A Model for Trust-Based Access Control and Delegation in Mobile Clouds*

Indrajit Ray[1], Dieudonne Mulamba[1], Indrakshi Ray[1], and Keesook J. Han[2]

[1] Dept. of Computer Science, Colorado State University, Fort Collins, CO 80523
{indrajit,mulamba,iray}@cs.colostate.edu
[2] Air Force Research Laboratory/RIGA, 525 Brooks Road, Rome, NY 13441
Keesook.Han@rl.af.mil

Abstract. Multi-tenancy, elasticity and dynamicity pose several novel challenges for access control in mobile smartphone clouds such as the Android™ cloud. Accessing subjects may dynamically change, resources requiring protection may be created or modified, and a subject's access requirements to resources may change during the course of the application execution. Cloud tenants may need to acquire permissions from different administrative domains based on the services they require. Moreover, all the entities participating in a cloud may not be trusted to the same degree. Traditional access control models are not adequate for mobile clouds. In this work, we propose a new access control framework for mobile smartphone clouds. We formalize a trust-based access control model with delegation for providing fine-grained access control. Our model incorporates the notion of trust in the Role-Based Access Control (RBAC) model and also formalizes the concept of trustworthy delegation.

Keywords: access control model, delegation, mobile cloud security, trust.

1 Introduction

Smartphones and other mobile devices are increasingly shifting the personal computing model away from traditional desktops and laptops to *mobile cloud computing*. In this model, cloud computing, mobile devices and networks seamlessly interact with each other to provide newer types of services that were previously not possible (such as location based services). The unique characteristics of mobile cloud computing – namely, multi-tenancy, elasticity, massive scalability [15], and mobility – introduce novel challenges to authorization and access control. To begin with, multi-tenancy results in the co-residency of machines (virtual machines, database engines etc.) and other resources owned by different clients or tenants at the same privileged position in the cloud with respect to one another. As a result, a guest operating system can exploit vulnerabilities in the hypervisor and run processes on other guests or the host, and security breaches can arise in one smartphone client and propagate to another easily via the cloud. Proper authorization and access control techniques should therefore not only protect tenant resources from un-authorized disclosure and modification from attackers, but also should

* Approved for Public Release; Distribution Unlimited:88ABW-2013-2127 dated 02 May 2013.

L. Wang and B. Shafiq (Eds.): DBSec 2013, LNCS 7964, pp. 242–257, 2013.

allow *segregation* of tenants from one another, and *isolation* of computation, storage and network resources of the mobile cloud provider from tenants.

A mobile cloud environment is inherently very dynamic. The accessing entities may change, resources requiring protection may be created or modified, and an entity's access to resources may change during the course of an application execution. Users need to dynamically acquire permissions from different domains based on the services they need. Interactions among entities may occur in ad hoc manners and where the access-requesting entity may not be known in advance by the access-granting entity. In such situations, traditional identity-based access control models such as Discretionary Access Control (DAC), Mandatory Access Control (MAC) [21], or Role-based Access Control (RBAC) [17,12], that rely on the access-granter knowing the identity of access requester beforehand and authenticating the requester, can no longer be applied.

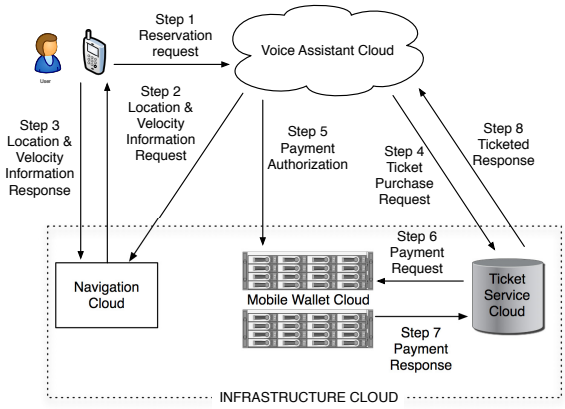

Fig. 1. Mobile smartphone cloud application illustrating need for delegation

Last but not least, in a mobile cloud environment resources are often distributed and managed by different service providers and clients move from one service provider to another to obtain the needed service. To support a specific service, there is frequently the need for coordination and interaction between these different providers. A tenant of one service provider may need to access other providers to obtain the relevant service. This is illustrated by the application scenario in Fig. 1. A mobile smartphone user invokes the voice assistant application (VAC) on her smartphone and instructs it to make a reservation for that evening's show at an en-route theater closest to her destination. VAC computes the time to reach the destination by accessing a navigation service (NavC), identifies candidate theaters by consulting a location service (LocC) and selects one from the list, contacts a ticket service provider cloud (TktC) for a reservation and contacts the user's mobile wallet provider (MobWC) to purchase the ticket from TktC.

For accessing different services the authorization sequence can potentially be as follows. The user initially authorizes the VAC for the ticket purchase task; however the VAC cannot carry out the task on its own and therefore needs to pass on the authorization (or portions thereof) to various other cloud providers. Moreover, if the different

service providers are all tenants of one or more infrastructure cloud(s) (as shown in Fig. 1), then each service provider needs to rely on the infrastructure to manage access control to its resources. *Delegation* of authorization is the principle that allows one service provider to act on behalf of the user to make the user's access rights available to other service providers. There is a need to manage and mediate interactions with distributed resources having distributed administrators. However, authentication of the requesting client (required in conventional access control) may be difficult in mobile smartphone cloud systems. For example, it may not be possible for the VAC to be registered with both the MobWC and the TktC. Hence, delegation may need to proceed without associated authentication. This is further complicated by the fact that often the privilege to delegate may itself need to be delegated.

