

# Breaking Down the Monarchy: Achieving Trustworthy and Open Cloud Ecosystem Governance with Separation-of-Powers

Anbang Ruan<sup>1,2,4</sup> Ming Wei<sup>3</sup> Andrew Martin<sup>2</sup> David Blundell<sup>5</sup> David Wallom<sup>1</sup>

<sup>1</sup>Oxford e-Research Centre, University of Oxford, UK

<sup>2</sup>Department of Computer Science, University of Oxford, UK

<sup>3</sup>School of Software and Microelectronics, Peking University, China

<sup>4</sup>Octa Innovations, China <sup>5</sup>100 Percent IT Ltd, UK

anbang.ruan@oerc.ox.ac.uk wei\_ming@pku.edu.cn andrew.martin@cs.ox.ac.uk

david.blundell@100percentit.com david.wallom@oerc.ox.ac.uk

**Abstract**—The cloud computing ecosystem is in urgent need of effective and practical trust establishment schemes. Cloud customers currently lack approaches to effectively verify the genuine behaviours of cloud services. They can only *blindly believe* that the Cloud Service Providers (CSPs) are honest enough to not tamper with their data, while many others have avoided using the cloud entirely. Trust establishment schemes, such as cloud auditing and cloud attestation systems, lack controls and transparency over their trust building processes, which only *blur* the effectiveness of the proclaimed trustworthiness. We argue that these problems ultimately result from the CSPs’ autocratic governance over all the activities inside the cloud. In this paper, we present a *Separation-of-Powers (SoP)* model by referencing the similar concepts from the discipline of the political philosophy. We define three independent roles to separate the powers of *definition*, *enforcement*, and *inspection* from the CSPs. These roles form the *collaborative-restrictive* relationship to facilitate trustworthy cloud services and achieve the *balance-of-powers*. We believe a model of this kind will open new opportunities for achieving trustworthy and open cloud ecosystem governance.

## I. INTRODUCTION

When customers outsource their private data or applications to third-party cloud computing infrastructures, they have to make critical decisions on how much they can trust the Cloud Service Providers (CSPs). With already abundant and still rapidly growing numbers of CSPs in the market, they seem to have plenty of choice. However, technology barriers and insufficient regulations have limited the range of these choices, and force customers to make compromises: this is mainly because CSPs have become the omnipotent authority for *defining*, *enforcing* and *inspecting* the services they provide. Customers, especially those requiring a specific level of assurance, need to know whether the services they purchased have been genuinely enforced, and whether only authorized operations are performed on their outsourced data or applications. But current CSPs lack the mechanisms to effectively provide this information to the customers.

Under this circumstance, customers have no choice but only to trust the CSPs whom they *believe* as relatively trustworthy. Most of the time, they choose the big CSPs, in the hope that these CSPs are less likely to risk damaging their reputations by tampering with their customers’ data. We define

this kind of trust as *Blind Trust*. Blind trust leads customers to bigger CSPs. It makes the big ones bigger and the small ones ultimately extinct. This eventually brings into existence *Monarchies* in the Cloud Computing ecosystem. As former Sun CTO Greg Papadopoulos once commented [5]: “*there will be, more or less, five hyperscale, pan-global broadband computing services giants.*”. These powerful monarchies will eventually control every customer’s detailed activities inside the cloud and leave them with no choices but compromises.

To establish trust with CSPs, various cloud auditing schemes [2], [7], [3], [6] are enforced by authorities who have established and well-known trustworthiness, e.g. government agencies. They evaluate the cloud implementations against certain criteria and issue certificates to vouch for the CSPs’ trustworthiness. However, these evaluations are usually performed in a one-off manner, with a relatively long interval, e.g. once a year. It is thus hard for a certificate to reflect the most up-to-date status of a cloud infrastructure. Meanwhile, they examine the entire infrastructure at a higher-level point of view. It is therefore difficult for a customer to determine, on a session-basis, whether their exact SLAs (Service Level Agreements) have been fulfilled.

On the other hand, several authors have proposed the concepts of cloud attestation [21], [11], [12], [20], which allows customers to verify the genuine enforcements of the cloud services by using Trusted Computing technology [9], [10], [19]. In these systems, the cloud servers have built-in tamper-proof trustworthy hardware chips: the *TPMs* [10] (Trusted Platform Modules). TPMs are equipped *PCRs* (Platform Configuration Registers) to reliably record the genuine behaviours of a computing platform. *Attestation* services are hence designed to gather the TPM-generated trust evidence, as well as to testify to the cloud services’ properties. However, these methods mostly require CSPs themselves implement the *attestation delegates* inside their cloud infrastructure. These delegates attest all the services inside the cloud to vouch for their genuine behaviours. Though customers could further attest the delegates’ integrity to infer the cloud’s trustworthiness, these attestations only prove their *existence* rather than their *effectiveness*. It is still opaque to customers how their acquired cloud services are examined.

We refer to the trust semantics established by these cloud auditing or cloud attestation schemes as *Blurred Trust*, as even though the trustworthiness of a CSP is audited or attested by a third party, not enough information is given to actually prove the establishment of trust. Blurred trust sheds light on a new era where the CSPs' authorities can be challenged and their powers be controlled, but the lack of effective and practical models and technologies still allow the CSPs to rule customers' activities inside the cloud.

From a sociological perspective, people's rights are better protected when the powers of governance are separated and enforced by cooperative and mutually restrictive authorities. Their rights are further strengthened when people have the right to substitute a misbehaving party from an authority with a competing party from the same authority. We observe that similar principles are also applicable to the governance in a cloud ecosystem: when a CSP is the only authority to testify to the enforcement of its own behaviours, customers have no means to verify whether the services they procure are enforced; when a CSP has become the only authority to define the properties of its own behaviours, customers have no means to verify whether the enforced services will fulfil their requirements. Therefore, we argue that the ultimate causes of the trust issues come from the CSPs' overpower; and not until these powers are separated will customers establish effective trust in the CSPs.

