Managing Identity and Authorization for Community Clouds *

Jeff Chase and Prateek Jaipuria
Department of Computer Science
Duke University

Steve Schwab and Ted Faber
USC/ISI

August 21, 2012

Technical Report CS-2012-08
Department of Computer Science, Duke University

Abstract

A community cloud operates to serve multiple organizations who have entered into sharing arrangements with one or more cloud providers. Members of the participating organizations may also collaborate on shared projects, which may lead them to exercise shared control over virtual machines or other cloud-hosted resource instances. Software running in the cloud instances may serve the community members or act on their behalf. For these reasons a flexible framework for identity and authorization is essential for community clouds.

This paper gives an overview of the trust framework adopted in NSF's GENI project, which may be viewed as a multi-provider infrastructure cloud serving a community of researchers with various institutional and project affiliations. The authorization framework for GENI combines elements of existing solutions to related challenges: Web-based Single-Sign On and federated identity management (e.g., Shibboleth), virtual organizations in grid computing, access control based on roles or attributes, and public key cryptosystems with delegated trust and proxy certificates. The GENI solution uses a form of Attribute-Based Access Control (ABAC) incorporating a trust management logic and authorization policy language called Role-based Trust (RT).

1 Introduction

In the NIST Cloud Taxonomy, a *community cloud* serves multiple participant organizations that request and consume cloud services [20]. The concept of a community cloud raises the question of how providers authenticate consumers (users), qualify them for service, and hold them accountable for their actions in the cloud. A user's privilege at a provider is based on membership in organizations, roles and status within organizations, and relationships and agreements among organizations and providers, and community policies and provider policies for authorization and resource management. These affiliations, roles, relationships, and policies may change frequently. In contrast commercial clouds may serve any user who can pay.

Many elements of a solution can be found in the research literature and in the existing practice of systems facing similar challenges, including services for multi-institutional research collaboration and resource sharing.

*This paper is based upon work supported by the US National Science Foundation through the GENI Initiative and under NSF grants OCI-1032873 and CNS-0910653, and by the State of North Carolina through RENCI.

Section 2 summarizes foundational elements of our approach that are common from many previous systems: keypairs, roles, groups, and external identity services. Section 5 discusses related authorization solutions for grid computing and other federated systems in more detail.

This paper presents a trust framework for federated community clouds, combining these common elements with a general trust management system incorporating logic-based authorization and inference. Our solution is suited to the needs of federated multi-provider systems that serve user communities spanning multiple organizations.

One example of such a system is NSF's GENI initiative (Global Environment for Network Innovation), a suite of infrastructure to support research in network science and engineering. Our approach was developed with support from GENI and NSF's Trustworthy Computing program, and has been adopted in the GENI architecture. GENI accommodates multiple providers offering diverse virtual infrastructure services or other virtual resources. We illustrate the trust system by describing a prototype implementation developed for initial use within the ExoGENI testbed [3] being deployed for GENI. ExoGENI is a network of cloud provider sites—privately administered clouds based on OpenStack software—on multiple campuses, linked using network circuit fabrics and the ORCA control framework [2]. Although this paper introduces some GENI terminology, the concepts and trust management solution generalize to other related systems.

Some elements of the trust management problem are specific to advanced cloud platforms such as GENI. Cloud services enable their users to assemble sets of virtualized resources that run user-specified software. In GENI these execution contexts are called *slices*. A slice may span multiple participating providers. A key objective of our cloud trust architecture is to provide proper authorization, accountability, and trust management for slices. Like a grid job, a slice is a unit of activity that consumes resources under user direction, and may interact with other entities by sending and receiving messages. However, a slice may host a long-running activity that changes over time, and may be controlled by multiple users over time. Other entities may assign trust to a slice based on who controls it, what software it is running, and/or which providers host it. The resources and powers that a provider grants for a slice may depend on the nature and purpose of the activity hosted in the slice. In this paper we focus on how providers authorize operations on slices and infer attributes to drive other policy choices involving slices, such as resource allocation and interconnection.

We find that logic-based trust management is a flexible and powerful solution for trust and authorization in federated community clouds. Our prototype uses *libabac*, an open-source library for *attribute-based access control* (ABAC) based on the RT family of role-based trust logics [19]. We show that our solution based on RT/ABAC trust logic offers several key advantages:

- It supports authorization based on user identity, group affiliations, and the nature of a specific activity (slice).
- It offers general support for flexible delegation of rights including capability-based access control for slices and other global objects.
- It enables flexible declarative authorization policies and delegated policy evaluation combining policy rules from multiple entities in a federated system.
- It captures the trust structure of federated systems in a declarative representation. Key aspects of trust in federated systems reduce to choices about whose assertions to believe, whose commands to accept, or what sensitive information to reveal and to whom. Trust logic offers a powerful formalism for participating servers to represent these choices. Our approach factors these choices out of the control framework software: the federation structure emerges from the combination of local policies, and may be changed without modifying the control software.

For example, providers in a federated cloud are generally autonomous and participate in federations by choice. We show how a provider can use trust logic to represent this choice in local policies that delegate trust to external coordinator entities (authorities) to approve users and slices, e.g., based on endorsements of those

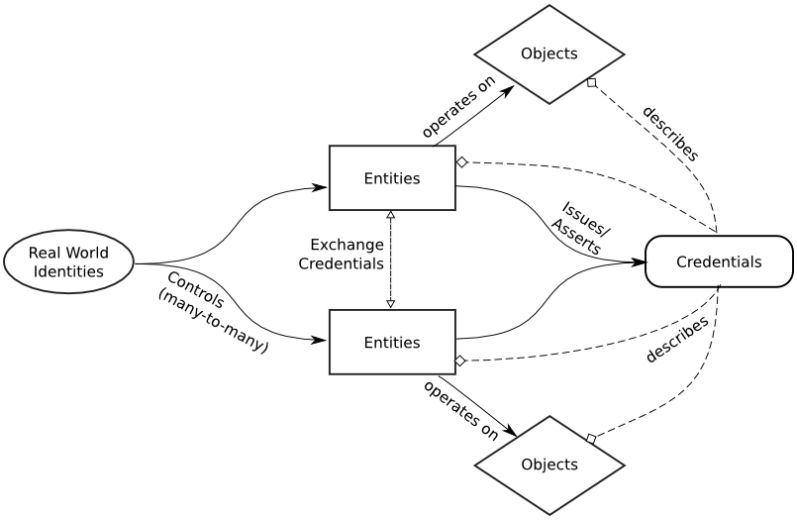

Figure 1: Basic terminology and trust model. Real-world identities such as users, institutions, and enterprises exercise control over software entities, which exchange messages over a network. Entities speak with keypairs and operate on objects. Entities may issue credentials, which are statements of belief about other entities and/or objects, authenticated under the issuer's keypair. For example, a credential may assert roles or attributes of a subject (a public key) or object. Entities make decisions about trust or authorization by reasoning from credentials according to a local policy.

external services by a federation root. Trust structures may extend to complex multi-federation scenarios: for example, a provider may affiliate with multiple federations, or federations may agree to accept one another's members to varying degrees.