In this work, we propose a new trust-based access control model that supports complex delegation across different members of the mobile cloud for providing fine-grained access controls. This model is based on extensions to the RBAC model, and adapts concepts from the trust-based access control models proposed earlier by us [7,19]. We assume an existing context sensitive non-binary trust model such as the one in [16]. For this work, we identify the different trust-based access control model elements that are needed for addressing specific challenges in smartphone clouds (Section 3.1), and the relationships among those elements (Section 3.2). We formalize the model and give the rules of access (Sections 3.3 and 3.4). The smartphone cloud system is very dynamic and allows frequent updates to its RBAC relations. Using traditional RBAC relations to control delegation in such environments is of limited advantage because of the resulting inconsistencies in role hierarchy. We adapt the notion of *administrative scope* to resolve dynamically any inconsistency involved in controlling delegations (Section 3.5). Finally, we propose algorithms to perform trust-based delegation, including chained delegation (Section 4).

2 Related Work

Role-Based Access Control (RBAC) is used extensively within organizations for administering and managing privileges and is often considered the de-facto standard for access control.

While the advantages of RBAC are numerous, researchers are increasingly identifying limitations in the model for newer and emerging applications. The biggest limitation is the lack of support for delegation in the standard RBAC model and the failure to support dynamic adaptation of access control policies based on changing needs. Many researchers have extended RBAC to support delegation [3,2,10,18,22,24,25]; however these models fail to support the need for ad hoc authorization and ad hoc delegation and thus cannot readily be adapted to the cloud environment. Researchers have also proposed Credential-based or Attribute-based access control models [6,5] to address the challenges of unknown users in access control. Unfortunately, these models do not allow access control decisions to be dynamically updated or revoked based on the history of the requester, or based on changing requirements of the requester.

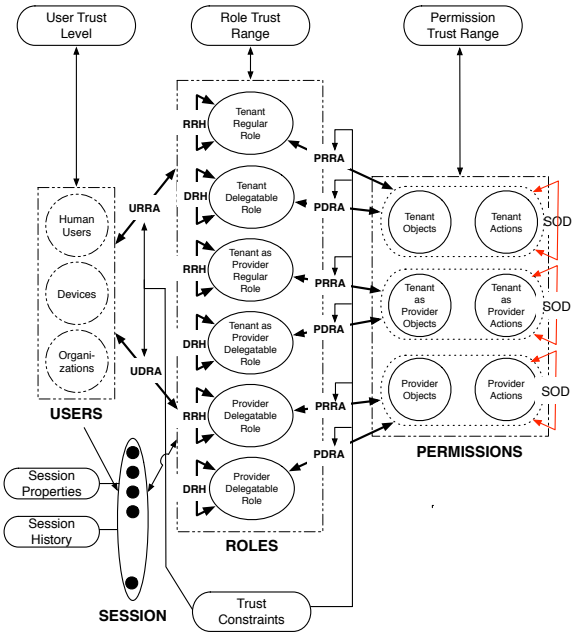

Fig. 2. Schematic overview of the new access control model for clouds

Recently, researchers have proposed the notion of risk-adaptable access control models [9,14,13] and trust-based access control models [7,4,23] to facilitate dynamic adaptation of access control policies based on operational needs and situational awareness. A recent work [1] combines these two philosophies into one comprehensive model. However, none of these works address the problem of delegation in mobile cloud systems. Nonetheless, these approaches look promising and form the basis of the current model.

3 Formal Access Control Model with Delegation

The proposed trust-based access control model is defined in terms of a set of elements and relations among those elements with trust-based constraints defined on these relations. We use a modified graph-theoretic approach similar to the one proposed by Chen and Crampton [8] to express the model semantics. This model is designed to integrate delegation and revocation, with revocation being the reverse process of delegation. The model offers the possibility to perform a single-step as well as a multi-step delegation and revocation. It enables both roles and permissions to be delegated. Fig. 2 provides a schematic overview of the proposed model. We use the access control scenario from Figure 1 to exemplify the model.

3.1 Model Elements

The model elements are of the following types: *user, user_properties, role, object, action, permissions, constraints, trust_level, session_instance, session_type, session,* and *session_history*. The corresponding element sets are represented by the symbols U, $Prop$, R, O, Act, P, $Const$, $\mathscr{D}$, S_I, S_T, S, S_H.

In this model, roles are separated into two broad classes: *regular roles* and *delegatable roles*. A user cannot delegate permissions assigned to a regular role, while permissions assigned to a delegatable role can be delegated. The cloud system is responsible for creating each regular role, while each delegatable role is created and owned by an individual user. Therefore, a regular role is a durable role, while a delegatable role is temporary, created and deleted at the user's discretion. In the cloud system, each user owns a set of delegatable roles that form a role hierarchy determined by the user. All user assignment to regular roles relations and all permission assignment to regular roles relations are managed by the cloud system, while each individual user is responsible for managing all user assignment to delegatable roles relations and all permission assignment to delegatable roles relations.

User. A user $u \in U$ is a human being, a device, an organization or any active agent running on behalf of these. Three categories of users can be found in the cloud, namely, tenant, a tenant-as-provider, and a provider. A tenant is a user that is receiving regular services offered in the cloud, while a tenant-as-provider is a cloud user that is receiving regular services as well as offering regular services. The last category, provider, refers to the cloud provider. To understand these notions, let consider the example of Netflix that uses Amazon's cloud. A Netflix subscriber is a tenant, while Netflix is the tenant-as-provider. The provider in this case is Amazon's cloud.

User_properties. Each user u has a certain set of properties $\mathscr{P}_u$, called *user_properties*. The set $Prop = \bigcup_{u \in U} \mathscr{P}_u$. A user can manifest any subset P_u of $\mathscr{P}_u$ (i.e., $P_u \in 2^{\mathscr{P}_u}$) in a particular session. User properties are used in our model to compute trust levels of users (see later in the list). Some examples of elements of user_properties are: user credentials, public key certificates, and membership in groups.