In this paper, we address the issues of breaking down the *Cloud Monarchies*. We present a *Separation-of-Powers (SoP)* model to disaggregate a CSP's authorities, and distribute them to three independent and collaborative *roles*, with each role's function provided by multiple competing *enforcement parties* (or *parties* for short). In order to achieve the *Balance-of-Powers*, we further introduce restrictions enforced among these roles, as well as among the parties of a same role. This model allows the cloud computing ecosystem to achieve the followings characteristics:

- 1) *Trustworthiness*. The collaborations and restrictions enable effective identifications for each role's misbehaviours. Customers are also endowed with *Freedom-of-Choices*, as they are able to substitute a role's enforcement party which does not satisfy their requirements.
- 2) *Openness*. CSPs of diverse sizes and established credibilities are all able to take part in the ecosystem, as they are all capable of obtaining effective trust from the customers. The *SoP* also encourages two new communities for implementing the services of *behaviours inspections* and *property definitions*.

In the next section, we first present the related work and elaborate our motivations with a case study. In Sec. III we present the *SoP* models. In Sec. IV we present a framework for implementing an *SoP*-compatible cloud infrastructure. Building blocks are also introduced, along with a preliminary implementation design, but we leave the complete implementation and evaluations to our future work. Finally, Sec. V concludes the paper and presents the future work.

## II. CASE STUDY AND RELATED WORK

Figure 1 depicts a simplified cloud infrastructure. As customer  $U$  no longer has full controls over the CSP  $P$ 's hard-

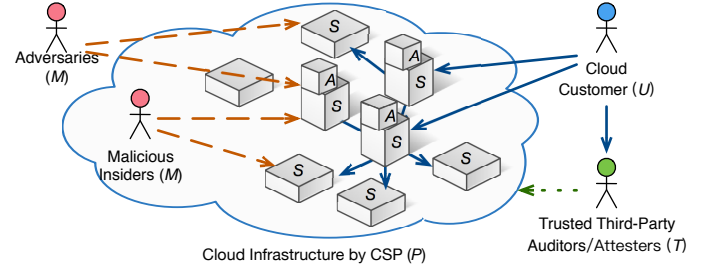


Fig. 1. A Simplified Cloud Scenario. Cloud Customer  $U$  deploys her application  $A$  on CSP  $P$ 's infrastructure.  $P$  enforces cloud services  $S$  to support  $A$ . As both outside adversaries or malicious insiders  $M$  may exist,  $U$  seeks assurance from Trusted Third-Parties  $T$  to vouch for  $P$ 's trustworthiness.

ware infrastructure, she may be concerned with the following threats:

- *Dishonest Providers  $P$* .  $P$  may not load adequate *Cloud Service Component  $S_i$*  to fulfil its SLAs with  $U$ ;
- *Intruders or Malicious Insiders  $M$* .  $M$  may tamper with both  $A$  or  $S_i$  to violate the SLAs or tamper with  $U$ 's data;
- *Faulty Trusted Third Parties  $T$* .  $T$  may not effectively report the above threats promptly.

### A. Audited Clouds

To prove its trustworthiness,  $P$  may invite third-party auditors to evaluate its cloud implementation against various cloud audit schemes. These auditors publish evaluation reports according to  $P$ 's capabilities of fulfilling certain criteria [1], so that  $U$  can make informed decisions on whether to use  $P$ 's cloud services. However, concerns may still arise when  $U$  examines these evaluation certificates. First of all, these evaluations are generally too coarse-grained for making assertive judgements on the satisfactory fulfilment of specific requirements. This is because they usually consider all the services implemented by the CSP, but  $U$  only acquires a small subset. For example, the certificate may identify  $P$ 's implementations of different levels of logging services but it is not clear to  $U$  whether the level she requires is actually assigned to serve her applications.

Secondly, these audits only certify the cloud system's *implementation* status, instead of the *enforcement* status, the later of which is  $U$ 's major concerns. System faults or insider attacks may result in violations of the certified service properties. For example, even if  $U$  is able to determine the employment of a logging service from  $P$ 's audited certificates, she still lacks confidence of the logging service's correct enforcement during the entire life cycle of application  $A$ .

Thirdly, the certificates usually have a relatively long validity period. They only vouch for the cloud's properties at the time the evaluations are performed. As cloud systems are evolving rapidly, with new features being implemented and new vulnerabilities being identified, the acquired cloud services may have very different properties from those at the time of certification. It is hard for  $U$  to judge how the cloud infrastructure has changed since its latest evaluation.

Finally, although most schemes aim to facilitate a transparent cloud system where customers can see though its genuine

properties, the auditing processes are usually opaque.  $U$  still lacks the knowledge for the detailed evaluation process. Consequently, the auditing schemes direct customers' trust towards  $P$  to new authorities: the auditing organizations. Customers could only *blindly* trust the honesty of these organizations, in the hope that the evaluations are free from enforcement faults or political scandals.

### B. Attested Clouds

Trusted Computing technology aims to equip end users with tools to validate the remote software platform's genuine behaviours. It defines the *trustworthiness* of a platform as the capability of recording and reporting the genuine behaviours of all the loaded software components [14], [17]. This capability is implemented by embedding each platform with a secure hardware device, the TPM (Trusted Platform Module) [10], and modifying the platform bootstrapping process to genuinely record the metadata of all the loaded software to the TPM. The TPM records these metadata details in its private storage called the PCRs (Platform Configuration Registers). The *TPM\_Extend* instruction (Eq. 1) is used to produced a chained hash to represent the integrity of a list of vales with a single hash digest. This recording process is generally referred as *measure*, and the resulting recorded values are called the *measurement values*. The *attestation* architecture is then applied to securely report all the measurement values to the remote users. These values help the users to determine the platform's exact software configurations, which in turn imply the platform's behaviours.

$$TPM\_Extend(PCR_{New}) = SHA-1(PCR_{Old} || Input\_Digest) \quad (1)$$

Accordingly, several cloud attestation systems have been proposed. Cloud verifier (CV) [21] 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. Santos et al. [20] proposed a new abstraction to let data be sealed and unsealed only by nodes whose configurations match a predefined policy. Abbadi [11] propose the *combined chain-of-trust*, which builds a single chain-of-trust to attest to a cluster of nodes who have exactly the same configuration. Stefan et al. [12] extend the OpenAttestation to support a Trusted OpenStack infrastructure, and to attest the IMA [19] measurement list of a node.

