Taxonomy of Trust Relationships in Authorization Domains for Cloud Computing

Juan M. Marin Perez[a], Jorge Bernal Bernabe[a], Jose M. Alcaraz Calero[b,*], Felix J. Garcia Clemente[c], Gregorio Martinez Perez[a], Antonio F. Gomez Skarmeta[a]

[a]*Departamento de Ingenieria de la Informacion y las Comunicaciones, University of Murcia, Spain*
[b]*School of Computing, University of the West of Scotland, Glasgow, Scotland*
[c]*Departamento de Ingenieria y Tecnologia de Computadores, University of Murcia, Spain*

Abstract

Cloud computing is revealing a new scenario where different cloud customers need to collaborate to meet client demands. The cloud stack must be able to support this situation by enabling collaborative agreements between cloud customers. However, these collaborations entail new security risks since participating entities should trust each other to share a set of resources. The management of trust relationships in the cloud is gaining importance as a key element to establish a secure environment where entities are given full control in the definition of which particular services or resources they are willing to share. Entities can cooperate at different levels of trust, according to their willingness of sharing information. This paper analyses these collaboration agreements defining a taxonomy of different levels of trust relationships among customers for the cloud. Privacy concerns, assumed risk, as well as easiness in the definition of the trust relationships have been taken into account. A set of different trust relationships have been identified and modeled, enabling entities to control the information they share with others in the cloud. The proposed model has been validated with a prototypical implementation. Likewise, some examples to illustrate the application of these trust models to common cloud collaboration scenarios are provided.

Keywords: cloud computing, trust management, access control, taxonomy

1. Introduction

Cloud computing is rapidly evolving to respond to new demands in IT-based enterprises. Cloud providers offer virtual infrastructure-as-a-service (IaaS) platforms, while cloud consumers rent such infrastructures on-demand, according to their needs. This scenario entails a multi-tenancy environment in which different cloud consumers are using and sharing the same infrastructure offered by the cloud provider. Originally, cloud providers focus on isolated scenarios by providing different control mechanisms to enable a security boundary that isolates consumers from each

*Corresponding author

Email addresses: juanmanuel@um.es (Juan M. Marin Perez), jorgebernal@um.es (Jorge Bernal Bernabe), contact@jmalcaraz.com (Jose M. Alcaraz Calero), fgarcia@um.es (Felix J. Garcia Clemente), gregorio@um.es (Gregorio Martinez Perez), skarmeta@um.es (Antonio F. Gomez Skarmeta)

other. However, in such a multi-tenancy environment, collaboration agreements between different consumers may also arise. Different organizations running their services in the cloud may need to establish new business collaboration agreements to enhance their business capabilities and cope with user demands. This lead to a federated scenario in which advanced inter-organizational collaborations may be established between tenants, and where resource control should be shared between the involving parties. For instance, let us assume two organizations offering their services in the cloud reach a business agreement to offer a common new service that requires the usage of their current services. Both partners are different cloud tenants that use their corresponding cloud computational resources to run their part of the common service. In case one of them runs out of enough compute resources, it can use the other's compute resources to scale-up and continue providing the common service. This way, the organizations take advantage of this agreement to share cloud resources avoiding the need to hire more cloud resources for the common service.

These collaboration agreements should be properly managed by defining the terms in which one customer trusts another. Each tenant should trust its collaborating partners as they have to vouch for the shared resources. This situation can lead into new security risks. Hence, the management of trust relationships between tenants is gaining importance as an essential part of the ongoing efforts to consolidate cloud computing architectures.

The term *Trust Management* is referred to the ability of managing, establishing, validating, and enforcing these secure trust relationships between entities. In this paper *trust relationship* refers to the conditions under which a tenant can specify elements of the domain of the other tenant to the define its authorization rules. This trust relationship will drive the interaction of the two tenants based on the security mechanisms that control the access to cloud resources. A trust relationship should not be confused with authorization rules. They do not govern the access to the actual resources of the tenant in the CSP, but they control the expressiveness of the authorization rules that can be defined by the administrator. Following the above example, a trust relationship like the ones described in this paper can be defined to govern the way this interaction should be performed. This trust relationship will allow the tenant without enough compute resources to specify authorization rules for its users to be able to access the resources of the other tenant to scale-up the common service.

The establishment of collaboration agreements between two tenants is a tradeoff between isolation and collaboration. It directly depends on the level of security and privacy that needs to be ensured. For instance, specifying a trust relationship which enables a tenant to define authorization rules allowing other tenant's users to access its resources may imply some loss of privacy. This would also enable the tenant to get knowledge of the other tenant's users. Each level of security and privacy possible may lead to the definition of a new kind of trust relationship. Current solutions rarely cover the possibility to deal with trust relationships or they just enable a single kind of trust relationship between tenants. This fact leads to a rigid scenario with a clear lack of flexibility that cannot cover the vast number of collaboration agreements in the cloud. There is a clear need to determine and define the different security levels available in cloud computing, and map them to different kinds of trust relationships, creating a complete taxonomy of trust relationships. We have analyzed a wide number of public and private cloud computing stacks like Eucalyptus, OpenStack, OpenNebula, CloudStack, HP Cloud, Amazon EC2 and GoGrid among others and to the best of our knowledge, there is not any available trust management system for cloud to enable

the specification of more than one type of trust relationship to manage which particular actions, resources, services and other enterprise information can be shared with other tenants and for which purpose. This lacks in current cloud computing stacks and contradicts a clear demand by cloud users. In this context, it is imperative to provide a set of tools to enable tenants to define fine-grain security and privacy contexts over which they can establish a reliable collaboration in a secure environment. This management of trust relationships between tenants is gaining importance as an essential part of the ongoing efforts to consolidate cloud computing architectures. This is the main rationale of this research work.

The main contribution of this paper lies on the trust management in cloud environments, analyzing different trust relationships between tenants. The inclusion of trust relationships enables collaborations between tenants, resulting in a multi-domain system to manage such collaborations. They regulate the extent to which different tenants can interact with others. Eventually, these trust relationships will condition the underlying authorization system, since it is this system the one that ultimately controls the access to the resources. It is important to remark that our contribution does not provide a novel trust model to calculate whether or not tenants are trusted between them. It is assumed that there is a static or dynamic approach already available to define or calculate those trust values. Instead, this paper proposes a novel taxonomy of trust relationships where existing models for calculating trust can be applied to enable real collaboration and resource sharing in the cloud. The proposed approach has been validated by a prototypical implementation for OpenStack [1], a well-known cloud computing stack. We also illustrate and discuss the identified kinds of trust relationships. Some common application scenarios are provided, identifying what particular trust relationship is more suitable for each one.

The remainder of this paper is organized as follows. Section 2 overviews related work about different kinds of collaboration and trust models for cloud computing. Section 3 describes the system and authorization models adopted in this proposal with the aim of managing the access control for cloud-based federated scenarios. Different kinds of trust relationships are presented in Section 4. Afterwards, Section 5 describes the proposed authorization architecture with trust management support for cloud Computing . Section 6 illustrates how the trust management approach described in this paper can be implemented. Then, Section 7 provides different scenarios and examples in cloud environments, where different trust relationships can be applied. Section 8 discusses about the suitability of the trust relationships and their implications in terms of security and privacy. Finally, Section 9 provides some concluding remarks.

2. Related Work

In grid computing, Vijayakumar et al. [2] describe a good example of dynamic trust model in which resource selection is done according to trust relationships. The trust factor value of each entity is determined from the self-protection capability and the reputation weight of that particular entity. Jint et al. [3] also present an access control model for Grid based on trustworthiness. They provide an algorithm to compute the trustworthiness of a certain user accessing to a resource. Regarding cloud computing, Kumar et al. [4] define a a trust management architecture which serves as registry for trust values, a trust calculator calculates trust based on feedbacks. Wang et al. [5] define a quantitative evaluation method to determine the reputation of trust objects using uncer-

tainty. Abawajy et al. [6] focus on the problem of how to determine service trustworthiness in inter-cloud computing environments. They design an inter-cloud resource manager responsible for mediating in the resource exchange between peering clouds. They also provide a Reputation Manager in charge of collecting, calculating and maintaining certain trust measurement of peering clouds. Boursas et al. [7], however, compute trust values based on past experiences and reputation in federated environments like cloud or grid. They show the importance of aggregating, correlating, and refining the trust information to deal with dynamic trust management. They also define a trust based access control (TBAC) framework whereby implement the trust model. Hu et al. [8] use service provider, service requester and service broker information to evaluate the trust value at the SaaS layer of the cloud. They, use D-S (Dempster-Shafer) evident theory to fuse the tripartite trustworthiness.

