardware-based attacks.

8 Implementation

A RepVisor carries two responsibilities: 1) to gather the trust evidence (i.e., the attestation tickets) of its dependency nodes; and 2) to disseminate the tickets to facilitate the other nodes to gather the evidence they need. The locally stored tickets help the delegate to effectively determine the $cTCB_{Node}$ of a target node. Moreover, by analyzing the *web-of-trust* matrix, the delegate can deduce the trust dependency of the target, and hence to better identify the $cTCB_{App}$ for a target application. In this section, we present the implementation of the RepVisor and the attestation delegate.

8.1 RepVisor Structure

We built *RepVisor* based on the *NeuronVisor* we implemented in [27]. *RepVisor* prototype integrates with the OpenStack-XenServer system [28]. Fig. 7 depicts the XenServer-based

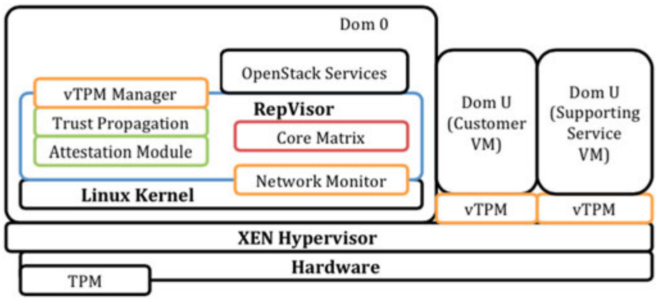

Fig. 7. RepVisor implementation architecture.

cloud node structure. Each node is deployed with a Xen hypervisor. Dom0 is the privileged virtual machine, which manages the underlying hardware and implements the virtualization layer. It also hosts necessary components to support the cloud services. OpenStack facilities, e.g., Compute or Scheduler [8], are deployed in Dom0, while Customer VMs are also deployed as DomUs.

RepVisor includes a user-space application and a Linux kernel module. RepVisor attests to and disseminates the properties of the *coreTCB* of a cloud node, which implements the fundamental responsibilities a node carries in a cloud infrastructure [29]. In our prototype, this *coreTCB* includes the entire Dom0. Therefore, to attest to this TCB, a core chain-of-trust [3], [30] should be built to include the software components for bootstrapping both the Dom0 and the OpenStack domains, together with all the service components loaded in these environments.

To enforce this chain-of-trust, XenServer's measured boot supplemental pack [28] is enabled. The IMA-enabled Linux kernel is also deployed in both the Dom0 [30] and the OpenStack domains to measure all the loaded software components inside. User space trusted computing services are also deployed in these domains, including TrouSers [31] for implementing the trusted computing services libraries, and the OpenPTS [32] for implementing remote attestations.

The vTPM manager links the chains-of-trust of the Dom0 and the OpenStack domains together. It instantiates a vTPM for each VM, including both the OpenStack VM and the customer VM. Currently, we deploy the TPM emulator to implement the vTPM, and we do not consider the property-based translations [17], [18], [33]. The binary codes of the TPM emulator are measured by the IMA in Dom0. This implements the vTPM-TPM binding. A more comprehensive research on this binding is in [34]. We leave the integration of this work with our system to future work.

The *Core Matrix* (or *Core*, for short) stores the trust evidence of the node's neighbors, particularly the LTV and the GTM. The *Core* is maintained by the *Attestation Module* and the *Trust Propagation Module*. The *Network Monitor* intercepts the communications among the DomUs, and queries the *Core* for whether the communication target host has recently been attested to. *Attestation Module* then attests to neighboring nodes when the *Network Monitor* fails to locate a recent attestation ticket. It updates the *Core* with the gathered attestation tickets (Section 5.2). Afterwards, *Trust Propagation Module* exchanges the updated *Core* with the neighboring nodes by enforcing the *Decentralized Attestation* protocols (Section 5.3). We now present each module respectively.