However, these systems all rely on a central service to attest the entire Trusted Cloud infrastructure. The cloud's immense scale will ultimately make this impractical in the production environment. Moreover, these centralized architectures create a super-power in the cloud ecosystem who is capable of dictating the trustworthiness of any cloud service but is itself very difficult to verify due to its complexity and heavy weight as it accommodates the entire cloud's attestation needs. Therefore, customers have no way to effectively determine whether this trust-establishing mechanism is honest or error-free.

In the scenario depicted in Figure 1, to enforce cloud attestations,  $P$  deploys the centralized attestation delegate to regularly attest every node and examine all the nodes' genuine behaviours.  $U$  then attests these delegates to ensure that they are running as expected. However,  $U$ 's attestation will not examine the detailed configuration status of the cloud services

she is concerned with, such as the logging service. It is also impractical for the  $P$  to allow  $U$  to inspect the attestation delegate's configurations to ensure that the logging service is attested as expected.

### C. Trustworthy and Open Cloud Ecosystem: the Expectations

Accordingly, a trustworthy and open cloud ecosystem discussed in Section I is desired to target the three threats, such that  $U$  will be able to:

- 1) determine the exact cloud services (or malicious software) that have participated in serving her cloud application  $A$ ;
- 2) determine the exact properties of each service (or malicious software) that have been identified;
- 3) choose freely based on her own preference among a wide-range of third-party providers for obtaining the above information.

In the following text, we will explain how the *Separation-of-Powers* model helps  $U$  to achieve these three goals.

## III. MODELS

The core idea of the *Separation-of-Powers* (SoP) model is to distribute the CSP's powers of *definition*, *enforcement* and *inspection* to three independent roles. These roles collaborate to attest to the trustworthiness of cloud services, and restrict each other's behaviours to achieve the *balance-of-powers*. The *SoP* comprises three models: the *Role Model*, which defines three roles to achieve the *separation-of-powers*; the *Collaboration Model*, which specifies how these roles collaborate to achieve cloud service attestations; and the *Restriction Model*, which enforces the mutual restrictions among these roles, as well as among the enforcement parties of a same role.

### A. Role Model

Fig. 2 depicts *SoP*'s Role Model, which contains:

- the *Cloud Services Enforcer* (CSE), which *enforces* cloud services compliance with customers' requirements;
- the *Trust Evidence Reporter* (TER), which *inspects* the behaviours of cloud services enforced by the CSEs;
- the *Software Property Definer* (SPD), which serves as the authority to *define* the properties for each software component constructing the cloud service.

The *CSE* is the reduced-power CSP. Its sole responsibility is to enforce the *cloud services*. Customers reach *Service Level Agreements* (SLAs) with the *CSE*, indicating the *properties* of the cloud services they have purchased, in order to support their *cloud applications*. For example, the services' categories or the levels of quality. The *CSE* constructs each cloud service by layering *software components*, which are the atomic elements for constructing a software platform. Consequently, a cloud service's properties are the aggregation of all its software components' properties. The *CSE* maintains details for how each cloud service is constructed and how it is instructed to support cloud applications, but in the *SoP*, the only way for customers to effectively verify the fulfilment of their SLAs is to consult third parties.

Additionally, the *SoP* identifies two complementary roles to enforce the powers of *inspection* and *definition* respectively:

the *TER* and the *SPD*. Under their assistance, customers will determine: 1) what cloud services have served their applications' requirements; and 2) whether these services possess the adequate properties to fulfil to the SLAs.

The *TER* is effectively a third-party attestation authority. It utilizes Trusted Computing technology to report the trust evidence, which testifies the cloud services' behaviours. This requires the *CSE* to integrate its cloud infrastructure with the Trusted Computing facilities, and to allow the *TER* to install its delegates, the *inspectors*, inside the cloud. The *inspector* collects tamper-proof *trust evidence*, which identifies the cloud services that have participated in serving each cloud application. Each cloud service is represented by an undeniable *digital digest*, which attests to the genuine construction of its software components. Consequently, the *TER* obtains a serial of digest values produced by the trusted hardware, but it cannot interpret them without the information from the *CSE* and the *SPD*.

The *SPD* possesses the knowledge for defining the properties for a piece of software component. In general, it examines the implementations of a cloud service's software components, and certifies each component's properties based on certain evaluation criteria. The *SPD* achieves this by installing its delegates, the *translators*, inside the cloud. The translator maps the cryptographic hash values of a software component's implementation files (binary or script codes) or configuration files to properties. However, without obtaining the information from the *CSE* and *TER*, the *SPD* cannot determine either the software composition of a cloud service, or the set of cloud services that have served a target application.

Consequently, the *Role Model* is designed in such way that each role possesses a piece of critical information for attesting to the trustworthiness of cloud services but cannot independently understand its piece without obtaining critical pieces from the other roles. This minimizes a role's opportunities to tamper with the information without being identified. The *SoP* makes an analogy to the *Executive-Legislature-Judiciary* model in the *Political Philosophy* context.

- The *SPD* acts as the *Legislature*, who “defines what should be done to achieve a certain result”.
- The *CSE* is the *Executive*, who “does what ought to be done to satisfy an agreement”.
- The *TER* serves as the *Judiciary*, who inspects the *CSE* and determines “what has been done”.