2 Background

This section summarizes foundational elements of our approach that are common to many previous systems, and our rationale for adopting and combining them. These elements are authentication of entities based on public keys, credentials as signed endorsements of public keys, authorization based on attributes (roles), group membership represented as roles, and external identity services that assert roles for real-world identities. Figure 1 illustrates some common concepts and terms used in this paper.

2.1 Entities and their keypairs

Asymmetric public-key cryptography is a foundation of our approach. We use the generic term PKI to refer to any system in which principals are represented by software entities that authenticate with public-private keypairs. [1] Generally, an *entity* is a software instance that can send and receive messages under a keypair. An example of an entity is a client tool or server speaking with the private key of a user or organization that controls it. All message exchanges between entities use an authenticated transport (e.g. SSL/TLS or signed messages) so the receiver of any message can verify the sender's public key. In discussing trust we presume without loss of generality that an entity's keypair belongs to that entity alone: if two entities share a private key then they are not distinct with respect to trust.

[1] By our use of the term, PKI does not imply an identity hierarchy based on Certifying Authorities (CA) and Distinguished Names (DN), which is the hallmark of X.509 PKI systems deployed today. Our use of a role-based trust management logic generalizes and replaces a CA hierarchy and global naming, although it also does not preclude their use.

The choice to use PKI is natural for cloud systems. Cloud server APIs commonly allow clients to authenticate using keypairs so that user tools may invoke them outside of a browser session easily ("hands free"). Network servers running inside or outside of a cloud commonly use keypairs to authenticate to their clients, e.g., using SSL/TLS.

2.2 Credentials and delegation

Another benefit of a PKI approach is that it enables one entity to endorse the public key of another without knowing or exposing the private key. An endorsement is a kind of *credential*, and may be issued as a signed certificate, such as an X.509 identity certificate or attribute certificate naming the endorsed entity's public key as a subject.

Signed credentials are often used to represent delegated trust. Generally, a *delegation* is an endorsement or assignment of a role to a *subject* entity's public key by an *issuer* entity. The issuer may publish the delegation for use by other relying parties, or the issuer may store it as part of a local policy.

2.3 Representing identity with roles

Our approach incorporates three additional elements that are common in various forms in related cloud and grid systems. This overview mentions a few of these systems; Section 5 discusses related work in more detail.

- **Roles.** RT/ABAC trust logic uses a *role-based* approach: a decision to trust another entity is determined from roles bound to it. A *role* or *attribute* is a named property or predicate of an entity. Roles simplify policy management. They decouple access control from specific real-world identity or names in a global name space. In essence they provide a level of indirection to access control: the authorizer grants a privilege to all entities possessing some specific role, without having to know in advance the global names of all entities wielding the role. They also enable use of logic-based policies that infer roles from other facts and rules. SPKI/SDSI [12] popularized these ideas in the 1990s.
- **External identity.** Private clouds and community clouds serve members of an enterprise or a community spanning multiple enterprises. It is common today for an enterprise to run an identity service to manage a directory of its users and their roles. The set of users for an enterprise, institution, or organization is an *identity domain* or user domain. For example, in web single sign-on (SSO) systems, each domain runs an *identity provider* (IdP) server: a user authenticates for a web browser session by logging in to the IdP for its home domain, which issues statements about the session to web applications (service providers or SPs) that the browser visits during the session.
- **Groups or projects.** It is often useful to grant privilege to a user based on the user's membership in a group associated with a specific activity. These groups are called *projects* in GENI. A project may involve users from multiple domains. More generally, groups may be nested: groups may have subgroups and may contain other groups.

Importantly, these elements are synergistic. Roles are useful when the entities in a system are frequently changing or managed by another party, such as an external IdP. For this reason modern SSO systems are role-based. Attributes (roles) asserted by an IdP or other attribute authority may represent membership in groups directly.

Shibboleth. For example, many academic institutions use the Shibboleth [21] identity service, which is based on the OASIS SAML standard for web SSO systems. SAML represents identity as a set of attributes (roles); an identifying global name is one kind of attribute, but other attributes are often more useful for authorization (e.g., status as a registered student or faculty member). Shibboleth tools support delegated management of nested groups; attributes on user identities represent membership in groups or rights to control group membership.

Bridging SSO identity and PKI. Although Shibboleth and other SSO services are designed for web systems, an IdP may also issue credentials that bind user attributes to keypairs. Amazon IAM introduces the term *identity broker* to refer to a portal bridging web sign-on to a key-based system. In a simple implementation, a web portal registered as an SP receives browser session attributes from a web SSO IdP, assigns or collects a keypair for the user in a web exchange over the authenticated session, and then issues credentials signed with its own keypair binding those attributes to the user's public key. We adopt a variant of this identity broker approach, described in Section 3.5.

2.4 Federated systems and federated identity

Our approach leverages existing external identity services (e.g., Shibboleth) to support federated infrastructure providers (e.g., GENI) and other cloud systems that serve user communities spanning multiple campuses or other organizations.

Early open-source cloud stacks offered only manual account management: an administrator at each cloud site creates accounts manually for authorized users in the community. External IdPs can offload this burden from the cloud resource providers, just as they factor identity management out of other web-based service providers (SPs). Unifying account management in external IdPs eliminates the redundant burden of managing user accounts at each SP. It is also easier for the users, who can then access the resources through an existing account from a home domain, rather than maintaining user accounts at each SP they use.

Support for external IdPs is especially important for any system whose user community spans multiple identity domains, whether or not the system itself has federated SPs. An SP may trust external IdPs and hold them accountable for managing accounts for users who are remote and unknown to the SP or its operators. Shibboleth and other modern SSO systems have basic support for *federated identity*—the ability for SPs to serve users from multiple identity domains [21]. Shibboleth provides a means for an SP to locate a user's home domain and IdP, and grant trust to IdPs that are endorsed by a federated identity root. For example, the InCommon federation (*incommon.org*) endorses the IdPs of its member institutions; some web SPs are configured to accept users from any InCommon institution.