8.2 Network Monitoring

Decentralized Attestation targets are determined by communication patterns, which are perceived by the *Network Monitor*. *Network Monitor* intercepts all the communications of the host, and enforces decentralized attestation protocols when the locally maintained attestation ticket to the communication target node is not available or outdated.

We implement the *Network Monitor* as the xtable-addon module, the xtable extensions [35] to the Linux kernel in Dom0. Xtable monitors the communication traffic and performs user-defined addon actions when particular conditions are met. *Network Monitor* intercepts the network traffic sending from all the DomUs. It also maintains a data structure in the kernel space to record the newest timestamp of every target node. It suspends the traffic when the timestamp indicates obsolete attestation, and outputs the IP address of the communication target node through a kernel device. The *Attestation Module* running in the user space monitors this device and performs attestations described above to the target node. This attestation also triggers the trust disseminations, and updates the Core Matrix. The resulting updated timestamps are also written to the kernel device and stored in the *Network Monitor* module.

The traffic made by the RepVisor is not intercepted by the *Network Monitor*. This includes the attestation and the trust propagation traffic. On the other hand, incoming attestation requests are intercepted. This is because the attestation target is determined by the IP address of the intercepted communication target, which usually is the DomU hosting the customer VMs. However, decentralized attestations only consider the integrity of the Core Services, i.e., the Dom0 on each node. Therefore, the *Network Monitor* intercepts all the attestation requests addressed to the DomUs' IP, and returns Dom0's integrity values (the *PCR* values).

8.3 Active Attestation

RepCloud uses the *Active Attestation* scheme [36] that we designed based on the Timestamp Hashchain-based Attestation method in [20]. Active Attestation generates reusable attestation tickets. Attestation tickets reporting the measurement value for each node are actively generated by the node itself, with a nonce value bound to a globally agreed timestamp. This allows a ticket to be used by multiple attesters, while their freshness can still be verified by deducing the timestamp. In our previous work in [36], we applied this attestation scheme in the Trusted MapReduce system, which enforces intensive periodically active attestations. It reduces the attestation overhead time from larger than 5 seconds to around a few milliseconds (0.0016 seconds).

We integrate this scheme with our RepVisor prototype. Each node is equipped with a timestamp certificate, which binds an initial nonce value and an initial timestamp. An interval value is also included, which specifies how frequently a new nonce is calculated. The *Attestation Module* thus actively *quotes* its TPM's *PCRs* for the Dom0. It generates an attestation ticket with an updated nonce value using the nonce updating protocol in [36]

$$T = \{M_{PCR}, nonce\}_{K_{AIK}^{-1}}, step, Cert\{AIK\}.$$

A ticket T includes the trusted computing-compliant attestation ticket. In our implementation M_{PCR} includes the measurement values for $PCR[0\text{-}7]$, which record the measurement values of the local chain-of-trust for bootstrapping the platform; and $PCR[10]$, which is used by the IMA [30] for measuring the loaded application components, libraries and kernel modules. In addition, the *step* provides clues to deduce a timestamp from the *nonce* [20], [36].

When receiving attestation requests from the *Network Monitor*, the *Attestation Module* first asks the target node to return its newest attestation ticket, T. This target node is identified by its IP address provided by the *Network Monitor*. The *Attestation Module* then queries the privacy CA to

TABLE 5
Average Time Intervals for Data Transfer

	Transfer Completion Time (Seconds)	StDv (Seconds)	Overheads
without DA	245	3	-
with DA	251	3	2.44%

verify the AIK of the target node, and determines the freshness from the timestamp deduced from the nonce. Valid tickets will be updated to the LTV in the *Core*. We identify each node by its IP address. An LTV entry thus contains the IP address, the latest attestation ticket, and a deduced timestamp, which indicates the time the ticket was generated.