All the above proposals focus on quantifying trust in order to determine whether to accept or deny collaborations with an unknown entity in distributed systems. In the authorization context, collaboration is specifically focused on accepting or denying the access to a protected resource. The calculated trust value determines if a user can or cannot access to a resource belonging to a different entity. The trust value is sometimes a Boolean value (yes or no), a set of trust values e.g. (unknown, no trust, marginally trusted, fully trusted), or a numeric value. But, in the end, it is used to determine if the access to the resource is granted or not (i.e., a boolean decision). In some cases, the authorization is established if the calculated trust value is bigger than a given trust threshold. In other cases [9] [10] [11] [12], protected resources have an associated access value and only those entities with a trust value greater than such a value are granted to access. Sometimes, actions defined over protected resources have an associated access value, and only those entities with a trust value greater than such a value can execute the action over the resource. All the aforementioned cases share a common fact: they only define one type of access control trust relationship, in which a subject belonging to a given entity tries to access to a resource belonging to another entity. This is only one of the trust relationship types proposed in our contribution. Recently Fakhar and Shibli [13] has provided a comparative analysis on security mechanisms in cloud computing in which they cover authentication, trust management and private and also validated this statement.

An advanced complex trust-based access control model is proposed by Yang et al. [14]. Authors extend the classic role-based access control (RBAC) with trust level requirements, which dictate the trust prerequisites for role activation under the privilege context. Fujun et al. [15] also extend RBAC with trust-based capabilities. However, authors do not cover multiple administrative domains and thus it does not fit on cloud computing. Xiong et al. [16] provide a trust-based access control model for Grid computing that dynamically grants privileges to subjects based on context and trust information. Xiao-jun et al. [17] provide a similar approach, but this time activating roles are based on calculated trust values. Note that dynamic granting of privileges does not fit in cloud computing scenarios, in which the customer's virtual infrastructure is managed and authorization information has to be clearly defined. Recently, Ngo et al [18] provide a access control service for virtualized cloud infrastructures with trust management. However this trust manager module is intended to ensure trusting computing and not for enabling federations between tenants.

Our paper goes a step forward on defining and identifying different kinds of trust relationships in order to establish a federation. The implications in terms of what an entity is able to access from

another one and for what purposes are clearly identified. Our contribution complements all the above research works in order to provide a more detailed vocabulary to define trust relationships in cloud computing. Concretely, static trust approaches can make use of the trust relationships presented herein to extend the scope of their trust relationships, whereas dynamic trust approaches can map their trust values to different trust relationships, rather than use mere Boolean decisions.

In case the reader is interested in computational trust, in [19] a complete related work about trust and reputation models in distributed systems can be found. Moreover, the trust-based access control models described above could also be extended in order to tackle the proposed ways of sharing authorization information. The previous survey is also complemented by Viriyasitavat et al. [20], which give a complete survey of different kinds of trust models about service-based interactions. It covers different areas such as e-commerce, content providers, virtual organizations, peer-to-peer, web services, grids, cloud Computing and even individual interactions. It proposes a classification that identifies six main types of trust: trust in content or information provisions, trust in content and information retrieval, external and internal service provision trust, trust in providing services, and trust in workflow Participants. They put special attention on describing security applied to service workflows. The trust relationship definitions provided in our contribution are more generic and can be used for any of those types, as long as they are applied to multi-tenant cloud environments.

Note that there exist a clear lack of trust management architectures for cloud environments that identify, analyze, and provide the definition of trust relationships that enables tenants to establish collaboration in order to provide federated capabilities.

3. System and Authorization Models

The trust relationships established between tenants are directly related to the policies that a tenant defines to control the access to resources. However, it is important to differentiate between the access control policies defined by a tenant to govern access privilege over a set of (shared) resources and the trust relationships defined by a tenant which allows defining the resources that other tenant can use in the definition in his access control policies. The usage of trust relationships enables the creation of a dynamic adaptive authorization system in which the expressiveness of the access control policies is changing according to the trust relationships defined by the tenant.

Thus, it is necessary for the reader to know some concepts regarding the models which are used as basis for the access control. Namely, the models used to describe the tenant information system and the models used for defining access control policies. An authorization model establishes the privileges of a subject to perform a given action over a given target resource. This model is composed by a set of statements that can be represented by a 3-tuple *(Subject, Target, Privilege)*. When a subject performs an action over a cloud resource, the authorization system cheeks the authorization database looking for authorization statements that grant the requested action (Privilege) over the resource (Target).

The cloud computing environment is characterized and differentiated by its multi-tenancy capabilities, where different tenants simultaneously share the cloud provider infrastructure. An authorization model used in these environments should take the multi-tenancy property into account. To cope with this issue, the 3-tuple is extended to a 4-tuple *(Issuer, Subject, Target, Privilege)*. This

4-tuple is interpreted as "the *issuer* states that the *subject* has the *privilege* to perform a given action over the *target*". For example, the tuple *(TenantA, TenantA:Alice, TenantA:volume4.CloudStorage, volumeCreate)* is interpreted as "*TenantA* states that *Alice* -which is a user of *TenantA*- has privileges to perform the action *volumeCreate* over *volume4* associated to the *CloudStorage* service -which is also owned by *TenantA*". The usage of tenant IDs as namespaces allows for determining the administrative domain in which the Subject and Targets are defined. These namespaces are especially relevant for our trust management system. It is also worth mentioning that the *Privilege* element of the statement is not scoped by a *Namespace*. The reason is that privileges usually refer to actions which can be performed over the infrastructure which is provided by the cloud provider. Thus, the set of actions which can be used in authorization statements is shared among all the tenants. In the most basic scenario, in which there is not any defined trust relationship, *TenantA* is only allowed to define authorization statements affecting its own subjects, privileges and targets, i.e. tenant A only can use its own namespace *TenantA* to define authorization statements.

Role Based Access Control (RBAC) is an authorization schema supported by most of the current authorization solutions. In this approach, the authorization model relies on the usage of roles which are associated to a set of privileges. Subjects can belong to one or more roles. All the users belonging to a role acquire the privileges associated to that role. To support this model, the 4-tuple used for authorization is therefore represented as *(Issuer, [Subject | Role], Target,Privilege)*.

Additionally, this authorization model can be extended to hierarchical RBAC (hRBAC) or conditional RBAC (cRBAC). Hierarchical RBAC enables the definition of role hierarchies, making a child role to inherit all the privileges defined for its parent in the hierarchy. Conditional RBAC (cRBAC) allows to define a set of conditions for an authorization statement, specifying the conditions under which the permissions apply. This enables the inclusion of context information in authorization statements, such as temporal and spatial restrictions. Supporting the inclusion of this context information requires the extension of the authorization statement to a 5-tuple *(Issuer, Conditions, [Subject | Role] ,Target, Privilege)*.

Thus, including the namespaces, we can sum up the authorization model to be represented by the 5-tuple *(Issuer, Tenant:Conditions, [Tenant:Subject | Tenant:Role] ,Tenant:Target, Privilege)*. The authorization statements defined by this 5-tuple can be formally represented by means of an authorization rule, where the antecedent of the rule represents the conditions to be fulfilled and the consequent gives the grants for the subjects to access the resources. It is assumed that the authorization system will deny by default any action over any resource, unless explicitly stated by an authorization rule.

$$Issuer : \{Namespace : Conditions\} \rightarrow \tag{1}$$

$$(\{Namespace : Subject\}|\{Namespace : Role\}, \{Privilege\}, \{Namespace : Target\}) \tag{2}$$

Definition 1: Generic authorization rule

The rule in Definition 1 represents a generic rule that we use in our implementation to enforce the aforementioned authorization capabilities. This generic authorization rule has served to enable

the tenant to define the access control policies. Note that in authorization systems without trust management capabilities, the set of namespaces used in this rule is kept fixed according to a pre-establish semantic in the system, usually limited to only the namespace of the own tenant. The trust relationships proposed in this contribution enable a tenant A to allow another tenant B to use different the namespace of (A:) -apart from its own one (B:) for one or more of the elements which appear in this authorization rule (conditions, subject, role, privilege or target). Note the importance of enabling the usage of a different namespace which entails pointing to resources owned by another tenant, thus creating a federated environment. Controlling the use of these namespaces for the definition of authorization statements is one of the main security issues directly related to trust management. This namespace usage is determined by the own tenant creating a user-centric authorization model by means of *trust statements* . Note the difference between *authorization statements* represented by the 5-tuple explained above, and *trust statements* defined to control namespace usage in such authorization statements.

There are different approaches to implement the described authorization model. We have validated the above authorization model in the past by providing an implementation based on semantic-web technologies, i.e. OWL ontologies and SWRL rules [21]. We also proposed a similar authorization model implementation by using a database-based approach [22]. It is important to note that the trust relationships and trust management system presented in this proposal are not bounded to any specific authorization model or cloud computing. Nevertheless, to evidence the feasibility of the proposal, we have made a proof of concept in a specific layer, infrastructure-as-a-service, using OpenStack, a well-known cloud computing IaaS system.