For these reasons many grid computing systems support external identity management of users and groups (e.g., virtual organizations). Section 5. discusses federated identity and federated grid systems in more detail.

3 Integrating role-based trust logic

Our trust framework for federated community clouds combines role-based trust, external identity services, groups, and flexible delegation with logic-based trust management and a *global object* design pattern suitable for representing slices and groups.

Our solution uses RT2/ABAC as implemented in the *libabac* toolkit. RT/ABAC [19] is a family of trust delegation logics based on safe datalog [8], a simple form of logic program.[2] Figure 2 summarizes the trust logic notation used in this paper. For simplicity we use standard logic syntax rather than introducing the set-oriented syntax used in RT/ABAC.

3.1 The case for trust logic

The key advantage of a trust logic is that it provides a general language for constrained delegations of trust and a logic-based policy framework for reasoning from delegations and local policy rules. A logic is a language

[2]Our current prototype uses a subset of RT2 called RT0 that does not have native support for object parameters. Instead, the RT0-based implementation generates new roles and rules for each new object on-the-fly from templates. RT2 support is recently completed in *libabac*, and will simplify the implementation because it supports roles parameterized to objects, and attributes of objects.

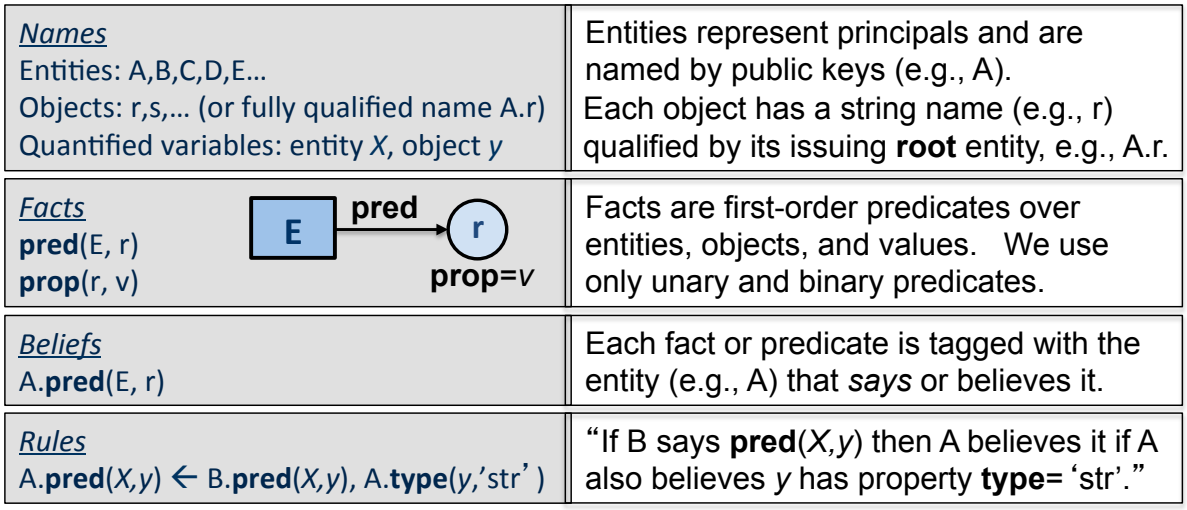

Figure 2: Trust logic syntax used in this paper. We use a restriction of RT2/ABAC, which reduces to a simple form of logic programming (datalog) supplemented with limited constraint features [19]. In this paper we use familiar logic programming syntax rather than the RT syntax. The ← symbol denotes implication. A comma in an inference rule denotes a conjunction of goals (*and*). In trust logic each atomic fact or rule head bears a *says* tag indicating which entity says/believes the fact or rule. Principals (entities) are named by their public keys. Objects are created and named by entities. Values are drawn from common data types, e.g., numbers or strings.

and inference procedure expressing facts (or beliefs) and rules (or policies), and applying rules to facts to derive other facts. The universe of discourse for a trust logic includes the entities themselves: entities make statements about one another. Entities may issue credentials as authenticated statements in the logic. Trust logics incorporate the source of each statement into the inference mechanism: an entity accepts or believes a statement only if it has suitable trust in its issuer.

In RT/ABAC and related trust management systems each entity has local policies that control what trust it places in other entities based on the credentials that it receives. Each entity runs a local inference engine to reason from its base of facts and rules. Trust is granted only if the authorizer can infer some given fact (a *guard predicate*) from its beliefs and policies.

With *libabac* we now have a practical implementation to leverage these advances. It defines a credential format, a logic-based policy language, a means to extend local policies with authenticated policy rules from other entities, and a self-contained linkable inference engine to reason from credential sets and policy rules. RT2 roles may have parameters that name objects or values such as strings or numeric constants or ranges. Thus an RT2 role may express attributes of an object, a binary relationship of a subject and an object, or an attribute with a value (e.g., a name). The *libabac* package uses X.509 as a transport: RT/ABAC credentials are X.509 attribute certificates with an RT logic statement encoded in the attribute field. The toolkit handles credential encoding and decoding and generates and checks signatures and expiration times on the certificates. The RT/ABAC inference procedure handles recursive processing of delegation chains.

3.2 Slices and trust

The slice abstraction simplifies authorization and accountability in networked clouds. A *slice* is a group of virtual resource instances (e.g., VMs). In networked clouds, the slice is a convenient abstraction to name, control, and contain groups of virtual resources that span multiple provider sites and are allocated and used for a common purpose. The virtual resource instances are called *slivers* in GENI. GENI introduces the new generic term *sliver* to encompass a diversity of virtual resource types. Every sliver is a member of exactly one