Role. The concept of role is the same as in the RBAC model. A role $r \in R$ is a job function with some associated semantics regarding the responsibilities conferred to the user. A user assigned to a role specifies the operational needs of the cloud. The set of roles R can be further subdivided into six disjoint subsets as follows:

1. TENANT-REGULAR-ROLE (TRR) – A set of job functions that are relevant for receiving regular services. For example, the human user in our scenario can be an elite member with the mobile wallet service provider that bestows him with certain privileges. In this case "elite member" will be an example of a tenant-regular role.
2. TENANT-DELEGATABLE-ROLE (TDR) – A set of job functions that are relevant for delegating services. Going back to our earlier scenario, the VAC instance working on behalf of the user to purchase tickets needs to have permission at the MobWC to use the user's wallet. Thus, a role such as "wallet user for ticket purchase" which the VAC needs to assume to execute the operation will be an example of tenant-delegatable-role.

3. TENANT-AS-PROVIDER-REGULAR-ROLE ($TPRR$) – A set of job functions that are relevant for receiving regular services as well as offering regular services. The NavC cloud is a provider of navigation services. It uses the infrastructure cloud to provide some of its services (such as the computation needed for the navigation). As a result, we will have a tenant-as-provider-regular-role "navigation computation" at the infrastructure cloud.
4. TENANT-AS-PROVIDER-DELEGATABLE-ROLE ($TPDR$) – A set of job functions that are relevant for receiving as well as offering delegation services. The VAC will have a tenant-as-provider-delegatable role "make ticket reservation" that will allow it to delegate different components of the ticket reservation.
5. PROVIDER-REGULAR-ROLE (PRR) – A set of job functions that are relevant for providing regular services. The instance of MobWC that will access the user's wallet needs to have permission to debit the wallet to provide the service. Thus, an example of this role will be "debit user wallet" that the instance of MobWC needs to acquire for this task.
6. PROVIDER-DELEGATABLE-ROLE (PDR) – A set of job functions that are relevant for providing delegation services. Assume that the TktC offers the ticketing service by using a database provider cloud to keep record of the transaction. Thus, an instance of the TktC that is performing the ticketing operation for the current user needs to delegate the record update operation to the database provider. A "decrement available seats" role will be an example of a provider-delegatable-role in this scenario.

Object. An object $o \in O$ is a data resource as well as a system resource. It can be thought of as a *container* that contains information. The set O (OBJECTS) is partitioned into three subsets:

1. TENANT-OBJECTS (TO) – Objects from this set are accessed when some services are needed. An example is the user's wallet at the MobWC.
2. TENANT-AS-PROVIDER-OBJECTS (TPO) – Objects from this set are accessed for receiving as well as providing services. An example is the database used by the TktC to keep record of transactions.
3. PROVIDER-OBJECTS (PO) – Objects in this set are accessed purely for providing services. Examples include objects used by the infrastructure cloud such as network objects.

Action. An action $a \in A$ is an executable image of a program that operates on some object. The set A (ACTIONS) may have three subtypes ACTIONS:

1. TENANT-ACTIONS (TA) – This set of actions act on objects from the set TENANT-OBJECTS. An example is "debit wallet".
2. TENANT-AS-PROVIDER-ACTIONS (TPA) – This set is comprised of actions that operate on elements of TENANT-AS-PROVIDER-OBJECTS. An example is "update database".
3. PROVIDER-ACTIONS (PA) – This set is comprised of actions that operate on elements of PROVIDER-OBJECTS. An example is "perform read on disk".

Permission. A permission $p \in P$ is an authorization to perform a certain task within the system. A permission is assigned to a role. The set P (PERMISSIONS) is partitioned into three subsets as follows:

1. TENANT-PERMISSIONS (TP) – The set of all ordered pairs $\langle TO, TA \rangle$ where $TO \in$ TENANT-OBJECTS and $TA \in$ TENANT-ACTIONS; that is, $TP = 2^{TO \times TA}$.
2. TENANT-AS-PROVIDER-PERMISSIONS (TPP) – The set of all ordered pairs $\langle TPO, TPA \rangle$ where $TPO \in$ TENANT-AS-PROVIDER-OBJECTS and $TPA \in$ TENANT-AS-PROVIDER-ACTIONS; that is, $TPP = 2^{TPO \times TPA}$.
3. PROVIDER-PERMISSIONS (PP) – The set of all ordered pairs $\langle PO, PA \rangle$ where $PO \in$ PROVIDER-OBJECTS and $PA \in$ PROVIDER-ACTIONS; that is, $PP = 2^{PO \times PA}$.

Trust_level. A trust_level $\in \mathscr{D}$ is a real number between 0 and 1, with 0 being the lowest level of trustworthiness and 1 the highest and intuitively encodes the security risk of the access decision. The larger the value of the trust_level the less risky it is to grant access to the user. A user u associated with a role r at some instant of time has a trust_level. The function $\mathscr{T}(u,r)$ gives the trust_level of the corresponding user associated with the role r.

Role_trust_range. A role_trust_range is an interval $[rt_{min}, 1]$ that is associated with roles. The role_trust_range indirectly encodes the situational factors under which the access decision is made. Each role $r \in R$ is associated with a role_trust_range that gives the minimum trust value rt_{min} required for a user to be assigned to the corresponding role. The function $\mathscr{L}(r)$ returns the value rt_{min} for the given role r.

Permission_trust_range. A permission_trust_range is an interval $[pt_{min}, 1]$ that is associated with permissions. Each permission $Perm \in P$ is associated with a permission_trust_range that gives the minimum trust value pt_{min} required by a role to be assigned the corresponding permission. The function $\mathscr{L}(Perm)$ returns the value pt_{min} for the given permission $Perm$.

Constraint. A constraint $\in$ Cons is defined as a predicate which when applied to a relation between two elements returns a value of "acceptable" (True or 1) or "not-acceptable" (False or 0). Constraints can be viewed as conditions imposed on the relationships and assignments and encode the situational factors under which access decisions need to be made.