4. Trust Relationships

Trust relationships allow describing and managing collaboration agreements between cloud tenants. Our proposal defines a taxonomy of kinds of trust relationships to deal with different customer needs. These trust relationships are used to ensure a proper level of security and privacy among the parties. A collaboration agreement between two tenants A and B implies the definition of a trust relationship between them. A trust relationship is an asymmetric relationship. That is, A trusts B does not imply B trusts A or vice versa unless the latter fact is explicitly indicated. Moreover, for the sake of simplicity, a trust relationship is also not transitive. Thus, A trusts B and B trusts C does not imply that A trusts C unless this is explicitly declared. We decided to foster the explicit appearance of trust relationships rather than inferring information by these semantics in order to avoid any undesired implicit and non-desired relationship.

In this paper, a trust relationship between two tenants A and B, in which tenant A trusts tenant B is defined using the following notation: $A \prec B$. When such a trust relationship is defined, it formally implies that tenant A allows tenant B to define access control policies using certain information of tenant A. Therefore, a trust relationship can be defined as a 3-tuple indicating that a *Trustor* (tenant A) trusts a *Trustee* (tenant B) for sharing certain *TrustedInfo* information. This 3-tuple can be seen in Definition 2.

Once the trust relationship is established, the *TrustedInfo* of the *Trustor* can be used by the *Trustee* with the aim defining its own authorization statement. This means that the *Trustee* can use the information defined in *TrustedInfo* to manage the access control within its own cloud in-

$$A \prec B = (Trustor_A, Trustee_B, \{TrustedInfo\}) \quad (1)$$

Definition 2: Generic trust relationship definition

frastructure. Notice that the *TrustedInfo* element in the tuple refers to any instantiation of the cloud computing system information model adopted by the cloud provider (VMs, Networks, StorageDisk, ...). Thus, the *Trustee* is able to use such information originally owned by the *Trustor* in its own authorization statements.

One of the main differentiating points of our proposal with respect to current state-of-the-art is that the *Trustee* can use the *Trustor*'s *TrustedInfo* on its authorization statements for different purposes. Each purpose lead to a different trust relationship which is the key differentiating point of our proposal. Following subsections will expose different kinds of them. As exposed in previous section, trust relationships extend the namespaces that a *Trustee* can use when defining its access control policies.

The different Trust Relationships identified in this paper can be grouped in different kinds according to the level of granularity in which the *TrustedInfo* can be expressed in the 3-tuple. We propose four groups or families of trust relationships, according to the level of granularity provided. Namely, *Universal*, *Existential*, *Typed* and *Fine-Grain* trust relationships. These groups are ordered from the lowest to the highest fine-grain detail on the definition of the *TrustedInfo* which is shared between tenants. It should be noticed that these four groups or families do not refer to the kind of information that can be shared between the tenants, but they refer to the level of control offered by those trust relationships to manage the authorization information that is shared.

To clearly differentiate the trust relationships, we use the following notation: $A \prec^{G}_{N_1,N_2,..N_m} B$. This notation determines different trust operators referred to trust relationships. The G superscript identifies the group or family of trust relationships. In turn, the $N_1, N_2, ..N_m$ subindexes identify the parts of the authorization statement in which the *Trustee* is allowed to use *Trustor's* information, specifying its namespace. These subindexes can take the values C, S, R, T to indicate respectively conditions, subjects, roles and targets elements. The different kinds of trust relationships are also numbered for referencing purposes.

4.1. Universal Trust Relationships

The group of *universal* trust relationships define the $TrustedInfo$ literally as the complete information model of the *Trustor*. When a universal trust relationship is defined, all the system information model of the *Trustor* can be used by the *Trustee* to define its authorization statements. However, the *Trustor* can determine how this information should be used in these statements.

In a first kind of trust relationship, the *Trustor* allows the *Trustee* to use its information only as context information, i.e. to define the conditions for its authorization statements. The usage of conditions in authorization statements does not imply granting any privilege to anyone. In other words, using this information as conditions for authorization statements do not compromise the *Trustor* security - although it obviously affects privacy. Taking into account the *Trustor*'s system

information as context information for the *Trustee*'s authorization statements can be useful in cloud environments, while implying a minimal risk for the *Trustor*. For example, a *Trustee* can describe an authorization statement indicating that some of its VMs could be started only in the case that a particular *Trustor*'s VM is not running (e.g. for failure tolerance purposes). It is worth noticing that, although this kind of trust relationship implies a minimal risk for the *Trustor*, it still involves a privacy risk. It may enable the *Trustee* to collect information from the *Trustor*'s system, since it is allowed to define conditions based on its information. When these conditions become true, they can give the *Trustee* some knowledge about the status of the *Trustor* system.

According to the notation defined before, this first trust relationship is denoted as $A \prec^{U}_{C} B$. The C subscript in the trust operator denotes that the trust relationship enables the *Trustee* to use the $TrustedInfo$ only as part of the Conditions in its authorization statements. The superscript U in the trust operator stands for *Universal*, and indicates that the *TrustedInfo* contains all the elements of the *Trustor's* system model. Formally, the 3-tuple used for the definition of this trust relationship implies the extension of the namespaces in the condition field of authorization statements. This can be seen in Definition 3. The left side of the arrow indicates the definition of the 3-tuple trust relationship, in which A indicates that it trusts B and the TrustInfo shared is all the "system elements" of tenant A. The right side of the arrow indicates the implication of this trust relationship with respect to the 5-tuples that tenant B can define in his authorization system. Thus, B can define conditions of authorization statements scoped not only in his own namespace, but also using the namespace A. *AnyCondition*, *AnySubject*, *AnyRole* and *AnyTarget* refer to any instances available in the information system belonging to the associated namespace.

$$A \prec^{U}_{C} B = (A, B, A : SystemElements) \Rightarrow \quad (1)$$

$$(B, [B : AnyCondition | A : AnyCondition], [B : AnySubject | B : AnyRole], B : AnyTarget, AnyPrivilege) \quad (2)$$

Definition 3: Universal Trust Operator for Conditions

A second kind of trust relationship is that in which the *TrustedInfo* can be used not only as Conditions, but also as Subjects of the authorization statement. This would enable the *Trustee* to grant *Trustor* users access to its own resources. In this case, the trust operator is denoted as $A \prec^{U}_{C,S} B$, where C refers to Conditions and S refers to Subjects. It indicates that the *Trustee* can use the whole system information of the *Trustor* as Conditions and it can also use the *Trustor's* users and roles as Subjects in its authorization statements. Definition 4 shows the 3-tuple and the namespace extension associated to this trust relationship.

This second kind of trust relationship also do not imply serious security concerns since you are just enabling others to grant privileges to your users over others' infrastructures. However, there is an important security concern because the *Trustee* could try to discover all the userIDs of the users available in your system. To cope with this privacy concern, we can decide to define a third kind of trust relationship in which the grants are defined based on roles rather then subject increasing the privacy of the tenant reducing the amount of exposed data. This is the trust relationship defined as $A \prec^{U}_{C,R} B$,

$$A \prec^{U}_{C,S} B = (A, B, A : SystemElements) \Rightarrow \quad (1)$$
$$(B, [B : AnyCondition | A : AnyCondition], [B : AnySubject | A : AnySubject | B : AnyRole], \quad (2)$$
$$B : AnyTarget, AnyPrivilege) \quad (3)$$

Definition 4: Universal Trust Operator for Conditions and Subjects

These previous trust relationships cannot be enough for certain collaborations which may require a closer trust. It may happen that tenants could desire to establish a collaboration where the *Trustor* wants to share its resources with the *Trustee*, being this able to grant its users the access to the resources. This trust relationship is achieved by enabling the *Trustee* to specify *Trustor* resources in the *Target* part of its authorization statements. Definition 5 shows the 3-tuple and the Trustee namespace extension associated to this trust relationship.

$$A \prec^{U}_{C,S,T} B = (A, B, A : SystemElements) \Rightarrow \quad (1)$$
$$(B, [B : AnyCondition | A : AnyCondition], [B : AnySubject | A : AnySubject | B : AnyRole | A : AnyRole], \quad (2)$$
$$[B : AnyTarget | A : AnyTarget], AnyPrivilege) \quad (3)$$

Definition 5: Universal Trust Operator for Conditions, Subjects and Targets

The trust operator $\prec^{U}_{C,S,T}$ is quite risky from the *Trustor* security perspective, since it gives the *Trustee* total control to manage the *Trustor* information system. This trust relationship enables the *Trustor* to completely delegate the access control management to the *Trustee*. Therefore, it must be carefully used for establishing trust relationships in the cloud.

Table 1 summarizes the trust relationships described so far and some other similar ones directly derived from them. As can been seen, there are fifteen different universal trust relationships. They correspond to the fields of the authorization statement for which the namespace is extended, and their possible combinations. It has been analyzed all these combinations in order to identify if there is any combination which makes no sense. Note that the previous described relationships have been justified and they represents different levels of security and trust. The analysis lies in the fact that the rest of combinations are modifications from the previous ones to keep the same security level but increasing the privacy of the exposed data, thus all of them make sense in real scenarios.

Trust relationships arranged in Table 1 are sorted according to the associated risk for the *Trustor*, from the lowest (ID:1) to the highest one (ID:15). For the sake of simplicity, only trust relationships 1, 5 , 6 and 14 have been fully described in the above text. Nevertheless, the semantics of the other three trust relationships is similar and can be easily deduced from the notation.