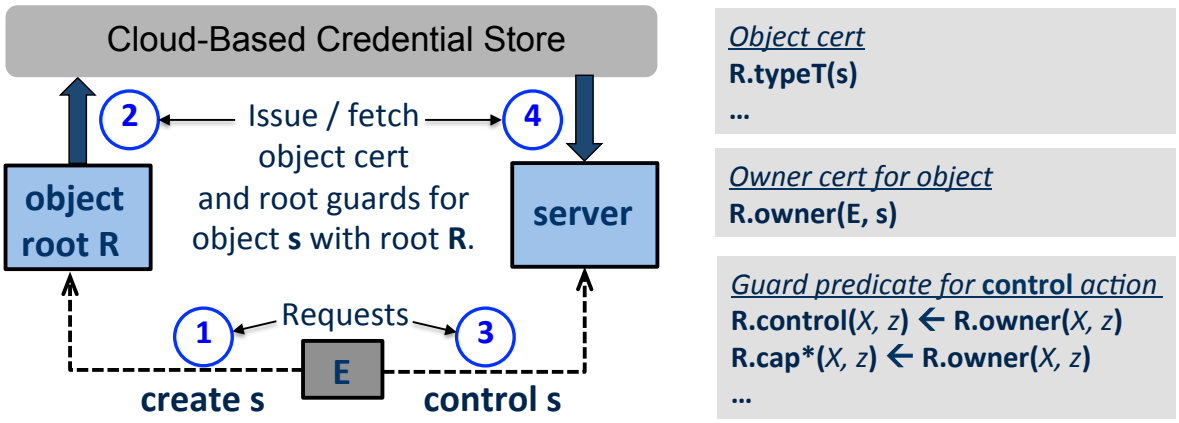

Figure 3: Creation and use of a global object. A client entity E requests an object root service R to create an object s (such as a slice), and then passes s by reference to a server (such as a cloud site) to operate on it. The root issues statements about the new object ($R.s$) in the trust logic. These statements include attributes of s, root policies (*guards*) governing access to operations on s (such as **control**), and roles granting rights and powers over s to its owner (E). Authorization policies in the servers use these statements to control access to the object. In this example the statements are shared through a global credential store (see Section 4.1).

slice. The slice concept is inherited from PlanetLab [22] and its Slice Federation Architecture (SFA).

In our approach, slices are the granularity of authorization and accountability for slivers: e.g., if a provider grants an entity privilege to control a slice, then that entity can control any local sliver in the slice. This decision simplifies the system at the cost of ruling out authorization policies that distinguish among slivers in the same slice.

A slice—or rather the software running within it—can also be a subject entity in the trust architecture. A cloud site may endorse a keypair as bound to a particular slice, e.g., in response from a Certificate Signing Request (CSR) that the cloud provider can verify originates from an instance within the slice [7]. The slice and its owners and project leader are accountable for actions taken with a slice key. Other entities may choose to trust the slice key in part on that basis. One goal of our cloud trust architecture is to enable entities to manage trust in slice keys based on statements made by various entities about the key, slice, program, project, and/or slice owner(s). We view slice keys as generalizing GSI proxy keypairs—which enable a grid job to speak for a user for a limited time—to the case of long-running networked execution contexts controlled by multiple entities and spanning multiple providers. However, GENI does not yet support slice keys, and they are outside the scope of this paper. (See [7] for further treatment.)

Each slice is bound to a project (Section 2.3). Each project has a *leader*—a user who is ultimately accountable for the behavior of slices in the project.

3.3 Global objects

One contribution of the work is to show how to use RT trust logic in a practical distributed system with global objects, such as slices and projects. A *global object* has a global name which a requester may use to name the object by reference in a request to a server. Many servers may support operations that involve any given object. The authorization policy for an operation on a global object must reason from statements about the object made by other entities.

For example, various GENI-API operations on provider sites operate on slices, e.g., by binding slivers to a slice or controlling slivers within a slice. In particular, a request to a provider to create a sliver names the slice that will contain the sliver; in principle, any provider may create a sliver for any given slice. Each

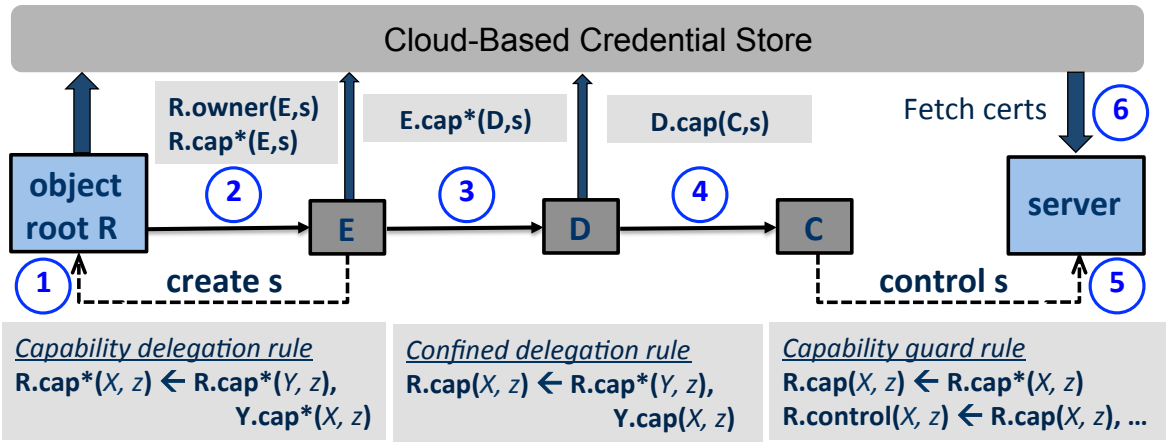

Figure 4: An example of capability delegation and confinement governed by declarative policy rules. This figure extends Figure 3 with a transitive delegation of rights to operate on the object from the owner (E) to the requester C through an intermediate entity D. The right is represented here as a generic capability by the **cap** role, which includes both delegatable (**cap***) and non-delegatable (**cap**) forms. The server infers from the declarative policy rules that the delegated capability is valid.

provider applies a local policy to authorize these actions on slices. At minimum, the provider must verify that the slice is bound securely to an approved project, and that the requester has privilege to operate on the slice. The user, project, and slice also have attributes that a resource allocation policy may consider.

Figure 3 illustrates creation and use of a global object such as a slice. A global object name is a UUID/GUID issued by a particular entity: the entity that asserts the existence of a global object and publishes its name is called the object's *root entity*. The root entity for an object issues a signed *object certificate* asserting the object's name and attributes. A fully qualified object name includes the object's root (see Figure 2). These object names are globally unique self-certifying identifiers: they are *authenticated* in the sense that any entity can verify that the object certificate—or any other statement about the object in the trust logic—originates with its root or is accepted by the root according to its policies (i.e., the root *says* the statement is true).

The root entity may also define and issue roles/attributes that confer a privilege to operate on the newly created object in some way. A role may represent a right to operate on an object: at most one parameter in each role predicate may be a string representing a globally unique name for a global object. For example, such a role may represent privileges to control a slice or membership privileges for a project. The root may also issue policy rules in the trust logic that define the root's constraints for to approve operations on the object. We refer to policy rules issued by the root as *root guards*. Servers that operate on the object may obtain the policy from the object's root and apply those rules locally, possibly combining them with local policies or other policy rules from other entities.

