I Was Confused: Robust Accountability for Permission Delegation in Cloud Federations

Samane Abdi
Insight Centre for Data Analytics
Department of Computer Science
University College Cork, Ireland
Email: samane.abdi@insight-centre.org

Abstract—Cloud federations allow cloud providers to mitigate the limitation of local resources, enabling service provision for more customers. They allow a provider to either share its unused resources with providers of other clouds or request resources from them when its own resources are not enough to respond to customer demand. Thus, access control is required to manage access to those shared resources. The service provider determines who can access cloud resources by delegating permissions for that purpose. Breakdown in accountability can occur in delegation of permissions among multiple clouds because of ambiguity regarding the interpretation of permission specifications. This paper discusses the breakdown in accountability for delegated permissions in cloud federations, and then demonstrates a trust model to achieve breakdown-robust accountability.

Keywords—Federation, Cloud, Accountability, Permission, Open environments.

I. INTRODUCTION

Cloud computing is a new paradigm in which applications, data, and IT resources are provided to customers as services in an open manner (over the Internet) rather than running locally on the customer’s machine. A cloud enables customers to access these services with a high degree of freedom anywhere and any time. The cloud services range from Infrastructure as a Service (IaaS) to Platform as a Service (PaaS) and Software as a Service (SaaS) [1], [2]. IaaS offers infrastructure services, such as compute clouds, cloud storage, message queues, etc. PaaS offers complete platforms, solution stacks and execution environments, while SaaS is a software delivery model driven by a multi tenancy architecture. In order to be more beneficial and productive, clouds can share their resources among multiple service providers of different clouds while guaranteeing that each service provider has enough resources to achieve its best performance. This can be achieved through cloud federations [3].

A cloud federation is a collaboration among service providers of different clouds to share their resources to take advantage of aggregation and produce an enlarged computing utility. For example, a cloud may share its resources with service providers of other clouds when it has resources which are beyond the needs of its own customers. Similarly, a service provider from one cloud may request resources from another cloud when its workload cannot be satisfied by its own cloud resources. In this way, both service providers benefit because one can obtain more customers, and the other will retain its existing customers.

Despite the potential gains achieved from cloud federations, there are security concerns regarding access to cloud resources [4], [5]. Because of the cloud federation characteristics such as large amounts of distributed resources, highly dynamic and flexible infrastructures, and lots of distributed customers, a decentralized approach for managing access to cloud resources is required. Trust management systems are intended to provide a decentralized approach and are suitable to address access control for cloud resources when they establish federations. In this approach, permissions are delegated from the owner of resources of one cloud to the service provider of another cloud rather than controlling directly the access to cloud resources, thereby forming a chain of delegations. Delegation chains provide permission evidence for accessing a resource, and also ensure the accountability for the delegated permission. Breakdown in accountability may arise when there is not a unique interpretation for a permission specification. Thus, delegation chains which are used by service providers to verify their access to cloud resources may not reflect the accountability for all delegators in the chain. Many existing trust management systems such as PolicyMaker [6], KeyNote [7], and RT [8] use ad-hoc strategies to ensure that each permission is specified sufficiently in the sense of knowing who originated the permission. The origin of a permission does not necessarily reflect the accountability for that permission when there is ambiguity regarding a permission specification. In receiving two identical permission specifications the delegator may be confused in inferring the proper accountability for further delegation. Therefore, permission accountability should not rely on ad-hoc strategies; rather a systematic way of providing accountability is required. Thus, a reliable mechanism is required to provide a global unique interpretation for a permission specification that also clearly references the originator of that permission. A policy language was introduced for trust management that provides a systematic way of providing global unique permission specifications [9]. In this paper, we use the approach of providing global unique permission specifications to achieve robust accountability of delegated permissions in cloud federations.