When a trust relationship enables the usage of Subject or Target elements in authorization statements, it implies that these particular targets or subjects can also be used as Conditions. Once

Id	Trust Relationship	Condition	Subject	Role	Target
1	$A \prec^{U}_{C} B$	X			
2	$A \prec^{U}_{R} B$			X	
3	$A \prec^{U}_{S} B$		X		
4	$A \prec^{U}_{S,R} B$		X	X	
5	$A \prec^{U}_{C,R} B$	X		X	
6	$A \prec^{U}_{C,S} B$	X	X		
7	$A \prec^{U}_{C,S,R} B$	X	X	X	
8	$A \prec^{U}_{T} B$				X
9	$A \prec^{U}_{C,T} B$	X			X
10	$A \prec^{U}_{R,T} B$			X	X
11	$A \prec^{U}_{S,T} B$		X		X
12	$A \prec^{U}_{S,R,T} B$		X	X	X
13	$A \prec^{U}_{C,R,T} B$	X		X	X
14	$A \prec^{U}_{C,S,T} B$	X	X		X
15	$A \prec^{U}_{C,S,R,T} B$	X	X	X	X

Table 1: Summary of Universal Trust Relationships

the *Trustee* is able to access to those targets or subjects to give grants to them, it is meaningless to avoid using that information as Conditions. However, it does not mean that the capability of enabling or disabling the usage of Conditions in authorization statements is not worthy for other kind of information. Models 2-5, 6, 10-12 in Table 1 are examples of these cases. In these models, it is allowed to use subject and/or target elements (e.g. VMs, volumes, users, roles, ...) but no additional information (e.g. context information) as conditions. If conditions are enabled in the trust relationship, it means that additional information can be used apart from those targets and/or subjects already used in another part of the rule.

4.2. Existential Trust Relationships

The group of *existential* trust relationships define the $TrustedInfo$ as a specific set of instances of the information model of the *Trustor*. In this context, an instance represents a concrete occurrence of an object available in the *Trustor* information system model, including context information, VMs, storage volumes, users, roles, etc.

There is an important conceptual difference between the existential and universal trust relationships. Universal trust relationships share the complete information set, including further new information that can be added to the *Trustor* information system, i.e. it is a open set. On the other hand, existential relationships keep the shared information limited to the specified instances, even when new ones are added to the system, i.e. it is a closed set. The existential trust operator, presented in Definition 6, determines the $TrustedInfo$ as a set of instances $I_1, I_2, ...I_n$, each one belonging to one concept $C_1, C_2, ...C_m$. These instances can be used in different fields of the *Trustee* authorization statements, depending on the trust relationship employed. The kinds of trust

relationships in this group are analogous to the universal ones. Definition 6 shows an example of an existential trust relationship for conditions, analogous to relationship 1 in Table 1.

$$A \prec^{E}_{C} B = (A, B, TrustedInfo) \tag{1}$$
$$where\ TrustedInfo = \{I1_{C1}, I2_{C1}, \ldots, In_{C1}, I1_{Cm}, I2_{Cm}, \ldots, In_{Cm}, \} \Rightarrow \tag{2}$$
$$(B, [B : AnyCondition | A : I1_{C1} | A : I2_{C1} | \ldots | A : In_{C1} | A : I1_{Cm} | A : I2_{Cm} | \ldots | A : In_{Cm}], \tag{3}$$
$$[B : AnySubject | B : AnyRole], B : AnyTarget, AnyPrivilege) \tag{4}$$

Definition 6: Existential Trust Operator for Conditions

Table 2 summarizes the identified existential trust relationships. Columns in Table 2 specify a set of instances I, each one belonging to a particular concept. It should be noticed that it is the same set of instances in all columns. This set determines the information that is shared in the collaboration agreement. If the set appears in one column of the table, it means that the shared instances can be used in the corresponding field of the authorization statement. It is important to remark that henceforth roles are considered equivalent to subjects. This is because the real reason for managing them distinctly in the *Universal Trust Relationships* described in the previous subsection were not related to security but only to privacy, trying to minimize the information exposed between tenants. Now, in the rest of trust operators, tenants have a fine-grain control over the information exposed to the user, so that there is not a real necessity to differentiate between subjects and roles and both are handled together.

Id	Trust Model	Condition	Subject / Role	Target
16	$A \prec^{E}_{C} B$	$\{I\}$		
17	$A \prec^{E}_{S} B$		$\{I\}$	
18	$A \prec^{E}_{C,S} B$	$\{I\}$	$\{I\}$	
19	$A \prec^{E}_{T} B$			$\{I\}$
20	$A \prec^{E}_{C,T} B$	$\{I\}$		$\{I\}$
21	$A \prec^{E}_{S,T} B$		$\{I\}$	$\{I\}$
22	$A \prec^{E}_{C,S,T} B$	$\{I\}$	$\{I\}$	$\{I\}$

Table 2: Summary of Existential Trust Relationships

4.3. Typed Trust Relationships

The *Typed* group of trust relationships is defined as an hybrid combination of the universal and existential ones. Concretely, these trust relationships define the $TrustedInfo$ as a set of instances and concepts. Instances compose a closed set, similar to existential trust relationships. In fact, typed trust relationships can be considered as an extension of the existential ones. In turn, concepts represent open sets of instances. Each concept represents all the instances available in

the information system about that concept, including future instances. A concept is defined as a class or type of resource (e.g. a VM) or an attribute (e.g. overhead of a VM) in the information system. This combination of open and closed sets of instances results in a powerful trust operator with differentiated expressiveness. This is the reason for which we decided to consider them as a different kind of trust relationships. It provides high expressiveness to define trust relationships, which makes it easier the specification and management of such relationships. Definition 7 shows an example of a typed trust relationship for conditions, analogous to relationship 1 in Table 1.

$$A \prec^{T}_{C} B = (A, B, TrustedInfo) \tag{1}$$

$$where\ TrustedInfo = \{I1_{C1}, I2_{C1}, \ldots, In_{C1}, I1_{Cm}, I2_{Cm}, \ldots, In_{Cm}\} \cup \{C1, C2, \ldots Ck\} \Rightarrow \tag{2}$$

$$(B, [B : AnyCondition | A : I1_{C1} | A : I2_{C1} | \ldots | A : In_{C1} | A : I1_{Cm} | A : I2_{Cm} | \ldots | A : In_{Cm} | A : C1 | A : C2 | \ldots | A : Ck], \tag{3}$$

$$[B : AnySubject | B : AnyRole], B : AnyTarget, AnyPrivilege) \tag{4}$$

Definition 7: Typed Trust Operator for Conditions

Table 3 summarizes the identified kinds of typed trust relationships. As for the previous table, note how columns in Table 3 define the same set of instances I and concepts C. This set determines the information that is shared in the collaboration agreement. The column where it appears indicates that it can be used in the corresponding field of the authorization statement.

Id	Trust Model	Condition	Subject / Role	Target
23	$A \prec^{T}_{C} B$	$\{I\} \cup \{C\}$		
24	$A \prec^{T}_{S} B$		$\{I\} \cup \{C\}$	
25	$A \prec^{T}_{C,S} B$	$\{I\} \cup \{C\}$	$\{I\} \cup \{C\}$	
26	$A \prec^{T}_{T} B$			$\{I\} \cup \{C\}$
27	$A \prec^{T}_{C,T} B$	$\{I\} \cup \{C\}$		$\{I\} \cup \{C\}$
28	$A \prec^{T}_{S,T} B$		$\{I\} \cup \{C\}$	$\{I\} \cup \{C\}$
29	$A \prec^{T}_{C,T,S} B$	$\{I\} \cup \{C\}$	$\{I\} \cup \{C\}$	$\{I\} \cup \{C\}$

Table 3: Summary of Typed Trust Relationships

4.4. Fine-Grain Trust Relationships

All the previous groups of trust relationships define the $TrustedInfo$ as a single set of information. Thus, when trust operators enable the *Trustee* to use such information in more than one authorization statement field, there is no way to determine what subset of this information can be used in each different field. To cope with this issue, the group of fine-grain trust relationships extends the $TrustedInfo$ to enable the definition of a variable number of information sets. Each set is composed by the instances and concepts that can be used in each field of the authorization

statements. As an example, let us consider a scenario in which a *Trustor* wants to share two subjects, Alice and Bob, with a *Trustee*. But it wants to limit the usage of Bob only as condition while Alice can be used as a condition and as a subject. This would allow the *Trustee* to assign privileges to Alice, but not to Bob, while either Bob or Alice could appear as conditions (e.g. the authorization statement only applies when Bob is logged-in). Definition 8 shows an example of a fine-grain trust relationship for conditions, subjects and targets, analogous to relationship 15 in Table 1.