The separation of the *TER* and the *SPD* from the original *CSP* enables fair judgements as one cannot effectively justify its own trustworthiness. On the other hand, the *SPD* and the *TER* further separate the duties from a single *Trusted-Third-Party (TTP)* to prevent it from gaining excess powers. This clear separation-of-powers allows the enforcement of effective restrictions among each role, which helps achieving the *balance-of-powers*.

## B. Collaboration Model

Fig. 3 depicts the collaborations among the three roles. To verify the trustworthiness of the cloud services, customers should gather information from each role respectively, including:

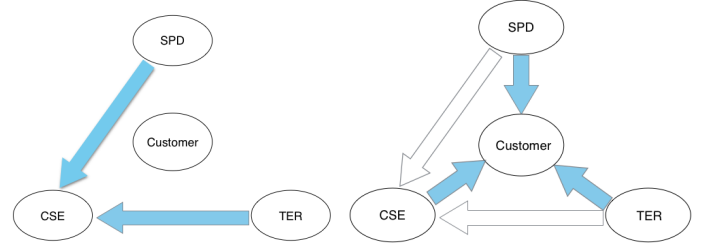


Fig. 2. Role Model. The *CSE*'s power is limited by the *TER* and the *SPD*.

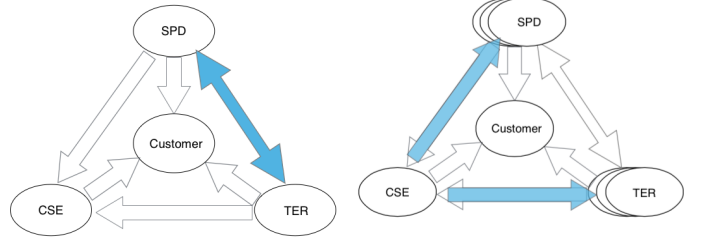


Fig. 3. Collaboration Model. Customers rely on the three roles to make judgement.

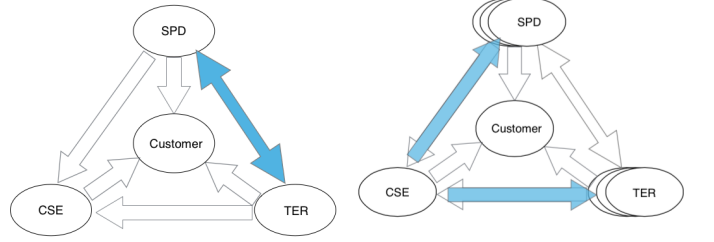


Fig. 4. Restriction Model with Mutual Inspection. The *SPD* and *TER* restrict *SPD* and the *TER* by having competing entities.

- an *Enforcement Digest* (or *Digest* for short) from the *TER*, recording the identities of cloud services loaded by the *CSE* for serving the target application;
- a list of *Service Manifest* (or *Manifest* for short) from the *CSE*, declaring the software composition of each cloud service;
- a list of *Property Definitions* (or *Definitions* for short) from the *SPD*, certifying the properties for each piece of service component.

The *Manifest* ( $M$ ) lists the identifications of the software components ( $C_j$ ) loaded by the *CSE* for constructing a cloud service ( $S$ ). When employing Trusted Computing, these components are identified by their *measurement values*, i.e. the cryptography hash values ( $Hash(C_j)$ ) of their executable codes or configuration files. Meanwhile, to hide the implementation details, a *Transforming Function* ( $Tr$ ) enforced by the *translator* from the *SPD* must be applied to each *Hash*. This makes sure only the specified *SPD* can interpret  $M$ 's entries. For example, when employing the Property-based Attestation schemes [18], [17], [14], these Transforming Functions can be implemented as the *measurement functions* (Sec. IV-C1), which directly translate each *Hash* to a set of *SPD*-defined properties. On the other hand, *SPDs* may simply implement the  $Tr$  as an encryption function, which makes sure that only the designated *SPDs* can decrypt the *Hash*. A *Manifest* is signed by the *CSE*'s key  $K_{CSE}^{-1}$ .

$$M_{SPD}(S) = \{Tr_{SPD}(Hash(C_j))\}_{K_{CSE}^{-1}} \quad (2)$$

The *Digest* ( $Dig$ ) lists the identities of the cloud services ( $S_i$ ) that the *inspectors* perceive to have participated in supporting the customer's application ( $A$ ). It is signed by the *TER*'s key  $K_{TER}^{-1}$ .

$$Dig_{SPD}(A) = \{id(S_i)\}_{K_{TER}^{-1}} \quad (3)$$



A  $S_i$  is effectively identified by the aggregated *measurement value* of all its loaded software components. This value is calculated by the *inspectors* using the Trusted Computing facilities installed on each cloud service: by using the TPM's *TPM\_Extend* instruction (Eq. 1) to iteratively *measure* the list of software components  $C_{ij}$  loaded by  $S_i$ . By using the *measurement values* as the components identities, the *Dig* also records each  $C_{ij}$ 's genuine behaviours.

$$id(S_i) = TPM\_Extend(\{Hash(C_{ij})\}) \quad (4)$$

The *SPD* maintains a database recording the property certificates for each software component ( $C_j$ ). When it receives property querying requests, it returns the *Definition* ticket (*Def*), which binds a list of properties (*Pr*) to  $C_j$ .  $C_j$  is identified by the translated hash value ( $Tr_{SPD}(Hash(C_j))$ ). *Def* is signed by the *SPD*'s key  $K_{SPD}^{-1}$ .

$$Def_{SPD}(C_j) = \langle Tr_{SPD}(Hash(C_j)), Pr \rangle_{K_{SPD}^{-1}} \quad (5)$$

With the *Collaboration Model*, a cloud user ( $U$ ) verifies the trustworthiness for her cloud application's ( $A$ ) supporting cloud services  $S_i$  as follows:

- 1)  $U$  fetches the *Dig*( $A$ ) from the *TER*'s inspectors regarding her application  $A$ ;  $U$  then examines *Dig*( $A$ )'s integrity with the *TER*'s signature;
- 2) For each *Dig*( $A$ )'s entry representing a  $S_i$ 's identity,  $U$  requests the *CSE* for the corresponding *SPD*-translated *Manifest*  $M_{SPD}(S_i)$ ;
- 3) With each  $M_{SPD}(S_i)$ ,  $U$  emulates the *TPM\_Extend* function by iteratively hashing each entry.  $U$  then compares the final value with the  $id(S_i)$ . This verifies whether the  $M_{SPD}(S_i)$  represents the cloud service's genuine behaviours inspected by the *TER*;
- 4) When  $M_{SPD}(S_i)$  is verified,  $U$  interprets each of its entry by requesting the *Def*( $C_j$ ) from the *SPD*. This translates the  $M_{SPD}(S_i)$  to a list of properties  $Pr_j$ ;  $U$  now compares  $Pr_j$  with her SLAs with the *CSE* to verify the satisfaction of her requirements.

The *Collaboration Model* forces the *CSE* to rely on two independent roles, having its behaviours inspected and defined by independent third-parties. Meanwhile, the separations of the *TERs* and the *SPDs* avoid the *CSE*'s trustworthiness from being maliciously arbitrated. In the *SoP*, the *TERs* could only obtain an opaque aggregated hash value, without the knowledge to understand how a cloud application is constructions. On the other hand, the *SPDs* could only be aware of a single software component's properties, without the capability to interpret how a cloud service is assembled. Therefore, without *CSEs*' *Manifests*, the *TERs* and the *SPDs* could not assemble enough information to peek into the cloud's internals. This gives the *CSEs* the opportunities to enforce access controls, and only reveal its properties to the most relevant parties: the customers. Moreover, the *SoP* allows more restrictions to be enforced between the *TERs* and the *SPDs*, so that their genuine behaviours can be examined. This is achieved by the *Restriction Models* explained next.

### C. Restriction Models

To achieve *balance-of-powers*, *Restriction Models* are designed to prevent *TERs* and *SPDs* from gaining excess powers.

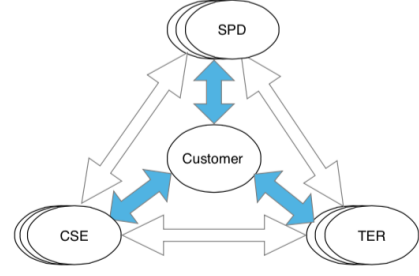


Fig. 6. Freedom-of-Choice. Customers gain the ultimate powers to choose the *CSE-TER-SPD* combinations.

Typically, two threats are considered: *Malfunctions* and *Conspiracy*.

1) *Malfunctions*: *Malfunctions* denote the malicious behaviours or accidental faults of a single role. As either the *TER* or the *SPD* could become faulty or untrustworthy, their *malfunctions* may misinterpret the *CSE*'s genuine behaviours. Moreover, as both the *TER* and the *SPD* install software delegates inside the *CSE*'s infrastructure, their misbehaviours may further facilitate insider attacks. Accordingly, the *SoP* introduces two restriction models: *Mutual-Inspections* and *Multiple-Parties*.

*Mutual-Inspection* is the restrictions the *TER* and the *SPD* enforce on each other (Figure 4). Firstly, the *SPD* defines the properties of the *inspector*. Therefore, the *CSE* is able to verify *inspectors*' behaviours. Secondly, the *CSE* inspects the genuine enforcements of the *translators*. Consequently, every time when an attestation is performed, the measurements of the *translators* are also returned. Therefore, both the *CSE* and the customers will know whether the *translators* are correctly running. However, malicious *TERs* may still manipulate the *Digest* to return incorrect measurement values for their misbehaving *inspectors*. Dishonest *SPDs* may also forge false *Definition* to assign good properties to their malicious *translators*. This is mitigated by introducing the *Multiple-Parties*.

*Multiple-Parties* indicates that a *CSE* is allowed to employ multiple *enforcement parties* (or *parties* for short) for each role (Figure 5). Accordingly, a *CSE* may have multiple *SPDs* to define the properties for its software components. It may also employ multiple *TERs* to inspect its enforced services. This rule introduces competitions and restrictions among the parties from a same role. An *inspector*'s measurement values will be inspected by the one of other *TERs*; and the properties of a *translator* will also be defined by multiple *SPDs*. Moreover, with the assumption that most enforcement parties are willing to produce genuine services, comparing results among the parties of a same role will further help identifying adversaries' hidden dishonesty,

The *Mutual-Inspection* and *Multiple-Parties* break the opaque and omnipotent *Trusted-Third-Parties* (*TTPs*), which are heavily relied on in the current cloud auditing and attestation schemes, into two restricting roles, with each having multiple competing enforcement parties. Therefore, in the *SoP*, each single party's behaviours are constrained as every one is subjected to inspections. We now discuss the *Conspiracy* among maliciously collaborating parties.

2) *Conspiracy*: *Conspiracy* indicates the situation when two parties maliciously collaborate to circumvent the *SoP*. Conspiracy between the *TER* and the *CSE* will allow the *CSE* to allege arbitrary behaviours to meet the SLAs. Conspiracy between the *SPD* and the *CSE* will allow the *CSE*'s behaviours be interpreted as satisfying the SLA, even if they are genuinely recorded by the *TER*. Finally, conspiracy among all three roles will ultimately devolve the model to the original monarchical CSP, who mandates every aspect in the cloud computing ecosystem.

The *SoP* mitigates *Conspiracy* by enabling the *Freedom-of-Choice* from the customers (Figure 6). In *SoP*, customers no longer solely choose a *CSP* for their application, but the combination of *CSE-TER-SPD* is chosen instead. Any enforcement party regarded as less trustworthy will result in the whole combination being discarded. Moreover, the *Multiple-Parties* allows customers to employ multiple *TERs* or *SPDs* to verifying the trustworthiness of a *CSE*. As a result, *CSEs* are driven to employ *TERs* and *SPDs* who are more reputable than others. They are also encouraged to choose multiple *TERs* and *SPDs* to enrich the range of *CSE-TER-SPD* combinations, so that they may attract a wider scope of customers.