The paper is structured as follows: section II discusses the problem that results in breakdown of accountability for delegated permissions when more than two clouds wish to federate. Section III and IV deal with how to prevent the problem and provide proper accountability. Section V demonstrates the use of the proposed trust management for secure cloud federations. Section VI outlines the current implementation of the trust model. Related works are discussed in section VII. Finally, in
In this section, we introduce an example to explain the ambiguity regarding permission specification that results in breakdown in accountability for that permission when different clouds establish a federation. Note that, accountability refers to tracking of a principal’s activity under proper permission. The permission might be delegated by a principal who must also be held responsible for the actions activated by that permission. Because of the decentralized nature of clouds, trust management systems are suitable to manage trust relationships and access control among the principals of a cloud. In trust management systems, certificates (cryptographic assertions) specify delegation of permissions among principals. Principals may further delegate their permissions to other principals. A delegation statement indicates that the authority for a permission is delegated from one principal to another principal(s). The delegation statement is denoted as \( P \xrightarrow{X} Q \), whereby principal \( P \) signs a statement that it authorizes principal \( Q \) for permission \( X \). Principals are uniquely identified by their public keys whereas the uniqueness of permissions are not addressed in a systematic way. A principal is considered to be accountable for a permission, if it accepts responsibility for how the permission is used by other principals. Ambiguity regarding the unique interpretation of a permission can result in confusion of the principal who is considered to be accountable for that permission. Suppose that cloud \( A \) wants to federate with cloud \( B \) and cloud \( M \) to use their data storage resources. All clouds use trust management for controlling access to their resources. The service provider of cloud \( B \), identified by its public key \( k_B \), issues a delegation statement enabling cloud \( A \)'s service provider, identified by its public key \( k_A \), to access data storage space at cloud \( B \). This is denoted by statement

\[
c_1 : k_B \xrightarrow{dataStorage} k_A
\]

On the other hand, suppose that the malicious service provider for cloud \( M \), identified by its public key \( k_M \), intercepts the delegation statement issued by \( k_B \) (\( c_1 \)), and replaces it by the following delegation statement issued by \( k_M \):

\[
c_2 : k_M \xrightarrow{dataStorage} k_A
\]

The service provider of cloud \( A \) does not realize that it received the same permission specification and therefore leads it to believe that permission \( dataStorage \) is related to accessing data storage resources at cloud \( M \). As a consequence, the service provider at cloud \( A \) grants access to the data storage space of cloud \( M \) to its customer, identified by public key \( k_C \) denoted as:

\[
c_3 : k_A \xrightarrow{dataStorage} k_C
\]

However, the customer \( k_C \), colluding with the malicious service provider for cloud \( M \), can use the statements \( c_1 \) and \( c_3 \) as proof of authorization to access the data storage space of cloud \( B \). Regarding accountability, when cloud \( A \) issues the statement \( c_3 \) (considered as the accountable principal for delegated permission in \( c_3 \)), it believes that the statement \( c_2 \) provides the appropriate accountability for cloud \( M \) (as a track record of accountability for delegated permissions). However, \( k_A \) is confused and should not be held accountable for the inadequacy in permission specification that is identified by the service provider for cloud \( B \). One may argue that this breakdown in accountability could be prevented by adding...
extra information about the originator to the permission specifications. For example, the permission \(\text{cloudB/dataStorage}\) is clearly related to its originator. However, a malicious service provider at cloud \(M\), may intercept the statement:

\[ c4 : k_B^{\text{cloudB/dataStorage}} \rightarrow k_A \]

and issue a delegation statement to delegate permission \(\text{cloudB/dataStorage}\) to cloud \(A\), as:

\[ c5 : k_M^{\text{cloudB/dataStorage}} \rightarrow k_A \]

Cloud \(A\) does not realize that cloud \(M\) does not have any authority over cloud \(B\) resources and in further delegation to its customer \(k_C\), issues the following statement as:

\[ c6 : k_A^{\text{cloudB/dataStorage}} \rightarrow k_C \]

The service provider at cloud \(A\) claims that it cannot be held accountable for the confusion when the service provider \(k_C\) uses the delegation statements \(c4\) and \(c6\) to use the data storage of cloud \(B\). Within the demonstrated scenarios, it is impossible to identify the accountable principal when there is ambiguity regarding permission specifications.