$$A \prec^{F}_{C,S,T} B = (A, B, TrustedInfo) \tag{1}$$

$$where\ TrustedInfo = \{C\}, \{S\}, \{T\}\ and \tag{2}$$

$$\{C\} = \{I1_{C1}, I2_{C1}, \ldots, In_{C1}, I1_{Cm}, I2_{Cm}, \ldots, In_{Cm}\} \cup \{C1, C2, \ldots, Ck\} \tag{3}$$

$$\{S\} = \{I1_{C1}, I2_{C1}, \ldots, In_{C1}, I1_{Cm}, I2_{Cm}, \ldots, In_{Cm}\} \cup \{C1, C2, \ldots, Ck\} \tag{4}$$

$$\{T\} = \{I1_{C1}, I2_{C1}, \ldots, In_{C1}, I1_{Cm}, I2_{Cm}, \ldots, In_{Cm}\} \cup \{C1, C2, \ldots, Ck\} \tag{5}$$

Definition 8: Fine Grain Trust Relationship for Conditions, Subjects and Targets

Note how this fine-grain trust relationship can be combined with both existential and typed ones, by applying a fine-grain definition when more than one field of the authorization statement is used. This happens in the existential trust relationships 18, 20, 21 and 22 and the typed trust relationships 25, 27, 28 and 29. Thus, the fine-grain model operator is denoted by two different superscripts: F and FT. The trust operators $\prec^{F}$ and $\prec^{FT}$ represent the combination with existential and typed trust relationships, respectively. The result of this combination is shown in Table 4.4. The reader can note how now each different field has its own set of information to be shared between tenants.

Id	Trust Model	Condition	Subject / Role	Target
30	$A \prec^{F}_{C,S} B$	$[\{I\}]_C$	$[\{I\}]_S$	
31	$A \prec^{F}_{C,T} B$	$[\{I\}]_C$		$[\{I\}]_T$
32	$A \prec^{F}_{S,T} B$		$[\{I\}]_S$	$[\{I\}]_T$
33	$A \prec^{F}_{C,S,T} B$	$[\{I\}]_C$	$[\{I\}]_S$	$[\{I\}]_T$
34	$A \prec^{FT}_{C,S} B$	$[\{I\} \cup \{C\}]_C$	$[\{I\} \cup \{C\}]_S$	
35	$A \prec^{FT}_{C,T} B$	$[\{I\} \cup \{C\}]_C$		$[\{I\} \cup \{C\}]_T$
36	$A \prec^{FT}_{S,T} B$		$[\{I\} \cup \{C\}]_S$	$[\{I\} \cup \{C\}]_T$
37	$A \prec^{FT}_{C,S,T} B$	$[\{I\} \cup \{C\}]_C$	$[\{I\} \cup \{C\}]_S$	$[\{I\} \cup \{C\}]_T$

Table 4: Summary of Fine Grain Trust Models

5. Trust Manager Architecture

The trust relationships described in the previous section are used by cloud tenants to define federated scenarios in cloud computing. The different layers of the cloud stack (Infrastructure-as-a-Service, Platform-as-a-Service and Software-as-a-Service) should be able to support these definitions, as well as the capability to manage these trust relationships. In case the reader is more interested in the definition of the cloud stack, Lenk et al [23] provide a comprehensible overview. In this paper, we describe a trust management architecture for the Infrastructure-as-a-Service layer. It is provided as a mere example which has been implemented in our prototype but it is noteworthy underline that our proposal is generic enough to fit in PaaS and, although it may become more complex, in SaaS. The main difference between the layers regarding the application of these trust relationships is related to the elements of the system model. In IaaS, these elements are usually a reduced and controlled set of elements that corresponds to those usually present in a virtual infrastructure (e.g. virtual machines, volumes, etc.), while in PaaS or SaaS the elements of the system model usually depend on the particular service offered. That is, different PaaS may offer different platform services and thus manage different system model elements. In the case of SaaS this may become even more complex and it is not the most suitable layer, although it could fit to services where the system model is controlled. It should be noticed that this affects the complexity of managing the set of authorization elements, but all of the cloud layers share basic assumptions that have been borne in mind to design our proposal: i) multi-tenancy and ii) shared privileges among all the tenants. Thus, the proposal is meant to be applied in cloud computing and not other scenarios like grid computing where the resources shared between users are completely heterogeneous.

Figure 1 gives an overview of the proposed architecture for trust management in the IaaS layer. The figure represents a Cloud Service Provider IaaS providing service to different tenants. The *AuthzTrustManager* represents the IaaS authorization system that controls the access to the resources provided by this IaaS. The way in which trust relationships are managed is as follows. For each tenant, this system keeps up-to-date a repository that contains the trust and authorization statements defined for that tenant in the system. Since our authorization model is based on semantic web technologies, this repository consists on a Knowledge Base (KB) that stores the tuples representing such statements. Notice that even though the *AuthzTrustManager* is logically centralized since it is in charge of controlling the access to the whole IaaS, it can be implemented in distributed modules to avoid bottlenecks. In our prototype we have implemented a centralized version.

Once a trust relationship $A \prec B$ is defined using one of the proposed trust relationship operators, the authorization system can use the *Trustor* information as specified in this trust relationship. It should be noticed that this does not involves a security risk for the *Trustor*, since the *Trustee* will only be allowed to reference such information for defining its own authorization statements, according to the defined trust relationship.

The *AuthzTrustManager* should keep the trust information up-to-date with the preferences specified by the tenants. For this reason, the *AuthzTrustManager* provides an API with operations to manage trust relationships, such as `addTrustRelationship` and `deleteTrustRelationship`. Moreover, the *AuthzTrustManager* governs the maintenance of authorization information. Thus, it

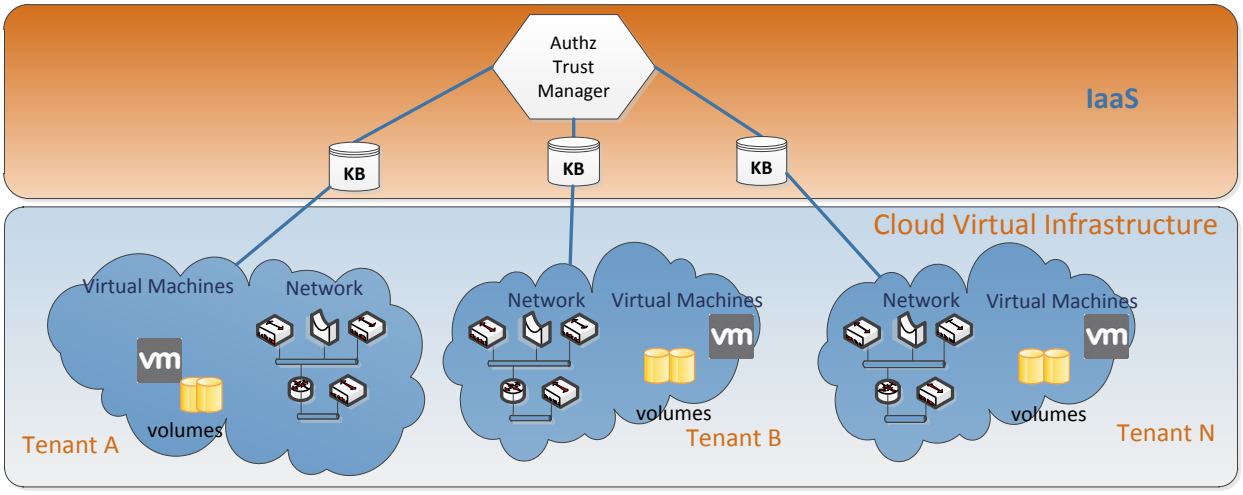

Figure 1: Semantic Authorization Architecture for Cloud

also supports operations like `addGrant` and `deleteGrant`, which are used to manage authorization statements in the authorization system. Every time a new authorization statement is inserted the system, the *AuthzTrustManager* checks that statement in order to ensure that it complies with the current established trust relationships. The *AuthzTrustManager* also provides an *isAuthorized* operation to enable the IaaS to query about an authorization decision.

When a new trust relationship is inserted in the system, there is no action to be done automatically. However, when a trust relationship is removed from the system, the completely knowledge base associated to the *Trustee* has to be revisited in order to mark for removal all the authorization statements in which there is any reference to system elements of the *Trustor*. After all these statements are identified, two different actions can be done. The first option is to remove them directly. Thus, the deny-by-default policy will secure the system. The second option is to keep this authorization statements and only remove those parts of the statement pointing to elements of the *Trustor*. Doing this process, if the resulting authorization statement is valid, it can be keep in the knowledge base. However, if the resulting statement is not valid, for example due to empty field, it is removed automatically. It should be noticed that a particular change in a trust relationship statement could imply a difficult updating process for its related authorized statements in the KB. This is why it is considered the add and delete operations for the trust relationships over the KB.

The *AuthzTrustManager* acts as a Policy Decision Point (PDP) and the IaaS platform forwards user access attempts to it for it to take an authorization decision. When a decision needs to be taken, the *AuthzTrustManager* first retrieved the knowledge base maintained by the tenant requesting the access. Then, the *AuthzTrustManager* checks the trust information to ensure that the request can be validated against the authorization model. Then, the authorization statements are evaluated and the decision is sent back to the IaaS, which will grant or deny the access.