Session_instance. A session_instance $\in S_I$ is a 'login' instance of an user. It is the set of roles activated by the user in that login instance. A user can instantiate multiple logins thereby initiating multiple session_instances at the same time. A session_instance is uniquely identified by a system generated id.

Session_type. A session_instance is identified with a type that is defined by the set of user_properties manifested in that session_instance by the user. For a session_instance u_s activated by a user u with set of properties $Prop_u$ ($Prop_u \subseteq \mathscr{P}_u$), the session_type is $prop_u$. Formally, the set $S_T = 2^{Prop}$.

Session. A session $\in S$ is identified by a session_instance with a session_type. A session with session_instance u_s and type $Prop_u$ is denoted by the symbol ${u_s}^{prop_u}$. Formally, $S = S_I \times S_T$.

Session_history. A session_history $\in S_H$ is a set of information regarding the user's roles, behavior and trust levels in a previous use of a session of that type.

3.2 User-Role and Permission-Role Assignment

We propose a graph theoretic semantics for our model. An *authorization graph* is defined in terms of a set V of vertices and a set E of edges. The set of vertices $V = U \cup TRR \cup TDR \cup TPRR \cup TPDR \cup PRR \cup PDR \cup TP \cup TPP \cup PP$ corresponds to the entities users, tenant-regular-roles, tenant-delegatable-roles, tenant-as-provider-regular-roles, tenant-as-provider-delegatable-roles, provider-regular-roles, provider-delegatable-roles, tenant-permissions, tenant-as-providers-permissions and provider-permissions respectively. The set of edges $E = URRA \cup UDRA \cup PRRA \cup PDRA \cup RRH_a \cup DRH_a \cup RRH_u \cup DRH_u$ corresponds to the different relationships between the model's entities.

We now define some relationships with roles that form the building block for the access control model, as follows:

Definition 1. *User-Regular-Role Assignment (URRA)* = $(U \times TRR) \cup (U \times TPRR) \cup (U \times PRR)$ *is a many-to-many relationship that provides the regular roles to which different users can be assigned in the cloud. It is represented as a pair of user-regular-roles.*

Definition 2. *User-Delegatable-Role Assignment (UDRA)* = $(U \times TDR) \cup (U \times TPDR) \cup (U \times PDR)$ *is a many-to-many relationship identifying delegatable roles to which different users can be assigned in the cloud. This relationship is represented as a pair of user-delegatable-roles.*

Definition 3. *Permission-Regular-Role Assignment (PRRA)* = $(TRR \times TP) \cup (TPRR \times TPP) \cup (PRR \times PP)$ *is a many-to-many relationship providing the set of permission-regular-roles that indicate to regular roles which permissions they are allowed to acquire.*

Definition 4. *Permission-Delegatable-Role Assignment (PDRA)* = $(TDR \times TP) \cup (TPDR \times TPP) \cup (PDR \times PP)$ *is a many-to-many relationship providing the set of permission-delegatable-roles identifying the permissions for which delegatable roles are authorized.*

The role-hierarchy defines parent-child relationships between different roles. This allows for easy administration of roles by reducing the number of explicit assignments in the *URRA*, *UDRA*, *PRRA*, and *PDRA* relations. Regular roles are organized in a Regular-Role-Hierarchy while delegatable roles are organized in a Delegatable-Role-Hierarchy with the two hierarchies being disjoint. We define the two hierarchies as follows:

Definition 5. *The Role-hierarchy (RH)* = $\Re \times \{a,u\}$*, is a partial order on the set of roles where* $\Re = ((TRR \times TRR) \cup (TPRR \times TPRR) \cup (PRR \times PRR))$ *in case role is a regular role. In the same case RH is written RRH to represent a regular role-hierarchy. When considering the case when role is a delegatable role, then* $(\Re = ((TDR \times TDR) \cup (TPDR \times TPDR) \cup (PDR \times PDR))$*, and RH becomes DRH to represent a delegatable role-hierarchy. The set* $\{a,u\}$ *can be considered as one of these:*

- *the* role-activation hierarchy
- *the* permission-usage hierarchy

Let $\leq_a$ denote the reflexive transitive closure of $\Re_a$ while $\leq_u$ denotes the reflexive transitive closure of $\Re_u$. Then role-activation hierarchy and permission-usage hierarchy are defined as follows.

Definition 6. *Role-activation hierarchy* ($\Re_a$) = $\{(r,r') : (r,r',a) \in \Re\}$ *is a subset of* $\Re$ *such that a user u may activate a role r in a session_instance if there exists a role* r' *such that* $\{u,r'\} \in URRA$ *and* $r \leq_a r'$ *if r and* r' *are regular roles, or* $\{u,r'\} \in UDRA$ *and* $r \leq_a r'$ *when r and* r' *are delegatable roles.*

Definition 7. *Permission usage hierarchy* ($\Re_u$) = $\{(r,r') : (r,r',u) \in \Re\}$ *such that a user is authorized for permission p if there exists roles* r,r' *such that u may activate* r'*, with* $(p,r) \in PRRA$ *and* $r \leq_u r'$ *in the case* r,r' *are regular roles, or with* $(p,r) \in PDRA$ *and* $r \leq_u r'$ *when* r,r' *are delegatable roles. In the first case* $\Re_u$ *becomes* RRH_u*, while it becomes* DRH_u *when roles are delegatable.*

3.3 Evaluating Access Requests

The cloud security administrator assigns trust constraints in the form of a corresponding *trust interval* to roles, permissions, and other associations between entities based on different characteristics of each model. Note that in the cloud structure, users, tenants or providers of the senior role can perform the same set of duties as its junior role; hence an entity who will be assigned to the senior role require more trustworthiness than the entity who will be assigned to junior role only. Based on this observation, when we introduce the notion of trust value, we assume that the trust value of the senior role always dominates the trust values of its junior roles.