#### IV. FRAMEWORK

In this section we present a reference framework for implementing a cloud infrastructure that supports the *SoP* model. We will also introduce the existing research that can serve as the building blocks for this framework. With these building blocks, we present a preliminary implementation design, and illustrate how the *Cloud TCB* of a cloud application is attested. We leave the full implementation and evaluation to our future work.

##### A. Design Principles

We propose three design principles for implementing a *SoP*-compatible cloud infrastructure:

- 1) *Scalable Manifest Management*. The cloud infrastructure should effectively manage the *Manifest* for each of its cloud services. These *Manifests* should reflect the genuine updates to each service's properties, so that their expected changes will not be interpreted as faulty. For example, the *Manifest* should be updated whenever a cloud service's components receive upgrades or patches.
- 2) *Fine-grained Cloud TCB (cTCB) identification*. The cloud infrastructure should support third-party *inspectors* to identify an application's *cTCB* with affordable effort. This *cTCB* should include all the components to fulfil a customer's SLAs, along with all the components that have the potential to behave maliciously. Customers should be able to choose freely among available *TERs* for gathering the their applications' *Digests*.
- 3) *Flexible Property Translations*. The cloud infrastructure should support the third-party *translators* to hide the software components' identities before they are recorded with the Trusted Computing infrastructure. Customers should be able to choose freely among available *SPDs* for the *Definitions* to interpret the concerned properties.

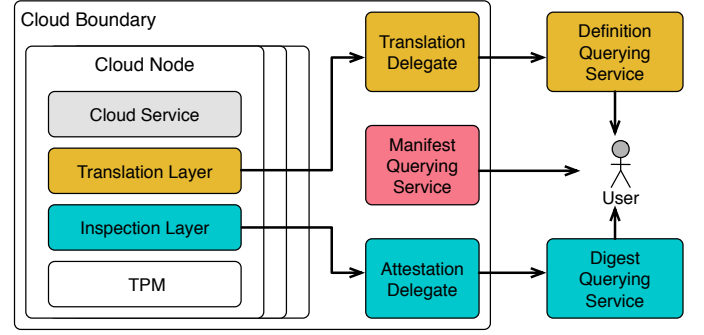


Fig. 7. Reference Implementation Framework

##### B. Reference Framework

In our reference framework (Fig. 7), each cloud node is equipped with a TPM. *CSEs* maintain the *Manifest* for each cloud service. The *Manifest Querying Service* is implemented to return the *Manifest* for interpreting a given *Digest*'s entry. When considering multiple *SPDs*, only the translated version of *Manifest* should be returned. The *Inspector Layer* supports the *inspectors* implemented by *TERs*, and the *Translator Layer* hosts the *translators* from different *SPDs*.

*Translators* translate the measurement values of each software component's executable codes before they are *measured* into the TPMs or vTPMs (virtual TPMs). For example, when the *Compute* service is loaded on an OpenStack cloud node, the hash values of the service's executable, supporting libraries, and configuration files will be calculated. These values are then used to calculate a translated version by each supported *translators*, before they are *extended* into the TPM or vTPM. To assist the translators, the *Translation Proxies* are installed inside the cloud, so that property translation computations are delegated from each node. This reduces the TCB of each node, and eliminates redundant computations. Translation Proxies ultimately report to the *Definition Querying Services* implemented by *SPDs* outside the cloud boundary.

*Inspectors* measure all the upper-layer services, including the *translators*. They also determine the dependency among cloud services hosted on different nodes. This is implemented by inspecting the upper-layer services' interactions and sharing this knowledge with other inspectors. This collaboration helps construct a communication topology, so that the dependency among the cloud services are deduced, along with the *Cloud Trusted Computing Base* for a cloud application [15], [16]. *TERs* deploy the *Attestation Delegate* inside the cloud, which will harvest the information possessed by individual inspectors and construct the *Digest* for a target application. The delegate then reports to the *Digest Querying Service* to provide trust evidence reporting service. Moreover, inspectors also examine the behaviours of its interacting peers, so that malicious ones will be identified. This implements the *Restriction Model*.

##### C. Building Blocks

In this section, we discuss existing research which helps implementing the *inspectors* and *translators*. We will present a preliminary implementation design based on these building blocks in the next section.

1) *Property-based vTPMs*: The Property-based vTPM Framework is presented by Sadeghi et al. in [18]. It supports third-parties *Property Providers* to install *Measurement Functions* in the vTPMs. These functions translate the binary-based measurement values into the provider-defined properties. Each vTPM maintains a set of vPCR (virtual PCRs) arrays, with each array recording the property-based measurement values produced by each *Property Provider*. When a VM loads and measures a software component, it first *translates* the measurement value by enforcing each measurement function. These translated values are then *extended* into the corresponding array of vPCRs.

This framework facilitates the *translator's* implementation, with the *SPDs* instantiated as the *Property Providers* and the *translators* implemented as the *measurement functions*. However, [18] suggests to implement the property translations inside each vTPM. This will result in immense redundancy and computational overheads. It also complicates the vTPM manager's implementation and enlarges the attack surface. We propose *translators* to delegate the actual translation operations to the *Translation Delegate*, which is implemented by the *SPDs* and is deployed inside the cloud. Each *translator* initiates secure protocols to request the *delegate* to provide the property translations before the measurement values are recorded into the vTPMs.

2) *Decentralized Attestations*: In [15], we presented the hierarchical definitions for the *Cloud TCB (cTCB)* for a cloud application. They designed the RepCloud framework, which dynamically identifies the *cTCB* for a target cloud application and gathers the corresponding trust evidence to attest to the *cTCB's* properties. Accordingly, the RepCloud framework can be adapted to support the *inspectors*.