6. Implementation

The proposed trust manager architecture has been prototypically implemented in order to validate the trust relationships described in previous sections over a real authorization system for cloud computing. The OpenStack IaaS cloud system has been modified to allocate our trust manager. OpenStack is an open initiative to provide an IaaS stack for cloud computing. It is becoming one of the most widely used IaaS stack solutions, with an important and increasing list of companies and organizations involved, such as Rackspace, NASA, Citrix, AMD, Intel, NEC or Hewlett Packard. We have extended our previous implementation of the authorization system for OpenStack with the ability to deal with trust relationships. This section provides some implementation details of the prototype.

Figure 2 shows the architecture implemented over OpenStack. We implemented two new APIs to manage the authorization and trust information that complement the IaaS API provided by OpenStack. We use a Trust API to establish, maintain and remove trust relationships between tenants. This API enables the user to define the different types of trust relationships described in this paper. Note that as part of the creation of a trust relationship, the tenant may need (depending of the trust relationships established) to know how to reference system elements belonging to another tenant. For the time being, this sharing of information is done in an out of band method so it is assumed that if A is defining a trust relationship with B it is because they have a pre-established agreement and have shared this information to enable the other part to refer its system elements.

We also use an Authorization API to manage the authorization information associated to each tenant. The key of our implementation is that our authorization API now validates every new authorization statement inserted in the system against the trust statements. This ensures that privacy and security is preserved in the definition of the authorization statements. This check is done in two different times. Firstly, when there is an insertion of a new authorization statement in the system. Secondly, when there is a removal of a trust relationship between tenants. It ensures the complete correctness of the system. Then, the authorization statements can be defined using any namespace in any of the fields, and in the moment in which it is inserted in the system, the validation against trust information decides if the authorization statement is valid or not. In case it is not valid, the statement is automatically discarded. In our implementation, we decided to keep trust and authorization knowledge bases physically separated in order to isolate information against security holes. The implemented architecture intercepts all the request issued to the OpenStack IaaS API (Nova API) and derivates these requests to the *Authorization Engine*, which uses the authorization information of the associated tenant to determine the access decision for the requested action.

The AuthzEngine and trust management TrustManager modules have been developed in Java, using the Java API for XML Web Services (JAX-WS) stack to provide a Web service interface. Pellet [24] has been used as reasoner and Jena [25] is used to perform ontology operations and to manage the authorization and trust tuples stored in the knowledge bases. For the proof of concept implementation, an interceptor has been placed in the EC2 IaaS API interface of the OpenStack Nova component, which receives all the requests to be served by the IaaS. Each request is checked against the Authorizer class of the EC2 nova-api module in order to determine if the user is autho-

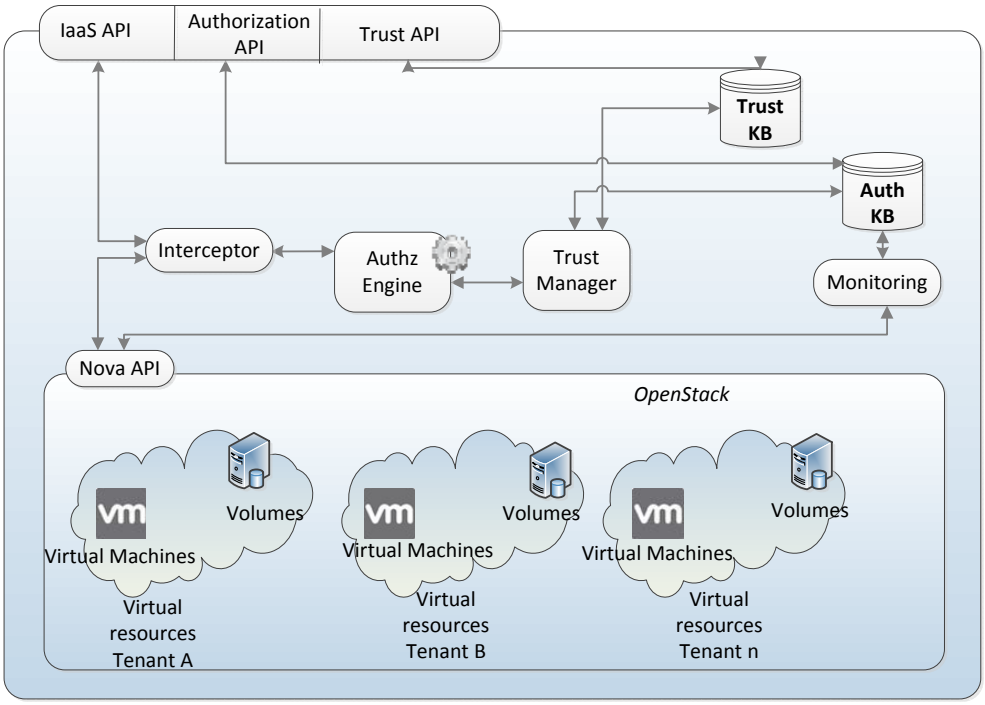

Figure 2: Authorization concepts

rized or not. The Authorizer class has been modified to intercept the authorization requests and to forward them to the AuthzEngine module which provides the authorization decision, accordingly.

Thanks to the use of a different KB and rule reasoner for each tenant, the trust and authorization system scales properly when reasoning with a large overall amount of users and resources as well as authorization and trust statements. We encourage the reader to read our previous research work [21] in order to get a complete overview of the implementation aspects of the authorization system including a complete and exhaustive performance, overhead and scalability testbed in cloud architectures. Note that every invocation request against the IaaS API does not require the usage of the trust manager, it just uses the authorization system. The inclusion of trust relationships enables the management of available concepts to define authorization rules. They affect the expressiveness of the authorization statements that can be defined by a cloud tenant, but they do not affect performance during authorization. For this reason, we have decided not to provide performance results again, as they are similar to the ones provided in our previous research work. Our extension only affects the operation of inserting new authorization statements, not to the response time when accessing a resource. Anyway, it should be also noticed that trust relationships between identities are established only over special circumstances.

7. Scenarios

As a result of our work on cloud computing infrastructures, we have identified a set of common scenarios for federated cloud computing environments which justify the need of this research work. A good example is the definition of a collaboration agreement between two tenants in which one

of them is offering some kind of service that is used by another one. For instance, let us assume a tenant B offering a service at PaaS level that runs on SystemX in B's virtual infrastructure at IaaS level. This service is used by another tenant A, and B wants to enable A to scale the underlying virtual infrastructure for their collaborative work. Thus, some users of A should be able to scale the virtual infrastructure of B as required by the needs of A while using the service. In this situation, B should give permissions in the IaaS infrastructure to these users of A for them to be able to scale up and down the service during their usage. Figure 3 depicts this scenario.

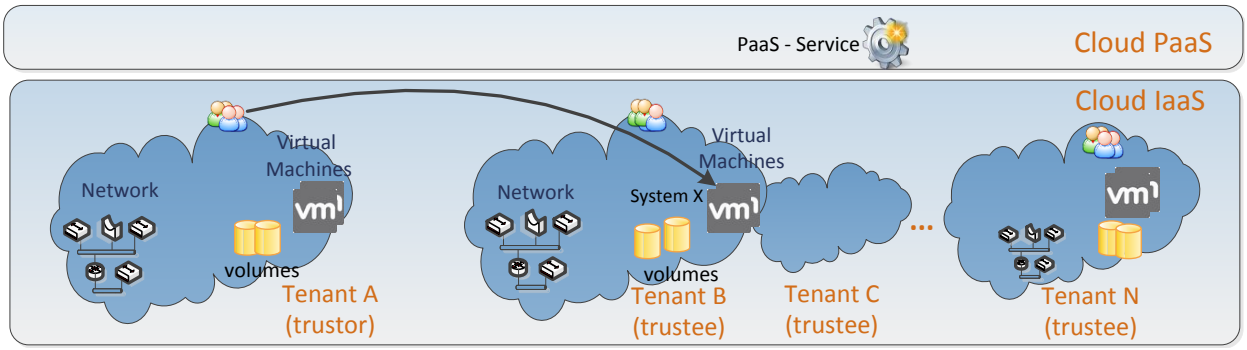

Figure 3: Trust relationship scenario example

This scenario can be controlled by means of a universal trust relationship in which tenant A allows B to define access rights to some parts of its provided infrastructure. This would grant users of tenant A access to B's resources:

$$A \prec_S^U B \tag{1}$$

In this case, the *Trustee* B is offering the service to the *Trustor* A. With this trust relationship, the *Trustor* is allowing the *Trustee* to use its users in the definition of access policies for managing the virtual infrastructure. The *Trustee* B has the control over its own provided infrastructure and specifies which users of the *Trustor* A are able to scale the PaaS service using its IaaS infrastructure. Usually, the *Trustee* will only allow *Trustor*'s users to manage the part of the infrastructure which is needed for the offered service.