Trust Propagation Module then enforces the DA protocols. It fetches the LTV and the GTM or the (Sliced GTM) of the target node, and merges them into its own GTM. Updated GTM contain is then disseminated to the corresponding neighbors. In our implementation, each GTM entry contains the IP addresses of the attesting and the attested node, the attestation ticket, and the timestamp.

8.4 Preliminary Performance Analysis

We have implemented the RepVisor. But as a large-scale physical data centre environment is not available to us, the effectiveness of the RepCloud and the *Decentralized Attestation* is not tested in the production system. Here we briefly discuss the RepVisor implementation's performance based on a two-node RepCloud system. Our focus is the network latency, which will be the RepVisor's the major overhead, as the network traffic is halted and the DA is enforced when the target platform's latest attestation ticket is obsoleted.

In our experiments, we run two communicating applications in two DomUs deployed in two connecting RepVisor-enabled Xen hypervisors. The Xen's network-passthrough feature was *not* enabled. Application A sends a $2\mathrm{Gb}$ data file to application B over a $100\,\mathrm{Mbps}$ LAN, with only the two machines attached. A then records the overall time interval for completion.

We first examined the time interval when the DA was disabled (the *without DA* in the Table 5). We then examined how the DA would affect the time interval. In this experiment, the DA on A's RepVisor is enabled, which will halt the traffic in every 0.6 seconds, and perform an attestation to B's entire Dom0. The reason for choosing the attestation interval as 0.6 was because this is the minimum interval for consecutively quoting the PCRs of a TPM [36]. This simulated the worse case scenario when enforcing the DA. In this experiment, the trust aggregation and dissemination mechanisms were disabled, as the RepVisor enforces them in parallel to the general traffics, so that no obvious network latency will be introduced.

Each experiment is performed 10 times, and the average results are calculated. The standard deviation (StDv) is also presented. As shown in Table 5 only the overhead of 2.44 percent was introduced. This is because of the introduction of the *Active Attestation*. As discussed in [36], each *Active Attestation* only incurs the latency of 0.0016 seconds, when a node with less-frequency-changing PCR values are frequently attested. Moreover, when the trust dissemination and aggregation mechanisms are enforced, we expect the overheads to be even lower, as the attestation tickets are reused, so that the redundant attestations are greatly reduced.

9 Conclusion and Future Work

In this paper we presented the RepCloud to achieve cloud service dependency attestation. We defined the concepts of *Cloud Trusted Computing Base* to organize the cloud service dependency in a hierarchical manner. Based on these definitions, we further designed the *Decentralized Attestation* method to implement *cTCB* identifications and attestations. *DA* presents a new approach to manage trust in the cloud by taking advantage of the reputation systems. We implemented RepCloud in our simulated cloud environment. The evaluations showed that RepCloud incurred much less overhead than the current schemes for achieving fine-grained *cTCB* attestations. Further analysis showed that, RepCloud still enforced the same level of state-change-discovery delay as the current centralized cloud attestation schemes.

In our future work, we aim to evaluate RepCloud and RepVisor in a production cloud infrastructure, which we could not currently achieve due to the lack of resource. More security analysis and evaluations will also be performed. We will further integrate *reputation calculations* and *service dependency discovery systems* in RepCloud to achieve more precise *cTCB* attestation. Various strategies for determining the critical communications to achieve more accurate attestations will also be investigated.

References

[1] F. Azmandian, M. Moffie, M. Alshawabkeh, J. Dy, J. Aslam, and D. Kaeli, "Virtual machine monitor-based lightweight intrusion detection," *SIGOPS Oper. Syst. Rev.*, New York, NY, USA, vol. 45, no. 2, pp. 38–53, Jul. 2011. [Online]. Available: http://doi.acm.org/10.1145/2007183.2007189

[2] Cloud security alliance. (2016, May). [Online]. Available: http://www.cloudsecurityalliance.org

[3] Trusted Computing Group. (2016, May). [Online]. Available: http://www.trustedcomputinggroup.org

[4] Trusted platform module main specification. (2014, Oct.). [Online]. Available: www.trustedcomputinggroup.org/resources/tpm_main_ specification

[5] N. Santos, K. P. Gummadi, and R. Rodrigues, "Towards trusted cloud computing," in *Proc. Conf. Hot Topics Cloud Comput.*, 2009.