In RepCloud, attestations are performed among cloud nodes based on their interaction needs. Each node attests its communicating peers and stores their collected attestation tickets in its *Local Trust Vector (LTV)*. From this *LTV*, one can examine the properties of the node's logical *neighbors*, including the node's genuine dependency services, as well as potential malicious entities. The *LTV* is further disseminated to relevant nodes based on the nodes' interaction relationship. The received *LTVs* are then aggregated and stored in each node's *Global Trust Matrix (GTM)*. Accordingly, by examining the *GTM* of a node, the dependency of a node's depending nodes can be deduced and attested to. By iterating this procedure, a *web-of-trust* is ultimately built, with which the trust dependency among nodes will be modeled. This model therefore defines the *cTCBs* for the cloud applications. According to our further work in [16], the *web-of-trust* of this kind achieve efficient *cTCB* attestation. It is also effective in identifying misbehaving single entities and is resilient to malicious collaboratives.

Accordingly, each *TER* could install its own *inspector* on each cloud node to implement the RepCloud protocols. However, when multiple *TERs* are supported by a CSE, the complexity and overheads aggregate. Therefore, we propose to have a cloud node only installing the *inspector* from one particular *TER*. All the supported *TERs* install their *inspectors* among different cloud nodes. Consequently, *inspectors* from different *TERs* attest each other, and disseminate their collected trust evidence with the RepCloud protocols. This constructs a

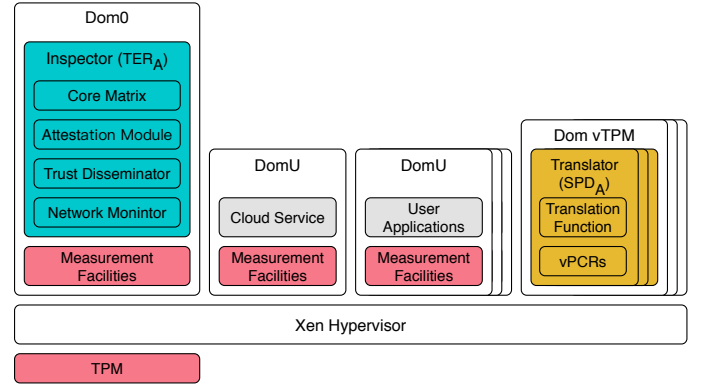


Fig. 8. Preliminary Implementation Design with the XenServer-OpenStack deployment.

single *web-of-trust* to attest to the trustworthiness of all the cloud applications' *cTCBs*.

#### D. Preliminary Implementation Design

Fig. 8 depicts a preliminary implementation design for a *SoP*-compatible cloud system. We leave the complete implementation and evaluations to our future work. The current design is based on the XenServer-OpenStack architecture [4]. All cloud services are deployed in DomUs, the less privileged VMs in the Xen's architecture. This deployment model reduces the cloud services' privilege and facilitates more effective service interaction inspections [16]. Measurement facilities are deployed inside each DomU to measure its *chain-of-trust*. Detailed constructions can be found in [12].

Each VM has an associated vTPM instance running in a separated *Device DomU* [13]. Each vTPM hosts the set of *translators* for the supported *SPDs*. Each *translator* maintains an array of vPCRs and implements a *translation function*. The *translators* interact with the *Translation Delegate* from the same *SPD* to get assistance with the property translation or obscuring. This vTPM can be implemented by extending the TPM emulator [8].

The *inspector* is hosted in Dom0, the Xen's privileged VM, as it needs the privilege to inspect the network interactions among the DomUs [16]. The *Core Matrix* (or *Core*) stores the trust evaluations of the node's neighbors, namely the *LTV* and the *GTM*. It is maintained by the *Attestation Module* and the *Trust Disseminator Module*. The *Network Monitor* intercepts the communications among the DomUs and queries the *Core* for whether the communication target host have recently been attested. *Attestation Module* then attests the neighboring nodes when necessary, and updates the *Core* with the gathered attestation tickets. Afterwards, *Trust Propagation Module* exchanges the updated *Core* with the neighboring nodes by enforcing the RepCloud protocols [15].

When a CSP *P* constructs a cloud service VM *S<sub>i</sub>*, it records a *white-list* for all the software components *C<sub>ij</sub>* that are to be load in order in *S<sub>i</sub>*. This *white-list* becomes *S<sub>i</sub>*'s *Manifest*. When *S<sub>i</sub>* is deployed, a *Dom vTPM* is firstly instantiated and preloaded with the supported *SPDs' translator*. Afterwards, all the loaded *C<sub>ij</sub>* inside *S<sub>i</sub>* will be measured by *S<sub>i</sub>*'s *measurement*

facilities and translated by each *translator*. When  $S_i$  communicates to its collaborating peers, the *inspectors* in the Dom0 initiates the RepCloud protocols to collect and disseminate the attestation tickets for building the *web-of-trust*. These tickets will include *measurement values* for the Xen hypervisor, the Dom0,  $S_i$  and its Dom vTPM. Therefore, the integrity of the *translators* and the *inspector* will be measured. The cloud service DomU and its Dom vTPM will also be measured, as they constitute the cloud node's TCB.

When attesting the cloud services, the customer  $U$  first chooses a *TER* she trusts and consults the *Digest Querying Service (DQS)*. The *DQS* contacts the corresponding *Attestation Delegate* who will in turn interrogate the appropriate *inspectors* to gather enough information for constructing the *web-of-trust*. From this *web-of-trust*, the *DQS* ultimately constructs the *Digest* for the target application's *cTCB*. Obtaining the *Digest*,  $U$  further requests  $P$  to return a *Manifest* for each identified cloud service. For each *Manifest* entry,  $P$  requests the *SPD* for the *Definition* certificate. As the measurement values for the *inspectors* and *translators* are also recorded, their behaviours are evaluated. Moreover,  $U$  may consult more supported *TER-SPD* combinations for different *Digest* and *Definition* versions. This give  $U$  the chance to challenge the authorities of a particular *TER* or *SPD*.