Additionally, an existential trust relationship can also be considered for this scenario:

$$A \prec_S^E B = (A, B, TrustedInfo) \tag{1}$$

$$where\ TrustedInfo = \{A:Alice, A:Bob, A:Admin\} \tag{2}$$

In this second case, the *Trustor* is not allowing the *Trustee* to access all its users, but rather it controls which specific users can be established by the *Trustee* to scale the infrastructure. Again,

the control over the infrastructure is still kept by its owner, i.e. the *Trustee* tenant B. By using this trust relationship, the *Trustee* is providing access control to the service transparently. It enables to refer *Trustor* users without having to register new users in the *Trustee* for the purposes of the service.

An authorization statement defined by tenant B could be defined as follows. SystemX is the set of virtual infrastructure elements of the tenant B that are accessible by users *Bob* and *Alice*, as well as any user belonging to role *Admin*. These users are granted to perform run and stop operations over the system to scale up and down the system according to the PaaS service needs. Notice that the rule could contain any amount of users as well as allowing the specification of roles to group users.

$$(B, true, [A : Bob, A : Alice, A : Admin], [B : systemX], [run, stop]) \quad (1)$$

In case tenant A makes use of several services offered by different tenants, the corresponding trust relationships and authorization statements should be defined. For instance, suppose that apart from making use of SystemX of tenant B, tenant A uses a storage service offered by another tenant C, it makes use of an entry point service with load balancing that is offered by tenant D, etc. In this case, the *Trustor* (tenant A) should share its authorization elements with multiple *Trustees* (tenant B, tenant C, tenant D ...). *Trustor* A should define trust relationships for *Trustees* in order to enable them to use A's users in their authorization statements:

$$A \prec_S^E B, \qquad A \prec_S^E C, \qquad A \prec_S^E D \ldots$$

This would allow the different *Trustees* to grant access to A's users for scaling the different services. For instance, tenant C may now define an authorization rule like 1 to allow database administrators (users with role DatabaseAdmins of tenant A) to mount and umount a specific volume. Similarly, tenant D may define an authorization rule like 2 to enable user Joe of tenant A to start or stop SystemY in order to scale the service according to A's needs.

$$(C, true, [A : DatabaseAdmins], [C : VolumeA], [mount, umount]) \quad (1)$$
$$(D, true, [A : Joe], [D : SystemY], [start, stop]) \quad (2)$$

Another scenario in multi-tenancy cloud environments is that in which two tenants sign some kind of agreement for sharing the IaaS infrastructure. For instance, tenants A and B are running a service which makes use of both tenants' virtual infrastructures. Thus, tenant A may allow tenant B to define access control rules for some part of A's infrastructure in order to share this part of the infrastructure. It should be noticed that in case of a two-way collaboration, B can similarly allow tenant A to define rules for part of its infrastructure.

In this scenario, universal trust relationships may not be adequate, since usually tenant A is not willing to allow access to all its infrastructure. An existential or typed trust relationship seems to be more feasible. It would only give access to the elements that are needed to run the service subject to the collaboration agreement. Moreover, the use of *Trustor* information as conditions is also interesting in this scenario in order to enable the *Trustee* to define access control rules based on the *Trustor* elements status. For example, to allow some users to start a virtual machine only if the overhead of another one is beyond 80%.

$$A \prec^{E}_{C,T} B$$

On the other hand, tenant A may also want to allow tenant B the usage of some kind of elements, but still not the whole infrastructure. For instance, to enable B the usage of A's virtual machines or the usage of virtual volumes for a shared database service. In such a case, a Typed trust relationship can be used. This relationship enables the *Trustor* A to define the class of elements to be used by the *Trustee* B, e.g. virtual machine instances.

$$A \prec^{T}_{C,T} B$$

Using these trust relationships, the *Trustee* is able to define authorization rules for its own users to access the *Trustor* infrastructure elements. However, in this kind of collaboration, it is usually desirable that users of both tenants make use of the infrastructure to run the service. A trust relationship enabling the *Trustee* to use both targets and subjects of the *Trustor* to define authorization rules can be used in this case. Again, either the existential or the typed relationships can be used.

$$A \prec^{E}_{C,S,T} B \quad \text{or} \quad A \prec^{T}_{C,S,T} B$$

Finally, a fine grained model is also useful in any of the aforementioned scenarios. This kind of model provides the most expressiveness to control the access to the *Trustor* information. It can be used to define the specific instances to be used by the *Trustee* for access control policy definition.

8. Discussion

There are some aspects raised during the definition of the different trust relationships which may be worthy to be discussed. Firstly, we realized that privileges either have to be shared between collaborating tenants or they have to be identical. Otherwise, actions which apply to one tenant's infrastructure may differ from those of another tenant and authorization rules might not be compatible. In cloud computing, the privileges are identical because all the tenants share the same underlying cloud provider. This entails that information schemas are common to the infrastructure, including the privileges and actions that can be performed in the infrastructure of such a cloud provider.

A set of different trust relationships have been identified for establishing collaboration agreements between tenants in cloud computing. These relationships enable the definition of several collaboration scenarios in which different security and privacy levels are determined. This can be an important differentiating key feature with respect to current cloud architectures that do not

support such inter-tenant collaboration. For example, OpenStack, CloudStack and Amazon EC2 do not enable federations in order to support inter-tenant collaborations. Other advanced solutions like HP Cells provide a basic trust definition based on universal operators. However, there is not any provider with fine-grain support for defining federation environments.

The trust relationships exposed in this paper can be compared according to several aspects like: i) the *Trustor's* privacy control over the data shared with the *Trustee*; ii) the *Trustor's* level of easiness on the trust management; and iii) the risk assumed by the *Trustor* with respect to the *Trustee*, i.e. the level of trust determined by the *Trustor* over the *Trustee*. For instance, a universal trust relationship usually entails a very high risk assumed by the *Trustor* since it is enabling access to all its authorization information to the *Trustee*, while with a fine grain trust relationship the risk is low since the *Trustor* has full control over what information can be used by the *Trustee* for authorization. Similarly, the privacy control in a universal trust relationship is low since this kind of trust statement offers no control over which elements can be used by the *Trustee* for authorization, while a fine grain trust relationship offers a very high control on this information. On another hand, the universal trust relationship is very easy to manage (the *Trustor* just has to define the trust relationship), while a fine grain trust relationship results more complex since the *TrustedInfo* set is more complex and contains more information that should be managed. A comparison of these aspects for the different kinds of trust relationships is depicted in Figure 4. Notice that the figure does not pretend to provide an accurate comparison. Instead, it aims to depict a general overview of the different analyzed aspects.

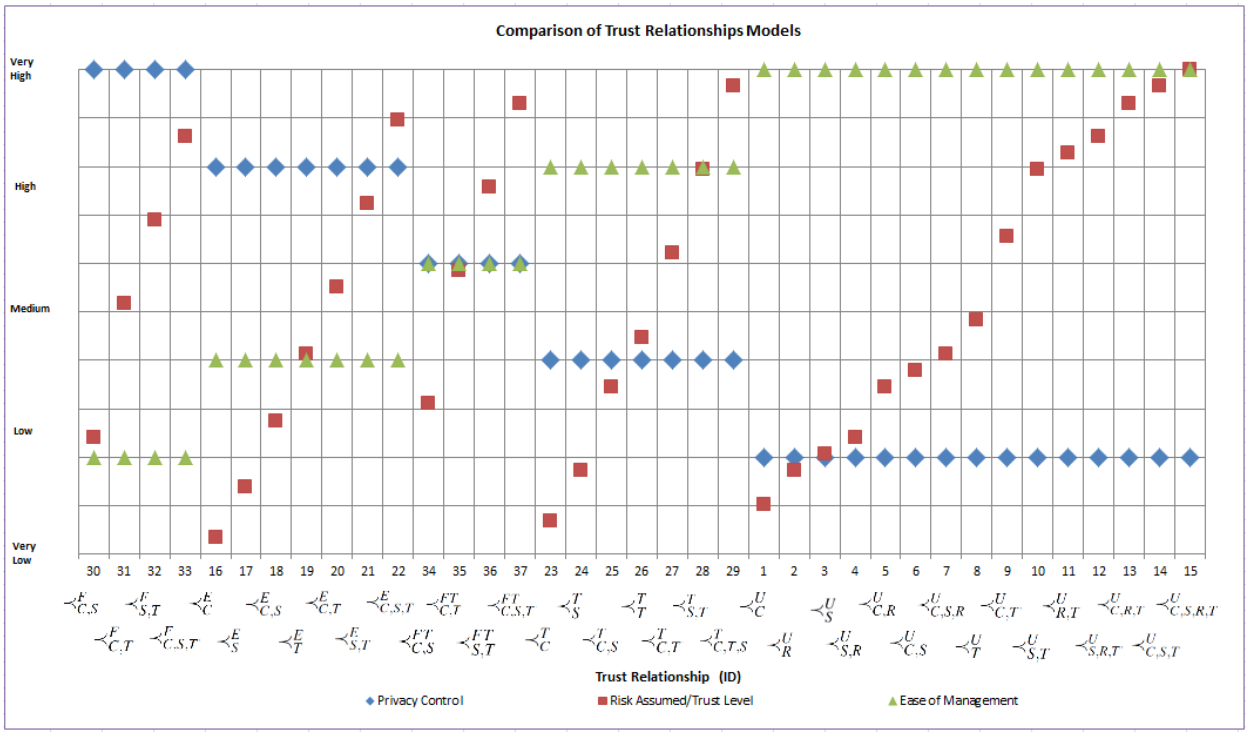

Figure 4: Comparison of Trust Relationships

In the figure, it can be seen how easiness of management and privacy control are indirectly

proportional. As the former is increased, the latter is linearly reduced. It can be also observed that all the trust relationships defined over conditions are less risky than those defined over subjects. In turn, these ones are less risky than trust relationships over targets, and so on. This fact creates a range of trust relationships ordered ascendantly, according to the risk assumed by the *Trustor*, i.e. the level of trust. This order is as follows: $\prec_C, \prec_S, \prec_{C,S}, \prec_{C,T}, \prec_{S,T}, \prec_{C,S,T}$. Figure 4 can be useful to graphically determine the most convenient trust relationship according to these levels.