3.4 Object capabilities

Access policies (guards) for a global object may allow the holder of rights over the object to delegate those rights. The trust logic provides a powerful formalism to express and check these delegations and the policy rules that govern them. A delegation chain that confers privilege to operate on a global object is rooted in the object's root entity. A server accepts the chain of delegations only if it conforms to the policy rules in the root guards, i.e., the server can infer that the root *says* access is permitted, based on the policy rules and statements about the object and requester by other entities. Any server may impose additional constraints or access checks in its own policy rules.

For example, the SFA slice model uses capabilities to authorize actions on slices. It is easy to represent a

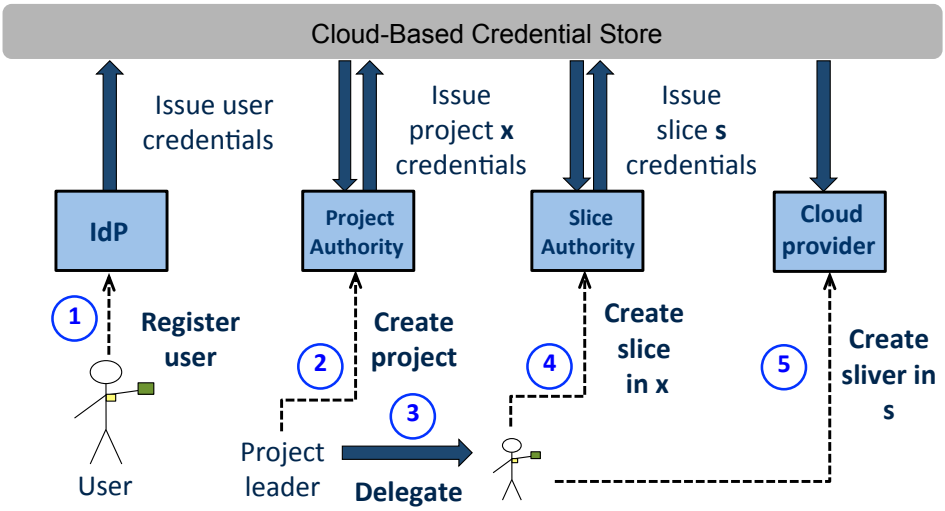

Figure 5: Coordinator entities and credential flow for a typical resource request in GENI. Each virtual resource instance ("sliver") is obtained from a single resource provider ("aggregate"), and is bound to a project and slice that have been approved by authorities trusted by that aggregate according to its policy. The authorities also publish usage constraints for the projects and slices they create. Access is fast because projects and slices are coarse-grained objects, and credentials for them are cached at the aggregates.

capability-based access control declaratively as one form of rights delegation policy enabled by RT/ABAC. A *capability* is simply a trust fact with a role that is understood to confer a specific right or privilege. Simple RT/ABAC policy rules allow users to delegate and confine capability roles [11]. Figure 4 illustrates a simple declarative policy for capability delegation with confinement. In practice such policies are likely to be augmented with additional access checks. For example, a policy might state that rights to control a slice may be delegated only to valid members of the project that the slice is associated with.

3.5 Coordinators

GENI defines various coordinator services to mediate trust among resource providers and users. The coordinators are entities that consume credentials, evaluate them according to a policy, and produce other credentials. The GENI coordinators include common services to approve users, slices, and projects, and bind attributes to them.

The resource providers or other entities may consider these attributes in their local policies to authorize actions and/or allocate resources. As with "sliver", GENI introduces the new generic term *aggregate* for the resource providers because the architecture encompasses a diverse range of infrastructures and platform services and other resource providers. A GENI aggregate may be a virtual machine service such as the standard OpenStack cloud sites used in the ExoGENI testbed [3]. Thus an example of a sliver is a VM allocated from an EC2-like cloud site, but that is only one example: the GENI architecture accommodates other kinds of slivers and aggregates.

Aggregates have local policies that determine which coordinators they trust and accept to approve users, projects, and slices and to issue various credentials. In the initial GENI deployment sponsored by the US National Science Foundation all coordinators use standard GENI policies and are approved by the sponsor or its delegate. Note, however, that the policies to approve and accept coordinators are also specified declaratively in the logic, as discussed below.

Figure 5 depicts the entities involved in approving a sliver allocation. The figure depicts three types of coordinators.

- **Identity Provider** (IdP) is any entity that is trusted to assert attributes of a real-world identity without proof. Every GENI user must possess a keypair that is endorsed by a GENI-approved IdP. In our prototype and the initial GENI deployment, the IdPs are simple web portal "identity brokers" linked to Shibboleth identity services, as described in Section 2.3. Also, some established GENI-affiliated testbeds such as Emulab [23] run GENI-approved IdPs that manage their own user accounts.
- **Project Authority** (PA) approves the creation of projects. A decision to approve a project is based at least in part on validated attributes of the requester. A PA acts as the root entity for projects that it approves: it issues a credential declaring the project and binding it to a name and other attributes. The PA also issues a credential designating the requester as the leader of the new project.
- **Slice Authority** (SA) approves the creation of slices. A decision to approve a slice is based at least in part on validated attributes of the requesting user and the user's association with a project. An SA acts as the root entity for slices that it approves: it issues a credential declaring the slice and binding it to a project, a name, and other attributes. The SA also issues a credential designating the requester as the *owner* of the slice, conferring a privilege to control the slice.

A deployment may have many coordinators of each type, authorized for their roles by transitive delegation from a federation root or other trust anchors accepted by the aggregates, as described below.

A project leader (or its delegate) may delegate rights to create slices in a project and otherwise participate in activities associated with the project. The owner of a slice may delegate privilege to control a slice or to obtain information about a slice. These delegations may be transitive at the discretion of the delegator.

With RT/ABAC, the policies for accepting these delegations may be constrained or extended by slice-specific policies of the SA that approved a slice, project-specific policies of the PA that approved a project, project-specific policies of the project leader or its delegate, and/or other policies of the GENI federation.

GENI-affiliated aggregates provide resources to GENI slices only in accordance with GENI-approved policies, based on attributes of users, projects, and slices asserted by GENI coordinators. A GENI aggregate may approve a sliver request from an entity that presents a credential set demonstrating privilege to control a valid GENI slice on behalf of a valid GENI user, in accordance with GENI-approved policies.