### III. Accountability for Delegated Permission

The notion of local permission that was introduced in [10], [11] ensures that despite the fact that permissions are identified arbitrarily, they are globally unique. This prevents the resource owners from issuing ambiguous permissions. The local permission \(\langle P \text{ Perm} \rangle\) identifies a permission specification named locally as \(\text{Perm}\) in the name space of cloud service provider \(P\). The permission \(\text{Perm}\) is bound to the statement signed by \(P\)’s public key \((k)\), \([\text{Perm}]_k\), as a reference to a global context. This provides a globally unique interpretation for permission specifications to prevent ambiguity. Therefore, principals such as a cloud service provider receiving two identical permission specifications cannot misuse the permission or get confused and use them for non-intended purposes. Unambiguous interpretation for each permission makes the originator of that permission accountable for any actions enabled by that permission \(\langle P \text{ Perm} \rangle\). For example, by originating a permission for granting access to its data storage space, the service provider for cloud \(B\) (owner of public key \(k_B\)) is implicitly accepting accountability for the use of this storage space, where: \(k_B \triangleright (k_B \text{ dataStorage})\). Moreover, in existing trust management systems the set of permissions implicitly have a globally defined pre-order relation. Local permissions are specified locally and an originator must explicitly define how the permissions which are originated locally, relate to other permissions globally. An ordering relation as \(X \rightarrow Y\) is explicitly defined between permissions \(X\) and \(Y\); where permission \(X\) is dominated by permission \(Y\). In other words, a principal that is authorized for the resources enabled by permission \(Y\) is considered to be authorized for resources enabled by permission \(X\). For example, the service provider for cloud \(B\) originates a local permission \(\langle k_B \text{ federation} \rangle\) authorizing federation with another cloud. It asserts that anyone authorized for federation has authority to use its cloud’s data storage space, denoted as:

\[ \langle k_B \text{ dataStorage} \rangle \rightleftharpoons \langle k_B \text{ federation} \rangle \]

This represents a policy that is local to cloud \(B\) and is globally interpretable.

Moreover, a principal \(Q\) may accept accountability for the permission \(\langle \text{Perm} \rangle\) by signing a statement to that effect. However, the principal asserting accountability must be authorized for the permission in the first place. This prevents a malicious principal claiming accountability for a permission (enabling access to a resource) that it is not trusted. For the example in Figure 1, \(k_M\) asserts that it accepts accountability for \(\langle k_B \text{ dataStorage} \rangle\), however, \(k_M\) is not authorized for \(\langle k_B \text{ dataStorage} \rangle\) and therefore, \(k_A\) cannot deduce that \(k_M\) is accountable for \(\langle k_B \text{ dataStorage} \rangle\), denoted as: \(k_M \triangleright (k_B \text{ dataStorage})\)

### IV. Compliance Checking for Accountability

Compliance checking is at the heart of a trust management system [6]. The inputs for a compliance checker are a request, a set of certificates, and a policy. The compliance checker checks whether a set of certificates proves that a requested action complies with the policy. Compliance checking in current trust management systems corresponds to the query (by principal \(P\)): is the requester \(r\) authorized to access the resources \(\text{authorized by permission \(X\)}?\) This is evaluated by verifying that the request is supported by a set of certificates that complies with the policy. Thus, accountability is not addressed directly in current compliance checking mechanisms for trust management.

In our approach, if principal \(P\) originates permission \(X\) then it is deduced that principal \(P\), as the originator of non-ambiguous permission \(X\), is accountable for that permission \((P \triangleright X)\). However, in the case that \(P\) is not the originator of permission \(X\), principal \(P\) should determine that some principal \(R\) can be held accountable for actions associated with the permission \(X\). This needs to be determined before signing a delegation statement to delegate permission \(X\) to other principals. This is evaluated by determining whether principal \(R\) (and earlier principal in the delegation chain) is accountable for permission \(X\), denoted as \((R \triangleright X)\), and that \(P\) trusts \(R\) to provide accountability. If the check succeeds then \(P\) delegates permission \(X\) to other principals. We define two compliance checking queries for determining the accountable principal: one accountability for access to a resource, and the other accountability for further delegation of permissions.

Compliance checking for accountability regarding accessing a resource owned by principal \(P\) corresponds to the query, CK1: is requester \(r\) allowed to access the resource \(\text{authorized by permission \(X\)} that principal \(R\) is accountable?\) This is evaluated by determining whether the requester \(r\) can prove (with a set of certificates) that it holds permission \(X\) and an accountable principal for permission \(X\) can be inferred.