[6] J. Schiffman, T. Moyer, H. Vijayakumar, T. Jaeger, and P. McDaniel, "Seeding clouds with trust anchors," in *Proc. ACM Workshop Cloud Comput. Security*, 2010, pp. 43–46.

[7] I. M. Abbadi, "Clouds trust anchors," in *Proc. IEEE 11th Int. Conf. Trust Security Privacy Comput. Commun.*, 2012, pp. 127–136. [Online]. Available: http://dx.doi.org/10.1109/TrustCom.2012.107

[8] OpenStack. (2016, May). [Online]. Available: http://www.openstack.org

[9] J. Schiffman, Y. Sun, H. Vijayakumar, and T. Jaeger, "Cloud verifier: Verifiable auditing service for IaaS clouds," in *Proc. IEEE 9th World Congr. Serv.*, Jun. 2013, pp. 239–246.

[10] E. Keller, J. Szefer, J. Rexford, and R. B. Lee, "NoHype: Virtualized cloud infrastructure without the virtualization," *SIGARCH Comput. Archit. News*, vol. 38, pp. 350–361, Jun. 2010.

[11] K. Hoffman, D. Zage, and C. Nita-Rotaru, "A survey of attack and defense techniques for reputation systems," *ACM Comput. Surv.*, vol. 42, pp. 1:1–1:31, Dec. 2009.

[12] S. Berger, R. Cáceres, D. Pendarakis, R. Sailer, E. Valdez, R. Perez, W. Schildhauer, and D. Srinivasan, "TVDc: Managing security in the trusted virtual datacenter," *SIGOPS Oper. Syst. Rev.*, vol. 42, pp. 40–47, Jan. 2008.

[13] N. Santos, R. Rodrigues, K. P. Gummadi, and S. Saroiu, "Policy-sealed data: A new abstraction for building trusted cloud services," in *Proc. 21st USENIX Conf. Security Symp.*, 2012, pp. 10–10.