3.6 Trust structure

RT/ABAC facts and rules can also represent the *trust structures* used for federation. For example, the decision by a cloud provider or GENI aggregate to accept credentials from a coordinator entity may also be represented as a role delegation. An aggregate's policy may delegate the approval choice to some other entity. The trust logic can represent a wide range of trust structures. For example, the entity approving a coordinator may itself be named by attribute it possesses—an attribute-based delegation.

To illustrate, Figure 6 depicts the trust structure of the initial NSF GENI deployment and its declarative representation in the trust logic. The set of approved aggregates and coordinators are endorsed by a single anchor entity called the *federation root*. The federation root endorses these entities based on operational policies for the deployment and agreements with the operators of those services, which are negotiated out of band. The federation root is globally known: all participating entities have a local fact to accept the federation root. Standard local policies accept all entities endorsed by the federation root. The local facts (e.g., accepted trust anchors) and policies are installed at each entity by a local operator.

Importantly, the trust structure in this example is declarative and fluid. The trust structure can be changed without modifying the software. Depending on the policies installed by the entities, changes made by one entity may propagate immediately to the others. For example, in Figure 6 any new identity provider endorsed by the federation root is immediately accepted by the other entities. Any aggregate could serve users of another parallel federation simply by installing a single fact to accept its federation root. Or, as another example, two federations could agree to peer and serve each other's users by installing policy rules at the

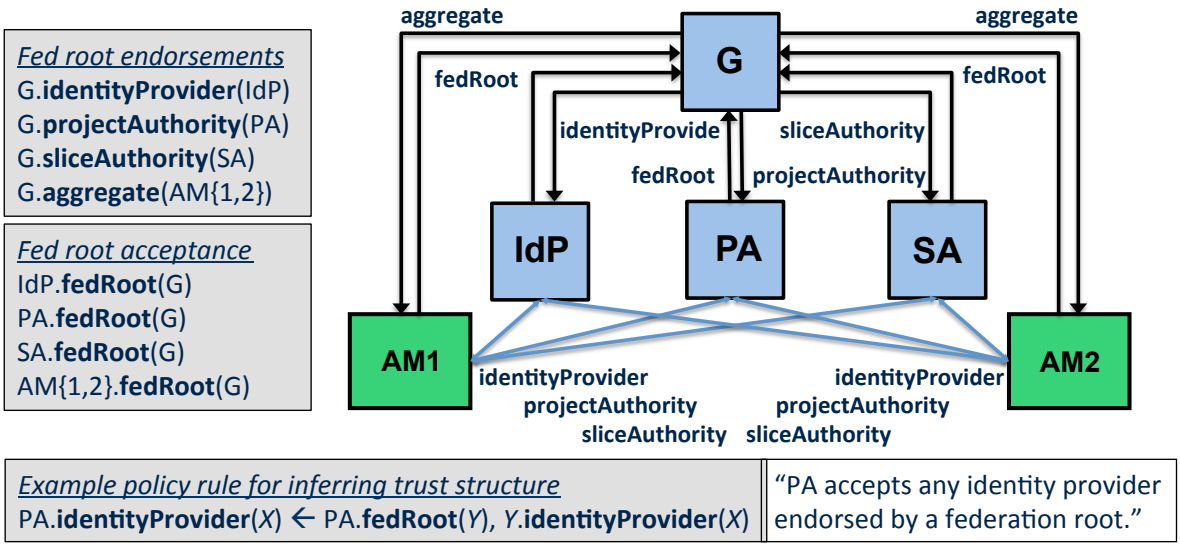

Figure 6: An illustration of declarative trust structure for the initial NSF GENI deployment. Entities labeled *AM* (Aggregate Manager) represent federation-approved cloud sites or aggregates. All participating aggregates and coordinators are endorsed by a common trust anchor called the federation root (G). Entities join the federation when the accept the root and are endorsed by the root. Entities install standard local rules to accept other entities endorsed by the root. The trust structure emerges by inference from declarative statements of the participating entities.

root to accept one another's coordinators under certain conditions. Thus the global trust structure emerges from local declarative policy choices of the participating entities.

4 Implementation

We have built a prototype that implements simple versions of each RT/ABAC credentials and flows among these entities. The initial GENI deployment has instances of each of these services, but our prototype is currently the only implementation that produces and consumes RT/ABAC certs. The prototype includes a web-based IdP, Project Authority, and Slice Authority implemented as PHP scripts that invoke Java programs linked to *libabac*. The prototype also includes an ORCA extension for RT/ABAC authorization policy in ORCA-based GENI aggregates.

All entities run with their own keypairs and exchange credentials through a shared web-based credential store, as illustrated in Figure 5 and described below. All public keys are endorsed with suitable roles by RT/ABAC credentials issued by a common trust anchor. All entities are seeded with these credentials so that they accept credentials issued by one another in their assigned roles.

The prototype IdP is a Shibboleth-linked identity broker, as described above. These IdPs issue credentials for users at least partly on the basis of statements made by Shibboleth/InCommon identity services of participating institutions. A user logs into the PHP web site using the home institution's Shibboleth identity server, and uploads a public key through a web form. The IdP harvests attributes from the Shibboleth session and reissues them as roles in RT/ABAC credentials signed with the IdP's private key. The IdP also issues a standard X.509 identity certificate to install in the user's browser.

To create a project, a user visits a separate web service entity, authenticating using its X.509 identity certificate. The prototype PA ingests RT/ABAC credentials for the user's public key, checks that the public key in the credential matches the keypair used by the browser, and runs an RT/ABAC policy on the credential

Guard policy to create a project PA.**createProject**(*X*) ← PA.**identityProvider**(*Y*), *Y*.**geniUser**(*X*), *Y*.**faculty**(*X*)	"The Project Authority PA approves creation of a project if an identity provider *Y* accepted by PA says that the requester *X* is a **faculty** member who has registered as a **GENI user**."
Guard policy to create a slice SA.**createSlice**(*X*) ← SA.**projectAuthority**(*Y*), *Y*.**project**(*p*), SA.**root**(*Y, p*), *Y*.**createSlice**(*X,p*)	"The Slice Authority SA approves creation of a slice if a Project Authority *Y* accepted by SA says that the requester *X* has **createSlice** rights in a valid project *p*, where *Y* is the **root** of *p*."
Guard policy to create a slice in a project PA.**createSlice**(*X, p*) ← PA.**owner**(*X,p*)	"The Project Authority PA approves creation of a slice in a project *p* if the requester *X* is the leader/**owner** of *p*."