In the case of a large number of *Trustees*, it should be considered how the different parameters are affected by increasing the number of *Trustees*. Thus, the privacy control is lowered as the number of *Trustees* increases since more information is shared with more *Trustees*. The amount of privacy control that is lost when adding a new trust relationship with another *Trustee* tenant is proportional to the privacy control provided by the specific kind of trust relationship that is established for that tenant. Similarly, the risk assumed or trust level that is established increases as the number of *Trustees* increases. Again, the increase in risk when a new *Trustee* tenant is added is proportional to the risk assumed by the specific kind of trust relationship established for that tenant. Regarding ease of management, it is also decremented as the number of *Trustees* increases since more trust relationships should be managed.

It is worth mentioning the important conceptual difference between the definition of open sets of information -usually referred to universal trust relationships- and the definition of closed sets of information -usually referred to existential trust relationships. This difference is an important issue regarding privacy control of the data shared between *Trustor* and *Trustee*. It is a key aspect of the proposed trust relationships. The provisioning of a hybrid method which enables the definition of closed and open sets provides a convenient way of controlling privacy in cloud computing.

9. Conclusions and Future Work

A complete set of trust relationships have been described, enabling the definition of different collaboration agreements in cloud computing scenarios. Moreover, a comparison of all these trust relationships has been provided, analyzing aspects such as privacy control, risk assumed, trust level and easiness of management. A trust management architecture has been also devised. It allows that tenants trust relationship definitions can be used to manage and control the collaboration agreements between them. This trust management architecture has been validated by means of a prototype implemented using semantic web technologies. Moreover, a diverse set of common scenarios in cloud computing has been described, proposing the most convenient trust relationships for each case.

As future work, we envisage to accomplish an empirical performance analysis of the trusted models in order to define a testbed in which the different trust relationships can be computationally compared. Note the complexity associated to the definition of a comparable scenario in which heterogeneous trust relationships can be analyzed. Moreover, the extension of the presented kinds of trust relationships, as well as the proposed trust manager in order to cope with inter-cloud trust relationships is also an expected step in this research line.

Acknowledgements

This work has been partially funded with support from the Spanish MICINN (project RECLAMO –Virtual and Collaborative Honeynets based on Trust Management and Autonomous Systems applied to Intrusion Management– with code TIN2011-28287-C02-02) and the European project "Interoperable Trust Assurance Infrastructure" (INTER-TRUST -ICT FP7- G.A. 317731), within the European Commission 7th Framework Programme (FP7-ICT-2011-8).

References

[1] OpenStack, Open Source Cloud Computing Software, http://openstack.org, 2011.

[2] V.Vijayakumar, R. Banu, Security for Resource Selection in Grid Computing Based On Trust and Reputation Responsiveness, International Journal of Computer Science and Network Security 8 (11) (2008) 107 – 105.

[3] J. Taige, Q. Xiaolin, A trustworthiness-based access control model in grid system, in: International Conference on Computational Intelligence and Software Engineering CiSE 2009, 1–6, 2009.

[4] M. K. Muchahari, S. K. Sinha, A New Trust Management Architecture for Cloud Computing Environment, in: 2012 International Symposium on Cloud and Services Computing, 136–140, 2012.

[5] S. Wang, L. Zhang, N. Ma, S. Wang, An evaluation approach of subjective trust based on cloud model, Transform 21 (2008) 1062–1068.

[6] J. Abawajy, Determining service trustworthiness in inter loud computing environments, in: ISPAN 2009 : Proceedings of the 2009 10th International Symposium on the Pervasive Systems, Algorithms and Networks, 784–788, 2009.

[7] L. Boursas, W. Hommel, Multidimensional Dynamic Trust Management for Federated Services, Computational Science and Engineering, IEEE International Conference on 2 (2009) 684–689.

[8] R. Hu, J. Liu, X. F. Liu, A Trustworthiness Fusion Model for Service Cloud Platform Based on D-S Evidence Theory, Cluster Computing and the Grid, IEEE International Symposium on 0 (2011) 566–571.

[9] H. Tran, P. Watters, M. Hitchens, V. Varadharajan, Trust and Authorization in the Grid: A Recommendation Model, in: IEEE (Ed.), Proceedings. International Conference on Pervasive Services, 433 – 436, 2005.

[10] J. L. Xudong Ni, A Trust Aware Access Control in Service Oriented Grid Environment, in: IEEE (Ed.), Sixth International Conference on Grid and Cooperative Computing, 1–6, 2007.

[11] B. Lang, Z. Wang, Q. Wang, Trust Representation and Reasoning for Access Control in Large Scale Distributed Systems, in: IEEE (Ed.), 2nd International Conference on Pervasive Computing and Applications, IEEE, 436–441, 2007.

[12] T. Zhao, S. Dong, A Trust Aware Grid Access Control Architecture Based on ABAC, in: 2010 Fifth IEEE International Conference on Networking, Architecture, and Storage, 1–6, 2010.

[13] F. Fakhar, M. A. Shibli, Comparative Analysis on Security Mechanisms in Cloud, in: 2013 15th International Conference on Advanced Communication Technology (ICACT), 145–50, 2013.

[14] R. Yang, C. Lin, Y. Jiang, X. Chu, Trust Based Access Control in Infrastructure-centric Environment, in: IEEE (Ed.), IEEE International Conference on Communications (ICC), 1–5, 2011.

[15] F. Fujun, L. Junshan, Trust based Authorization and Access Control, in: IEEE (Ed.), 2009 International Forum on Information Technology and Applications, 162–165, 2009.

[16] H. Xiong, B. Zhang, Research on Context and Trust-Based Grid Service Authorization Model, in: IEEE (Ed.), 2010 International Conference on Multimedia Information Networking and Security, 433–437, 2010.

[17] Z. Xiao-jun, L. Shi-qin, Y. Xue-li, Z. Guang-Ping, Dynamic Authorization of Grid Based on Trust Mechanism, in: IEEE (Ed.), 2010 International Symposium on Intelligence Information Processing and Trusted Computing, 417–421, 2010.

[18] C. Ngo, P. Membrey, Y. Demchenko, C. de Laat, Policy and Context Management in Dynamically Provisioned Access Control Service for Virtualized Cloud Infrastructures, in: 2012 Seventh International Conference on Availability, Reliability and Security, 343–349, 2012.

[19] F. G. Marmol, G. M. Perez, Towards pre-standardization of trust and reputation models for distributed and heterogeneous systems, Computer Standards and Interfaces 32 (4) (2010) 185 – 196.

[20] W. Viriyasitavat, A. Martin, A Survey of Trust in Workflows and Relevant Contexts, Communications Surveys Tutorials, IEEE PP (99) (2011) 1 –30.

[21] J. B. Bernabe, J. M. M. Perez, J. M. A. Calero, A. F. G. S. Felix J. Garcia Clemente, Gregorio Martinez Perez, Semantic-aware Multi-tenancy Authorization System for Cloud Architectures, Future Generation Computer Systems 32 (2014) 154–167.

[22] J. M. Alcaraz-Calero, N. Edwards, J. Kirschnick, L. Wilcock, M. Wray, Towards a Multi-tenancy Authorization System for Cloud Services, IEEE Security and Privacy 8 (6) (2010) 48–55.

[23] A. Lenk, M. Klems, J. Nimis, S. Tai, T. Sandholm, Whats inside the Cloud? An architectural map of the Cloud landscape, in: Proceeding at ICSE Workshop on Software Engineering Challenges of Cloud Computing, 1–6, 2009.

[24] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, Y. Katz, Pellet: A practical OWL-DL reasoner, Journal of Web Semantics 5 (2) (2007) 51 – 53.

[25] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, K. Wilkinson, Jena: Implementing the Semantic Web Recommendations, in: Proceedings of the 13th international World Wide Web conference, ACM Press, 74–83, 2004.