[14] S. Butt, H. A. Lagar-Cavilla, A. Srivastava, and V. Ganapathy, “Self-service cloud computing,” in *Proc. ACM Conf. Comput. Commun. Security*, 2012, pp. 253–264.
[15] S. M. Habib, V. Varadharajan, and M. Mühlhäuser, “A trust-aware framework for evaluating security controls of service providers in cloud marketplaces,” in *Proc. 12th IEEE Int. Conf. Trust, Security Privacy Comput. Commun.*, 2013, pp. 459–468. [Online]. Available: http://dx.doi.org/10.1109/TrustCom.2013.58
[16] A. Nagarajan, V. Varadharajan, and N. Tarr, “Trust enhanced distributed authorisation for web services,” *J. Comput. Syst. Sci.*, vol. 80, pp. 916–934, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022000014000142
[17] L. Chen, H. Löhr, M. Manulis, and A.-R. Sadeghi, “Property-based attestation without a trusted third party,” in *Proc. 11th Int. Conf. Inform. Security*, 2008, pp. 31–46.
[18] P. Jonathan, S. Matthias, E. Van Herreweghen, and W. Michael, “Property attestation—Scalable and privacy-friendly security assessment of peer computers,” IBM Research, Zurich, Switzerland, Tech. Rep. RZ 3548, 2004.
[19] P. Barham, R. Black, M. Goldszmidt, R. Isaacs, J. MacCormick, R. Mortier, and A. Simma, “Constellation: Automated discovery of service and host dependencies in networked systems,” Microsoft Research, Redmond, WA, USA, Tech. Rep. MSR-TR-2008-67, Apr. 2008. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=64514
[20] F. Stumpf, A. Fuchs, S. Katzenbeisser, and C. Eckert, “Improving the scalability of platform attestation,” in *Proc. 3rd ACM Workshop Scalable Trusted Comput.*, 2008, pp. 1–10.
[21] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, “The eigentrust algorithm for reputation management in p2p networks,” in *Proc. 12th Int. Conf. World Wide Web*, 2003, pp. 640–651.
[22] Trusted computer system evaluation criteria. (2016, May). [Online]. Available: http://en.wikipedia.org/wiki/Trusted_Computer_System_Evaluation_Criteria
[23] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig, “Trustvisor: Efficient TCB reduction and attestation,” in *Proc. IEEE Symp. Security Privacy*, Mar. 2010, pp. 143–158.
[24] A. Montresor and M. Jelasity, “PeerSim: A scalable p2p simulator,” in *Proc. IEEE 9th Int. Conf. Peer-to-Peer Comput.*, Sep. 2009, pp. 99–100.
[25] Eucalyptus. (2015, May). [Online]. Available: http://www.eucalyptus.com
[26] F. J. Krautheim, “Private virtual infrastructure for cloud computing,” in *Proc. Conf. Hot Topics Cloud Comput.*, 2009.
[27] A. Ruan and A. Martin, “Neuronvisor: Defining a fine-grained cloud root-of-trust,” in *Proc. 6th Int. Conf. Trustworthy Syst.*, 2014, pp. 184–200.
[28] Getting started with XenServer and OpenStack. (2016, May). [Online]. Available: https://wiki.openstack.org/wiki/XenServer
[29] A. Ruan and A. Martin, “Repcloud: Achieving fine-grained cloud TCB attestation with reputation systems,” in *Proc. 6th ACM Workshop Scalable Trusted Comput.*, 2011, pp. 3–14.
[30] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, “Design and implementation of a TCG-based integrity measurement architecture,” in *Proc. 13th Conf. USENIX Security Symp.*, 2004, pp. 16–16.
[31] TrouSerS—The open-source TCG software stack. (2015, May). [Online]. Available: http://trousers.sourceforge.net/
[32] Open platform trusted service user’s guide. [Online]. Available: http://iij.dl.sourceforge.jp/openpts/51879/userguide-0.2.4.pdf, 2011
[33] A.-R. Sadeghi, C. Stüble, and M. Winandy, “Property-based TPM virtualization,” in *Proc. 11th Int. Conf. Inform. Security*, 2008, pp. 1–16.
[34] S. Berger, R. Cáceres, K. A. Goldman, R. Perez, R. Sailer, and L. van Doorn, “vTPM: Virtualizing the trusted platform module,” Vancouver, B.C., Canada, in *Proc. 15th Conf. USENIX Security Symp.*, Art. no. 21, 2006.
[35] Xtables-addons. [Online]. Available: http://xtables-addons.sourceforge.net
[36] A. Ruan and A. Martin, “TMR: Towards a trusted MapReduce infrastructure,” in *Proc. 2012 IEEE Eighth World Congr. Serv.*, 2012, no. 8, pp. 141–148. [Online]. Available: http://dx.doi.org/10.1109/SERVICES.2012.28

Anbang Ruan is working toward the DPhil degree in computer science from the Department of Computer Science, University of Oxford. His research interests include research in cyber security and system security for years, and is currently focusing on researching and building the trusted public cloud infrastructure. He has played an important role in Oxfords system security team in participating European-wide projects, funded by FP7 and EPSRC.

Andrew Martin undertakes research and teaching in the area of systems security, in the University of Oxford. He conceived the University’s new Cyber Security Centre and helps to direct it, leading the University’s successful bid to be recognised as a Centre of Excellence in Cyber Security Research. His recent research focus has been on the technologies of trusted computing, exploring how they can be applied in grid and cloud computing contexts, as well as in mobile devices, in order to address their emerging security challenges. He has published extensively in this area, hosting several related international events in Oxford and speaking on the subject all over the world.