Figure 7: Summary of declarative policies in the initial prototype, for the coordinator entities in Figure 5.

set. The simple policy approves project requests by any faculty member. It issues RT/ABAC credentials for the project signed with the PA's private key. The prototype SA is similar.

The prototype also supports policy mobility. The PA issues policy rules for each approved project, and the SA issues policy rules for each approved slice. These rules are issued as signed RT/ABAC credentials, and are imported and interpreted by downstream entities. For example, the SA evaluates the PA's policy to determine if a requester is authorized to create a slice in the project.

Figure 7 summarizes the policies in the prototype, which are simple and standard. The PA's policy for a project approves creation of slices bound to the project by any approved GENI user with membership privilege in the project, as delegated transitively by the project leader. The SA evaluates the PA's policy after verifying that the policy rules are issued by the root PA entity for the named project, and that its local policy accepts that PA as a project authority, i.e., it is endorsed by the root trust anchor. Similarly, the SA-generated policy for a slice allows sliver requests from any approved GENI user with control privilege over the slice, as delegated transitively by the slice owner. Note that it is possible within the framework to generate different policies on a per-project or per-slice basis, e.g., some projects or slices might require a security clearance.

The aggregate evaluates these policies simply by verifying their source in the correct root entity, verifying that it accepts the root entity in the correct role, and passing all relevant credentials for facts and policy rules to the RT/ABAC inference engine, implemented in the *libabac* library. Aggregate policies may include other checks at their own discretion. Once a request is approved, an aggregate may associate slivers with slices and projects based on these RT/ABAC credentials. This association is important for accountability purposes. The aggregate can prove the legitimacy of the association by displaying the signed credentials in a proof returned by the inference engine.

We are extending our prototype to support other aspects of the trust model for the NSF GENI deployment. For example, GENI policy states that each GENI aggregate must authorize GENI operators (delegates of the GENI federation root) to control any GENI slice as needed to provide for the general safety and welfare. In particular the operators have a *kill switch* for rogue slices. The kill switch is easy to implement as a statement in an RT/ABAC policy [11]. The aggregate simply grants kill privilege over GENI slices to a named delegate of the GENI federation root, independent of whether the delegate possesses a capability from the slice owner.

An early prototype of RT/ABAC authorization has also been implemented for Emulab/ProtoGENI aggregates, with assistance from the ProtoGENI development team. This prototype accepts a credential set as an argument in a network request, and does not support policy mobility.

4.1 Credential storage and retrieval

A key challenge of the approach based on trust delegation logic, as described in this paper, is that any trust decision may involve a larger number of credentials (signed assertions or policy statements) drawn from various sources. The inference engine at the policy decision point must have access to these credentials. In RT/ABAC, the credentials are X.509 atribute certificates that contain logic statements (RT0, RT1, RT2) encoded within them. These certificates have expiration dates, and must be renewed as they expire. If a key is lost, there must be some means to invalidate the certificate. A given certificate may contain a statement referencing multiple keys.

Our design uses a *cloud-based credential store* supporting query and retrieval of credential sets, as illustrated in Figure 5 and other figures in Section 3. A cloud-based store enables any entity to fetch groups of credentials relevant to an authorization decision, with suitable privilege to access those credentials. It can also enable a credential issuer to proactively renew credentials before they expire, or to "poison" credentials that are no longer valid due to compromise of the endorsed key. Our current prototype is based on a simple Web-based credential store called *POD* [14]. In our ongoing work we are exploring decentralized credential storage based on a replicated key-value store that runs as a distributed cloud application.

Perhaps most importantly for a practical implementation of our framework, the credential store model and POD support caching of credentials. In the credential store model the authorizers "pull" credentials from the store, rather than receiving credentials "pushed" by the requester in every request. The authorizer fetches credentials only as needed to resolve cache misses or to refresh stale or expired credentials.

The store-based approach requires some means to name sets of credentials for querying the cache or store. This problem has been called the *credential discovery problem*. POD organizes and names credential sets by a subject and object and limited linking relationships. This topic is outside the scope of this paper.

5 Related work

5.1 Generalizing PKI with logic-based trust management

Our solution associates principals with attributes and uses an inference procedure for authorization decisions based on attributes and policy rules. Principals issue credentials as authenticated statements of belief, e.g., that a given subject possesses a given attribute. SPKI/SDSI [12] popularized these ideas in the 1990s. Our approach is similar but uses the logic-based language of RT/ABAC [19]. RT/ABAC generalizes SPKI/SDSI, e.g., by adding support for conjunctions in policy rules [18].

SPKI/SDSI and RT/ABAC (as implemented in *libabac*) may be viewed as generalizations of classical PKI. These systems name and authenticate principals by their public keys. Principals issue credentials as signed certificates; for example, the *libabac* encodes RT statements into X.509 attribute certificates. Both systems support a form of *role credential*, a statement of belief that an atomic predicate is true for a given subject, or, equivalently, that the subject holds a role or is a member of a set named by a role, which may be qualified with other parameters. A role credential endorses the subject's public key and delegates the named role to the subject. Classical PKI systems support a similar delegation mechanism in identity certificates, which bind a key to a distinguished global name of a real-world identity. SPKI/SDSI and RT/ABAC extend the PKI delegation mechanism to bind keys to attributes drawn from local name spaces. The extended mechanism supports limited delegations of trust, and also provides a rich basis for trust even in the absence of global distinguished names [12].

These trust management systems share a common heritage with other logic-based formalisms for authorization and trust, including Delegation Logic [17]. Trust delegation logics might be viewed as a generalization of belief logic (e.g., BAN logic), in that entities must reason about their own acceptance or belief in statements made by other entities. In particular, trust logics incorporate the *says* operator from BAN logic and its successors [15]. For example, in RT/ABAC every fact or rule is tagged with the entity that *says* it, i.e., the entity that stated the fact or rule in a valid signed credential, or is known to believe the fact as a consequence of its stated rules (if a fact is inferred).

Grid-inspired research has yielded several PKI-based trust systems that are similar in that they combine roles and delegations with some form of declarative policy [16]. Examples include the PERMIS [10, 9] system used in European grid initiatives. These systems generally follow the approach pioneered by SPKI/SDSI, but introduce custom policy languages. We believe that logic-based policy languages such as RT/ABAC are a better approach because they are expressive, tractable, and have strong theoretical foundations. For example, the PERMIS policy language is based on Keynote, an early trust management system [6]. Keynote researchers later went on to pioneer Delegation Logic [17], a predecessor of RT/ABAC. In our view the modern logic-based approach is a better foundation for growth of networked cloud ecosystems and the flexible trust management they require.