We have previously proposed the notions of *activation path*, *usage path* and *access path* in [19] to evaluate if an access control request should be denied or granted according to the current policy. We adapt and extend these notions here to determine access request for mobile smartphone clouds.

Definition 8. *An activation path (or act-path) between two nodes* v_1 *and* v_n *in the authorization graph is a sequence of vertices* $v_1,\ldots,v_n \in E$ *such that either* $(v_1,v_2) \in URRA$ *and* $(v_{i-1},v_i) \in RRH_a$ *for* $i = 3,\ldots,n$ *on a regular-role hierarchy or* $(v_1,v_2) \in UDRA$ *and* $(v_{i-1},v_i) \in DRH_a$ *for* $i = 3,\ldots,n$ *on a delegatable-role hierarchy.*

Definition 9. *A usage path (or u-path) between two nodes* v_1 *and* v_n *in the authorization graph is a sequence of vertices* $v_1,\ldots,v_n \in E$ *such that either* $(v_i,v_{i+1}) \in RRH_u$ *for* $i = 1,\ldots,n-2$*, and* $(v_{n-1},v_n) \in PRRA$ *on a regular-role hierarchy or* $(v_i,v_{i+1}) \in DRH_u$ *for* $i = 1,\ldots,n-2$*, and* $(v_{n-1},v_n) \in PDRA$ *on a delegatable-role hierarchy.*

Definition 10. *An access path (or acs-path) between two nodes* v_1 *and* v_n *in the authorization graph is a sequence of vertices* $v_1,\ldots,v_n$*, such that* (v_1,v_k) *is an act-path, and* (v_k,v_n) *is an u-path.*

Based on these definitions of usage path, access path and activations we propose three different access control models, namely, the *standard model*, the *strong model* and the *weak model*. The models differ with respect to the trust constraints that must be satisfied by the entities for the authorization to be successful.

3.4 The Standard Model

In the standard model, individual entities are associated with trust values and describe how much the user is trusted to perform a specific role. The role_trust_range associated with a tenant role, a tenant-as-provider role or a provider role specifies the minimum trust level with respect to that role that the user has to acquire in order to activate the corresponding role. Similarly, the permission_trust_range associated with a tenant-permission, tenant-as-provider-permission or provider-permission specifies the minimum trust level with respect to the current role of the user, that needs to be acquired in order to invoke these permission.

Let the trust values for the user be specified by a function $\mathscr{T} : ((U_h \times R_h) \cup (U_d \times R_d)) \rightarrow t \in \mathscr{D}$ and the trust ranges for role and permission by functions $\mathscr{L} : (R \cup P) \rightarrow [l, 1] \subseteq \mathscr{D}$.

- For $u \in U, r \in R$, $\mathscr{T}(u, r)$ denotes the trust value of u with respect to r;
- For $r \in R$, $\mathscr{L}(r)$ denotes the trust interval in which r is active;
- For $p \in P$, $\mathscr{L}(p)$ denotes the trust interval in which p is active.

For any path $v_1, \ldots, v_n$ in the graph $G = (V, E, \mathscr{T}, \mathscr{L})$, where $E = URRA \cup UDRA \cup PRRA \cup PDRA \cup RRH_a \cup DRH_a \cup RRH_u \cup DRH_u$, we write $\hat{\mathscr{L}}(v_2, \ldots, v_n) = \hat{\mathscr{L}}(v_2, v_n) \subseteq \mathscr{D}$ to denote $\bigcap_{i=2}^{n} \mathscr{L}(v_i)$. In other words, $\hat{\mathscr{L}}(v_2, v_n)$ is the trust interval in which every vertex $v_i \in R \cup P$ is enabled.

Authorization in the Standard Model is based on the policy decision point making the following three determinations corresponding to the requested access:

1. Is the role that the user needs to activate in the current session in order to acquire the desired permission, authorized for the permission?
2. Can the user activate the corresponding role?
3. Is the user authorized for the desired permission?

If the policy decision point makes a positive determination for all these conditions a decision to allow the access is made. These three determinations are made based on the following propositions.

Proposition 1. *A role $v_1 \in R$ is authorized for permission $v_n \in P$ if and only if there exists an u-path $v_1, v_2, \ldots, v_n$ and $\mathscr{L}(v_1) \subseteq \hat{\mathscr{L}}(v_1, v_n)$.*

Proposition 2. *A user $v_1 \in U$ may activate role $v_n \in R$ if and only if there exists an act-path $v_1, v_2, \ldots, v_n$ and $\mathscr{T}(v_1, v_2) \in \mathscr{L}(v_2)$.*

Proposition 3. *A user $v_1 \in U$ is authorized for permission $v_n \in P$ if and only if there exists an acs-path $v_1, v_2, \ldots, v_k, \ldots, v_n$ such that $v_k \in R$ for some k, $v_1, \ldots, v_k$ is an act-path, $v_k, \ldots, v_n$ is a u-path, v_1 can activate v_k, and v_k is authorized for v_n.*

3.5 Controlling Delegation

A *delegation operation* temporarily changes the access control state so as to allow the delegatee to use the initial user's access privileges. Delegation operations are classified into two kinds: *grant* operations and *transfer* operations [3]. Grant operations are defined by *grant delegation models* and allow the delegated access rights to be available to both the delegator and the delegatee, after a successful completion of a delegation operation. Transfer operations, on the other hand, are defined by *transfer delegation models* to allow the delegated access right to be acquired by the delegatee, while preventing the delegator from continuing to use the delegated access right. Most works done in this field focus on grant operation [3,2,18,22,24,25]. Crampton and Khambhammettu first introduced the notion of transfer operation in [10].

Designing a mechanism for delegation control involves specifying under which condition a delegation operation is permitted. This requires resolving, in order, the following two issues: (1) Determine whether or not a given user is authorized to delegate a role or permission available to the user. (2) Determine whether a given role or permission can be delegated to a user.