Similarly, compliance checking for accountability regarding further delegation of permission \(X\) corresponds to the query (by principal \(P\), CK2: is principal \(R\) accountable for permission \(X\), that principal \(Q\) delegated to principal \(P\)? This is evaluated by determining whether principal \(R\) is accountable and whether \(P\) trusts \(R\) to provide accountability. We assume that if a principal is trusted for some permission, then any assertion that the principal makes for accepting accountability for that permission is also trusted.
V. MANAGING A CLOUD FEDERATION USING TRUST MANAGEMENT

Trust management can be used to manage the trust relationships among different clouds for securely sharing their resources in establishing a federation. Suppose that the service provider for cloud $A$ (identified by public key $k_B$) in Figure 2 uses trust management for controlling access to its resources. When a request from an untrusted customer is made to access cloud $A$’s resources, the trust management helps the service provider of cloud $A$ in making well-founded access decisions. Suppose, the PaaS service provider for cloud $A$, $S$ (identified by its public key $k_S$) is unable to instantiate further resources for its customers as all cloud $A$’s resources are in use. In order that the service provider for cloud $A$ will be able to continue providing service to its customers, cloud $A$ decides to federate with cloud $B$. Thus, the federation agent in cloud $A$ ($FA$), identified by public key $k_{FA}$, sends a signed request for federation including the IP address ranges of cloud $A$, to the federation agent in cloud $B$ ($FB$), identified by public key $k_{FB}$ as in the following:

\[
\text{msg 1: } FA \rightarrow FB : \\
\{ \text{federation,IPrange = 192.168.1.1 – 192.168.1.100} \}_{k_{FA}}
\]

Note that, all service providers for cloud $A$ are in this IP range. In addition, along with the request, $FA$ sends the delegation certificates to prove that it can be trusted for using cloud $B$’s resources:

\[
\text{msg 2: } FA \rightarrow FB : \\
\{ k_B^{\langle \text{federation,IPrange} \rangle} \}_{k_{FA}}
\]

The federation agent for cloud $B$ confirms the signature on the message $msg$ 1 from the requester $k_{FA}$ and, if valid, then it queries the trust management as to whether the public key $k_{FA}$ is trusted to federate with cloud $B$.

As a consequence of a successful query, cloud $A$ federates with cloud $B$ and the resources of cloud $B$ can be shared with cloud $A$. It is worth noting that the shared resources are then within cloud $B$.

After establishing federation, the PaaS service provider $S$ (identified by its public key $k_S$) for cloud $A$ sends a signed request including its IP address (which is in the range of IP addresses of cloud $A$) for using the data storage space of cloud $B$:

\[
\text{msg 3: } S \rightarrow FB : \\
\{ \text{dataStorage, 192.168.1.50} \}_{k_S}
\]

The service provider for cloud $B$ defines the following local policy that any principal that is authorized for federation in the mentioned IP range is also authorized to access the data storage space:

\[
LP_B : \langle k_B \text{ dataStorage} \rangle \rightsquigarrow \langle k_B \text{ federation,IPrange} \rangle
\]

Service provider $S$ wishes to authorize cloud $A$’s customer $C$ (identified by its public key $k_C$) to use the data storage of cloud $B$. Before signing any delegation statement to customer $C$, service provider $S$ should determine that some earlier principals in the chain of delegations can be held accountable for actions associated with the permission $\langle k_B \text{ dataStorage} \rangle$. The service provider for cloud $B$ is the originator of permission $\langle k_B \text{ dataStorage} \rangle$ and therefore is held accountable. Then service provider $S$ defines its own local policy and originates a permission that explicitly identifies how the permission that it originates is related to the access permission for cloud $B$’s data storage space. The service provider $S$ asserts:

\[
LP_S : \langle k_B \text{ dataStorage} \rangle \rightsquigarrow \langle k_S \text{ dataStorage} \rangle
\]

and authorizes customer $C$ for the data storage space:

\[
k_S^{\langle k_S \text{ dataStorage} \rangle} \Rightarrow k_C
\]

Thus, when customer $C$ requests cloud $B$’s data storage space, it presents a set of certificates ($LP_B$, and $LP_S$) that
are intended to satisfy the compliance checking query. The compliance checking query evaluates whether it is safe to allow customer \( C \) to access the data storage space at cloud \( B \) for which the service provider \( S \) is held accountable.