#### E. Discussions and Future Work

Trusted Computing (TC) technology has been much questioned for its scalability and flexibility issues, especially when confronting the cloud's complexity and dynamism. Effectively identifying and attesting the *Cloud TCB* for a target cloud application are also the core challenges for implementing a *SoP*-compatible cloud system. This paper builds on our previous research [15], [16], [11], but we admit much more work is required. In future work, we will build a complete prototype and evaluate the *SoP*'s effectiveness and practicality. Trusted Computing is also incapable of defeating memory injection attacks or physical attacks. But it provides a trusted foundation to facilitate more mechanisms that target these threats. We will also investigate these issues in our future work. There are other obstacles towards the widest cloud adoption that cannot be solved merely by the *SoP*. But we see the *SoP* as a facilitator that helps improve the effectiveness of other methods. For example, the properties like performance, availability and data reliability are often part of SLAs, but are not aligned well with the remote attestations. However, the *SoP* helps assure the correctness and genuine enforcement of the software systems that inspect these properties. Also, the economies-of-scale is another factor limiting the growth of new CSPs, apart from the lack-of-trust. But the *SoP* gives small CSPs more opportunities to earn customers and grow.

#### V. CONCLUSION

Security concerns with the cloud have discouraged a wide range of potential cloud user and hence application models. In this paper, we identified that trust issues are one of the root causes, due to the CSPs' opaque and autocratic governance. Various cloud auditing and cloud attestation schemes aim at facilitating the cloud's transparency, but they rely heavily on *Trusted Third Parties (TTPs)* to serve as the ultimate *trust definer*. However, in reality, these TTPs are hard to

establish - their own trustworthiness and correctness also need inspections. We take on the issue by separating the CSPs' powers and eliminating the TTPs default trustworthiness: we designed the Separation-of-Powers (*SoP*) model to enforce the collaborative-restriction dynamics among three independent roles. *SoP* encourages fair competition and hence will facilitate the establishment of a fair market, as any party is able to gain equal trust from customers, regardless of its scale or established credibility. With this establishment, more service providers with diverse capabilities could participate in the cloud ecosystem. We will investigate this direction in our future work.

#### REFERENCES

- [1] Cloud security alliance. <http://www.cloudsecurityalliance.org>.
- [2] Cloudaudit. <http://cloudataudit.org/CloudAudit/Home.html>.
- [3] Cobit. <https://cobitonline.isaca.org/>.
- [4] Getting started with XenServer and OpenStack. <https://wiki.openstack.org/wiki/XenServer>.
- [5] Greg papadopoulos's comments. [http://news.cnet.com/2008-1011\\\_-3-6141598.html](http://news.cnet.com/2008-1011\_-3-6141598.html).
- [6] Hippa. <http://www.hhs.gov/ocr/privacy/>.
- [7] Iso27002. <http://www.27000.org/iso-27002.htm>.
- [8] Libtpms-based tpm emulator. <https://github.com/stefanberger/swtpm>.
- [9] Trusted Computing Group. <http://www.trustedcomputinggroup.org>.
- [10] Trusted platform module main specification. [www.trustedcomputinggroup.org/resources/tpm\\_main\\_specification](http://www.trustedcomputinggroup.org/resources/tpm_main_specification).
- [11] I. Abbadi and A. Ruan. Towards trustworthy resource scheduling in clouds. *Information Forensics and Security, IEEE Transactions on*, 8(6):973–984, June 2013.
- [12] S. Berger, K. Goldman, D. Pendarakis, D. Safford, E. Valdez, and M. Zohar. Scalable attestation: A step toward secure and trusted clouds. *Cloud Computing, IEEE*, 2(5):10–18, Sept 2015.
- [13] J. Cucurull and S. Gausch. Virtual tpm for a secure cloud: fallacy or reality? In *Proceedings of the 13th Spanish Meeting on Cryptology and Information Security*, pages 197–202. RECSI Press, 2014.
- [14] P. Jonathan, S. Matthias, H. Els, Van, and W. Michael. Property attestation – scalable and privacy-friendly security assessment of peer computers. In *Technical Report RZ 3548*. IBM Research, 2004.
- [15] A. Ruan and A. Martin. Repcloud: achieving fine-grained cloud TCB attestation with reputation systems. In *Proceedings of the sixth ACM workshop on Scalable trusted computing*, STC '11, pages 3–14, New York, NY, USA, 2011. ACM.
- [16] A. Ruan and A. Martin. Neuronvisor: Defining a fine-grained cloud root-of-trust. In *Proceedings of the sixth International Conference on Trustworthy Systems*, InTrust '14, 2014.
- [17] A.-R. Sadeghi and C. Stübke. Property-based attestation for computing platforms: caring about properties, not mechanisms. In *Proceedings of the 2004 workshop on New security paradigms*, NSPW '04, pages 67–77, New York, NY, USA, 2004. ACM.
- [18] A.-R. Sadeghi, C. Stübke, and M. Winandy. Property-based TPM virtualization. In *Proceedings of the 11th international conference on Information Security*, ISC '08, pages 1–16, Berlin, Heidelberg, 2008. Springer-Verlag.
- [19] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 16–16, Berkeley, CA, USA, 2004. USENIX Association.
- [20] N. Santos, R. Rodrigues, K. P. Gummadi, and S. Saroiu. Policy-sealed data: a new abstraction for building trusted cloud services. In *Proceedings of the 21st USENIX conference on Security symposium*, Security'12, Berkeley, CA, USA, 2012. USENIX Association.
- [21] J. Schiffman, Y. Sun, H. Vijayakumar, and T. Jaeger. Cloud verifier: Verifiable auditing service for IaaS clouds. In *Services (SERVICES), 2013 IEEE Ninth World Congress on*, pages 239–246, June 2013.