Relations have been the main way to control delegation [2,24,25] in role-based systems. Two principal relations, *can_delegate* and *can_receive*, are used for controlling delegation. Relations are suited to be used in a *RBAC* system but only where *RBAC* relations remain static. However, in a cloud system, *RBAC* relations, such the role hierarchy, are frequently updated. This situation can generate serious inconsistencies in relations *can_delegate* and *can_receive* that specify respectively, who is authorized to delegate access rights, and who is allowed to receive delegated access rights. For this reason, we have opted to use the concept of *administrative scope* [11] to deal with delegation control. Administrative scope is a concept borrowed from *RHA* family of administrative models, and is defined as follows [11].

Definition 11. *Let $r \in R$. We define $\sigma(r)$ the* administrative scope of a role *r as the set $\sigma(r) = \{r'\}$ such that if a role $r' \in \sigma(r)$ then in the graph formed by the role hierarchy, any path starting at r' passes through r.*

Definition 12. *Let s be a session activated by a user u. We define $\sigma(s)$ the* administrative scope of a session *s as $\sigma(s) = \cup_{r \in s} \sigma(r)$. The idea expressed by this definition is that the administrative scope of a session is simply the union of administrative scopes of all roles activated in the session s by the user u.*

The concept of administrative scope provides us with a way to divide a role hierarchy graph into sub-hierarchies that form a natural unit of administration for the role *r*. Two approaches are the most favored for performing role-based administration, *ARBAC*97 and administrative scope. Among these two approaches, administrative scope is found to be a more flexible approach [11]. This quality makes it suitable for role-based delegation in a highly dynamic environment that is the cloud system.

Using administrative scope, a user *u* is allowed to delegate a role *r* only if $r \in \sigma(s)$, the administrative scope of the users session. This limits the user to delegate only roles that are within his administrative scope. Administrative scope can be used to regulate who can receive a delegation. A user *v* is authorized to receive a delegation of a role

r from user u if for all $r' < r$ such that $r' \notin \sigma(s)$, there exists r'' such that $r' \leq r''$ and $(v, r') \in UDA$. In order words, the delegatee v is authorized to receive the delegation of a role r if v is already authorized for all role $r' < r$. In order for the delegatee to receive a delegation, the idea is to require him to be already assigned to any roles outside the delegators administrative scope that the delegatee will inherit by successfully receiving the delegation.

4 Chaining Delegation and Transfer

A user that has previously received some access rights through a delegation process may decide to further delegate those rights. This can result in what can be called a chained delegation. In this delegation chain the trustworthiness of users is used by each node as a factor to decide about the level of delegation. These trust values form a trust chain. We use the notion of trust graph expressed in [20] to formalize the notion of trust chain in a given context. A trust graph $T = (N, E)$ is a directed acyclic graph that is defined in terms of a set N of nodes and a set E of edges. Nodes represent entities in the delegation chain, while edges represent trust relationship between these entities. These entities could be either users or roles respectively for user delegation and role delegation.

Using trust graphs node n_i trusting node n_j can be represented by (n_i, n_j). The degree to which an entity n_i trusts n_j is represented by a weight denoted by $w(n_i, n_j)$, $0 < w(n_i, n_j) \leq 1$, on the edge (n_i, n_j). In order to control the propagation of the access right during a delegation process, every entity puts a constraint on its trust relationship. This trust constraint is denoted by $c(n_i, n_j)$ where $0 < c(n_i, n_j) \leq 1$. Access rights can be propagated only if $c(n_i, n_j) \leq w(n_i, n_j)$. This implies that delegation cannot be carried along every edge of the trust graph. Therefore identifying valid paths to carry on a delegation becomes a challenging problem.

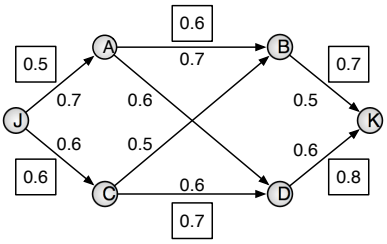

Fig. 3. Example of a trust graph

We consider the trust graph represented in Fig. 3. The numbers that are within boxes associated with the edges denote the trust values on the corresponding edges of the graph while the numbers that are not inside boxes denote the trust constraints. Assume that the entity J desires to identify all the valid paths it can use to delegate some access rights to entity K. Using the trust values and the trust constraints, the first challenge is to

determine all valid paths and thereafter to compute the transitive trust value for each of the found valid paths from J to K. We use three operators on trust values – a *comparison* operator to compare the trust value and the trust constraint, a *parallel* operator to find the minimum of two trust values, and a *sequential* operator that gives the product of two trust values.

Definition 13. *Comparison Operator* $\ominus : [0,1] \times [0,1] \rightarrow 0,1$ *is a binary operation whose input is a trust value and a trust constraint, and that returns* 1 *when the trust value is greater or equal to the trust constraint. Otherwise, the operator returns* 0.

Definition 14. *Parallel Operator, denoted* $\oplus : [0,1] \times [0,1] \rightarrow [0,1]$*, is a binary operator whose inputs are two trust values, and that returns the minimum of the two trust values.*

Definition 15. *Sequential Operator, denoted* $\otimes : [0,1] \times [0,1] \rightarrow [0,1]$*, is a binary operator whose inputs are two trust values, and that returns the product of these two trust values.*

The comparison operator is used to identify all the valid paths in a trust graph. Those valid paths are the ones able to carry on the delegation from J to K. The checking of valid paths may return either multiple valid paths, or a single valid path. In either case, we need to compute the transitive trust value. The parallel operator is used to compute the transitive trust value in case of multiple valid paths while the sequential operator is used for the same purpose but in case of a single valid path.