Returning to Figure 1, if the service provider for cloud \( A \) is not aware of the statement \( \langle k_B \text{ dataStorage} \rangle \rightarrow \langle k_B \text{ federation.IPrange} \rangle \), then in the presence of a malicious delegation statement \( k_M \langle k_B \text{ dataStorage} \rangle \rightarrow k_{FA} \), the service provider for cloud \( A \) will not be confused that \( M \) is authorized for accessing the data storage space of cloud \( B \) and mistakenly think that the service provider \( k_M \) is accountable. When the service provider for cloud \( A \) searches for an accountable principal, it cannot find any statement that a principal is held accountable to authorize \( k_M \) for the permission \( \langle k_B \text{ dataStorage} \rangle \). Therefore, verification of accountability is unsuccessful. This scenario is depicted in Figure 3.

**VI. IMPLEMENTATION**

An ontology-based approach [12] is used to implement the trust model for accountability. We model the policy engine in the Web Ontology Language (OWL) [13] and the Semantic Web Rule Language (SWRL) [14]. We also use a DL-reasoner (DL: Description Logic) [15] to reason over the asserted knowledge (policy statements and certificates) in the ontology. The federation agent of the local cloud requests to federate with the federation agent placed at the remote cloud. The current implementation provides a user-interface that allows the users to select the request and related certificates manually. Future research will address the design of an automated certificate chain discovery system whereby the set of certificates relative to a request will be discovered from different clouds automatically. The request from the cloud that wishes to federate is interpreted as a query within the Query Interpreter (implemented using Java and SQWRL). The trust management system is used by the agent to determine whether a requester is authorized for the cloud resources and who is accountable for the associated permissions. If both authorization and accountability checks succeed then the request for federation and consequently for sharing of resources is granted. Figure 4 depicts the implementation components.

**VII. RELATED WORK AND DISCUSSIONS**

Using policy based trust management systems provides a systematic security mechanism for automatic trust decisions regarding the open federations of different clouds. This is more controlled systematic way in contrast with reputation based trust management systems. A variety of trust management systems have been developed over the years to address the requirement for constructing trust and managing authorization without relying on a central authority. Many existing trust management systems such as PolicyMaker, KeyNote [7], SPKI/SDSI [16], [17], and RT [8], [18] are designed to specify arbitrary permissions. They assume unique and unambiguous permission names are provided by using a global name provider’s services. For example, X.509 [19] uses X.500 naming service, the KeyNote trust management system uses the Internet Assigned Number Authority (IANA) [20]. In addition, Role-based Trust management (RT) uses Application Domain Specification Documents (ADSDs) [8] to ensure the globally unique naming. Although, global name providers provide a unique interpretation for each name, the service providers may still use arbitrary names to represent their own cloud resources. It depends on the experience of the service provider administrator who originates the permissions to specify non ambiguous permissions for cloud resources and therefore accountability for those permissions without
any excuse for confusion. However, providing accountability for delegation of permissions should not rely on the ad-hoc strategies; it should be formalized in a systematic way.

Therefore, using local permission provides a reliable name schema for globally unique interpretations for permissions. Using local permissions, none of the service providers have excuse for confusion that is a result of inadequate information in permissions specifications. Thus, preventing ambiguity provides a reliable scheme for robust accountability for permission delegations in cloud federations.

VIII. CONCLUSIONS

Cloud federations offer effective distributed sharing of resources but require mechanisms to control access and provide accountability for the resources. In this work, we designed and implemented a trust management for sharing resources in a cloud federation and demonstrated an approach for robust accountability in the use of permissions. Permissions are delegated among the service providers and customers of clouds to allow each other access to non-local resources in cloud federations. By using unambiguous permission specifications, there is no confusion regarding the accountability for the permissions. The proposed compliance checking provides a means of determining accountability, and therefore can be used in preventing unauthorized access to the cloud resources.

ACKNOWLEDGMENT

This research was supported by Science Foundation Ireland under grant 08/SRC/11403. The author would like to thank Dr. John Herbert for critical reading of this manuscript, and Dr. Simon N. Foley for instigating this work.

REFERENCES