These operators are used within algorithm 1 to find the transitive trust value. The algorithm works as follows. Using a set of edges of the trust graph given as input, the algorithm uses the comparison operator on each pair of edges that form an arc to compare its trust values to its trust constraint. All arcs whose trust value is at least equal to the trust constraint are retained. If after being linked those arcs form one or more paths from the source node to the destination node, then those paths are considered to be the valid paths. If the algorithm find multiple valid paths, it uses the parallel operator to find the path with the minimum transitive trust value. Otherwise, the algorithm uses the sequential operator to return the transitive trust value of the unique path. In case of multiple valid paths, we made the design decision to choose the path with the minimum transitive trust value in order to be more conservative and more protective. Choosing the path with a maximum transitive trust value is also a valid choice, but is not considered in this work.

To see how the propagation of delegation works, we use the trust graph in Fig. 3 and execute our algorithm in order to find the transitive trust value from node J to node K. Before running the algorithm on the given graph, a processing step is necessary. This preprocessing involves doing a Depth-First Search on the graph in order to collect all the arcs from nodes J to K. The result of the DFS is as follows. $DFS(G) = (J,C),(C,D),(D,K),(C,B),(B,K),(J,A),(A,D),(A,B)$. This list of edges is provided to the algorithm as input.

Algorithm 1. Algorithm to identify all valid paths for delegation propagation

Input: edges $e_1, ..., e_n$
Output: set of valid paths and transitive trust on the valid paths
$valid\_paths \leftarrow \{\}$
for all i : 1 ≤ i ≤ n **do**
  $comp\_trust_i \leftarrow 0$
end for
for all $i : 1 \leq i \leq n$ **do**
  $compt\_trust_l \leftarrow w(e_i) \ominus c(e_i)$
  if $comp\_trust_l = 1$ **then**
    $valid\_paths \leftarrow (e_i)$
  end if
end for
if |Valid_paths| > 1 **then**
  Let $valid\_paths = \{(n_1, n_{2_1}, \ldots, n_{k-1_1}, n_{k_1}), (n_1, n_{2_2}, \ldots$
  $n_{k-1_2}, n_{k_2}), ..., (n_1, n_{2_j}, ..., n_{k-1_j}, n_{k_j})\}$
  min ← 1;
  for all $l : 1 \leq l \leq j$ **do**
    $trans\_trust_l \leftarrow 0$
  end for
  for all $l : 1 \leq l \leq j$ **do**
    for all $i : 1 \leq i \leq (k-2)$ **do**
      $trans\_trust_l \leftarrow trans\_trust_l + w(n_i, n_{i+1}) \otimes w(n_{i+1}), n_{i+2})$
    end for
  end for
  for all $l : 1 \leq l \leq j$ **do**
    $min \leftarrow min \oplus (trans\_trust)_l$
  end for
end if
if | $valid\_paths$ |= 1 **then**
  Let $valid\_paths = \{n_1, n_2, \ldots, n_k\}$
  $trans\_trust_l \leftarrow 0$
  for all $i : 1 \leq i \leq (k-2)$ **do**
    $trans\_trust_l \leftarrow trans\_trust_l \times w(n_i, n_{i+1}) \otimes w(n_{i+1}, n_{i+2})$
  end for
end if
return $valid\_paths, trans\_trust_l$

Given the list of arcs, we run the algorithm as follows.

- (J,C) : w(J,C) ⊖ c(J,C) = 0.6 ⊖ 0.6 = 1 valid_paths = {(J,C)}
- (C,D) : w(C,D) ⊖ c(C,D) = 0.7 ⊖ 0.6 = 1 valid_paths = {(J,C),(C,D)}
- (D,K) : w(D,K) ⊖ c(D,K) = 0.8 ⊖ 0.6 = 1 valid_paths = {(J,C),(C,D),(D,K)}
- (C,B) : w(C,B) ⊖ c(C,B) = 0.6 ⊖ 0.5 = 1 valid_paths = {(J,C),(C,D),(D,K),(C,B)}
- (B,K) : w(B,K) ⊖ c(B,K) = 0.7 ⊖ 0.5 = 1 valid_paths = {(J,C),(C,D),(D,K),(C,B),(B,K)}
- (J,A) : w(J,A) ⊖ c(J,A) = 0.5 ⊖ 0.7 = 0 valid_paths = {(J,C),(C,D),(D,K),(C,B),(B,K)}
- (A,D) : w(A,D) ⊖ c(A,D) = 0.4 ⊖ 0.6 = 0 valid_paths = {(J,C),(C,D),(D,K),(C,B),(B,K)}
- (A,B) : w(A,B) ⊖ c(A,B) = 0.6 ⊖ 0.7 = 0 valid_paths = {(J,C),(C,D),(D,K),(C,B),(B,K)}
- Finally valid paths = {J,C,D,K}, {J,C,B,K}

Since we have found two valid paths, we will run the first if statement of the algorithm as follows.

- $trans\_trust_1 = 0 + 0.6 \otimes 0.7 \otimes 0.8 = 0.336$
- $trans\_trust_2 = 0 + 0.6 \otimes 0.6 \otimes 0.7 = 0.252$
- $min = trans\_trust_1 \oplus trans\_trust_2 = 0.336 \oplus 0.252$

$trans\_trust_2 = 0.252$ is the minimum transitive trust value. Therefore the valid path J,C,B,K is the one that will be used to propagate the delegation of access rights from J to K.

5 Conclusion

In this work, we formalized a trust-based access control model for providing fine-grained access in smartphone clouds. The model extends traditional RBAC with the notion of trust and also addresses the problem of trustworthy delegations. A lot of work remains to be done. We plan to analyze this model and detect inconsistencies and errors in the policy specification. The access control configuration of the cloud is dynamic in nature. Towards this end, we plan to investigate how to perform access control analysis in real-time to ensure that security breaches do not occur. We also plan to deploy this model in real-world environments and study its impact on the performance and usability of cloud applications.

Acknowledgment. This work was supported in part by a fellowship from AFRL Information Institute under the 2012 Summer VFRP. The authors thank Mr. John Graniero AFRL/RIB for his support of this research. The views and conclusions are those of the authors, and should not be automatically interpreted as representing official policies, either expressed or implied, of the Air Force Research Laboratory or other federal government agencies.

References

1. Baracaldo, N., Joshi, J.B.D.: A Trust-and-Risk Aware RBAC Framework: Tackling Insider Threat. In: Proceeding of the Symposium on Access Control Models and Technologies, Newark, NJ (June 2012)
2. Barka, E., Sandhu, R.: A Role-Based Delegation Model and Some Extensions. In: Proceedings of the 16th Annual Computer Security Applications Conference, New Orleans, Louisiana, USA (December 2000)
3. Barka, E., Sandhu, R.: Framework for Role-Based Delegation Models. In: Proceedings of the 23rd National Information Systems Security Conference, Baltimore, Maryland, USA (October 2000)
4. Bhatti, R., Bertino, E., Ghafoor, A.: A Trust-based Context-Aware Access Control Model for Web-Services. In: Proceedings of the IEEE International Conference on Web Services (ICWS 2004), pp. 184–191. IEEE Computer Society, San Diego (2004)
5. Bobba, R., Fatemieh, O., Gunter, C.A., Khurana, H.: Using Attribute-Based Access Control to Enable Attribute-Based Messaging. In: Proceedings of the Annual Computer Security Applications Conference, Miami Beach, FL (December 2006)
6. Bonati, P., Samarati, P.: A Unified Framework for Regulating Access and Information Release on the Web. Journal of Computer Security 10(3), 241–272 (2002)
7. Chakraborty, S., Ray, I.: TrustBAC: Integrating Trust Relationships into the RBAC Model for Access Control in Open Systems. In: Proceedings of the 11th ACM Symposium on Access Control Models and Technologies, Lake Tahoe, CA (June 2006)
8. Chen, L., Crampton, J.: On Spatio-Temporal Constraints and Inheritance in Role-Based Access Control. In: Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security, Tokyo, Japan (March 2008)
9. Cheng, P.C., Rohatgi, P., Keser, C., Karger, P.A., Wagner, G.M., Reninger, A.S.: Fuzzy Multi-Level Security: An Experiment on Quantified Risk-Adaptive Access Control. In: Proceedings of 27th IEEE Symposium on Security and Privacy, Oakland, CA (May 2007)

10. Crampton, J., Khambhammettu, H.: Delegation in Role-Based Access Control. In: Gollmann, D., Meier, J., Sabelfeld, A. (eds.) ESORICS 2006. LNCS, vol. 4189, pp. 174–191. Springer, Heidelberg (2006)
11. Crampton, J., Loizou, G.: Administrative Scope: A Foundation for Role-Based Administrative Model. ACM Transaction on Information and System Security 6(2), 201–231 (2003)
12. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed NIST Standard for Role-Based Access Control. ACM Transactions on Information and Systems Security 4(3), 224–274 (2001)
13. Kandala, S., Sandhu, R., Bhamidipati, V.: An Attribute Based Framework for Risk-Adaptive Access Control Models. In: Proceedings of the 5th International Conference on Availability, Reliability and Security, Vienna, Austria (August 2011)
14. McGraw, R.W.: Risk-Adaptable Access Control. In: Proceedings of the 1st NIST Privilege Management Workshop, Gaithersburg, MD (September 2009)
15. Mell, P., Grance, T.: The NIST Definition of Cloud Computing. NIST Special Publication 800-145 (September 2011), `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`
16. Ray, I., Ray, I., Chakraborty, S.: An Interoperable Context Sensitive Model of Trust. Journal of Intelligent Information Systems 32(1), 75–104 (2009)
17. Sandhu, R., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role Based Access Control Models. IEEE Computer 29(2), 38–47 (1996)
18. Tamassia, P., Yao, D., Winsborough, W.: Role-Based Cascaded Delegation. In: Proceedings of the 9th ACM Symposium on Access Control Models and Technologies, Yorktown Heights, New York, USA (June 2004)
19. Toahchoodee, M., Abdunabi, R., Ray, I., Ray, I.: A Trust-Based Access Control Model for Pervasive Computing Applications. In: Gudes, E., Vaidya, J. (eds.) Data and Applications Security XXIII. LNCS, vol. 5645, pp. 307–314. Springer, Heidelberg (2009)
20. Toahchoodee, M., Xie, X., Ray, I.: Towards Trustworthy Delegation in Role-Based Access Control Model. In: Samarati, P., Yung, M., Martinelli, F., Ardagna, C.A. (eds.) ISC 2009. LNCS, vol. 5735, pp. 379–394. Springer, Heidelberg (2009)
21. U.S. Department of Defense: Trusted Computer System Evaluation Criteria. Department of Defense Standard DOD 5200-28-STD (December 1985)
22. Wainer, J., Kumar, A.: A Fine-Grained, Controllable, User-to-User Delegation Method in RBAC. In: Proceedings of the 10th ACM Symposium on Access Control Models and Technologies, Stockholm, Sweden (June 2005)
23. Ya-Jun, G., Fan, H., Qing-Guo, Z., Rong, L.: An Access Control Model for Ubiquitous Computing Application. In: Proceedings of the 2nd International Conference on Mobile Technology, Applications and Systems, Guangzhou, China (November 2005)
24. Zhang, L., Ahn, G.J., Chu, B.T.: A Rule-Based Framework for Role-Based Delegation and Revocation. ACM Transaction on Information and System Security 6(3), 404–441 (2003)
25. Zhang, X., Oh, S., Sandhu, R.: A Flexible Delegation Model in RBAC. In: Proceedings of the 8th ACM Symposium on Access Control Models and Technologies, Como, Italy (June 2003)