5.2 Attribute-based identity services

Our approach incorporates external identity services for user account management, such as the Shibboleth [21] web single sign-on (SSO) system deployed by the many academic institutions participating in the InCommon federation. The "identity providers" (IdPs) in the trust structure described in Section 3.5 are servers trusted by the cloud federation root that bridge the PKI-based trust fabric to external identity services. Amazon's Identity and Access Management (IAM, *http://aws.amazon.com/iam/*) has recently adopted a similar approach under the term *identity broker*. This approach requires no changes to the SSO system: it views the identity broker as no different from any other Service Provider (SP) using the web sign-on service.

Shibboleth and other SAML-based identity services represent the identity of a user as a set of attributes or roles. A distinguished global name is one kind of attribute, but often it is sufficient for an SP to know the user's role (e.g., student). In fact, the privacy policy in the identity service may choose not to reveal the user's name to an SP. Thus SAML follows the approach of SPKI/SDSI and other systems that base access control decisions on roles or attributes. We leverage SSO identity services as attribute sources for an attribute-based access control system (RT/ABAC) in which principals authenticate by public keys.

Identity attributes may represent a user's membership in a group. The Shibboleth ecosystem includes web-based services to manage group memberships. These services continue to evolve. Amazon's IAM also enables groups within an identity domain (account) and has recently introduced some more general support for roles. Our approach is consistent with IAM, but it also adds support for projects and groups whose members span identity domains.

OpenStack's Keystone module offers an open interface for an external identity and trust framework based on roles and a mapping of roles to privileges. OpenStack is the basis for many cloud deployments on academic campuses and elsewhere, including cloud sites in the ExoGENI testbed [3]. We believe that our framework is compatible with Keystone, but it has not been necessary to integrate it into Keystone because ExoGENI handles all identity and authorization in ORCA servers that interpose on calls to OpenStack through the GENI API.

5.3 Grid computing systems

Grids emphasize federated environments similar to what we propose: they combine multiple resource providers in a unified service for a community of users that may span multiple identity domains. The evolution of grid security architecture reflects many of the same choices in our approach for community clouds and federated clouds. For example, grids use PKI for similar reasons: PKI is convenient to authenticate messages from

hands-free user tools and from programs running inside the grid. We summarize other similarities below. What sets our approach apart is the use of general-purpose trust logic as a unifying formalism that can specify a wide range of trust mechanisms and policies declaratively, minimizing the need for custom software to implement each design choice. These include the choices implemented in today's grid systems and many other choices that we believe are important for future cloud services and other general infrastructure services such as GENI. We first summarize related choices in grid systems and then return to the comparison.

External identity and attributes. Early grid systems used simple mappings of external user identities (distinguished names) to local user identities (user IDs) with specific rights and powers (gridmap). Over time it was recognized that user privileges flow from memberships and roles in communities, and relationships of sharing and trust among these communities. This motivated a more dynamic and fluid trust decisions, based on identity attributes and third-party endorsements of identities that are not known to the provider. For example, grid designers also initiated early efforts to bridge web sign-on (SSO) systems to PKI-based grid security infrastructure [4]. Many deployed grids today bridge web SSOs to their PKI systems using variants of the simple identity broker concept outlined above. Examples include recent versions of MyProxy [5], the Short-Lived Credential Service portal (SLCS), and several others.

Groups: Virtual Organizations. The concept of a Virtual Organization (VO) serves as the unit of grouping for users and providers in many deployed grid systems [13]. Most importantly, VOs are groupings of users spanning multiple identity domains (although they may also involve groupings of grid resource providers). Many grid systems employ a service called Virtual Organization Management Service (VOMS [1]) to manage user membership in VOs. Each VOMS server is recognized by other entities as authoritative for one or more VOs or for a specific set of groups. The VOMS issues credentials containing statements about user membership and roles in VOs. Resource providers that trust the VOMS to make such statements may consider them in deciding whether to grant access.

Thus VOMS is an example of the general coordinator concept presented in Section 3.5. A VO corresponds loosely to a *project* in our architecture, and VOMS corresponds loosely to a Project Authority. There may be differences in granularity, but both approaches support multiple group coordinators and nested subgroups with delegated administration.

Certificates and delegations. Grid systems make frequent use of signed certificates and delegation. For example, a VOMS instance issues credentials as X.509 attribute certificates signed under its own keypair and binding a user's public key to one or more roles scoped to a named VO. RT/ABAC and the *libabac* toolkit use X.509 attribute certificates in a similar way, but also encode other kinds of statements in the certificates, including policy rules.

Grid standards also allow user delegations that endorse some entity to speak on the issuer's behalf: e.g., the RFC 3820 GSI/X.509 proxy certificate standard allows a user to endorse a *proxy keypair* for use by some entity under its control, such as a job running in the grid. These kinds of delegations are easily represented using a trust delegation logic such as RT/ABAC: in essence, the user endorses the proxy keypair as possessing a **subidentity** role, whose meaning is that the subidentity inherits privileges of the user and that the user is accountable for its actions. RT/ABAC allows the user to delegate other roles to the keypair, including user-defined roles. Thus it subsumes the GSI proxy certificate mechanism and also allows finer-grained control over the trust delegated to the subidentity.

Summary: grid systems vs. advanced clouds. To be sure, many of the challenges for trust management, user identity management, and authorization in cloud systems are already addressed in grid implementations and deployments, as summarized above. A key premise of our approach is that general trust logic is a solid foundation for additional trust management challenges that future cloud systems will face. Advanced cloud systems support a wide range of infrastructures, platforms, and uses that extend beyond the job execution systems that is the focus of grid implementations and deployments. (With that said, we recognize that the early grid computing vision and various leaders in the grid community have conceived of grids more broadly to encompass some of the platform services that we now call "cloud".)

The slice abstraction discussed in this paper loosely captures this distinction. Slices may host long-running services controlled by multiple users at different times, motivating the support for capability-based access

control discussed in Section 3.4. Our approach enables additional attributes on slices themselves, e.g., asserted by a new kind of authority service (Section 3.5). Infrastructure providers themselves may attest or assert properties of slices they host (e.g., [7]). A rich trust framework is an important foundation for constructing network service ecosystems by composing and interconnecting slices.

5.4 Trust structure for federated systems

Our declarative approach to trust structure is broadly applicable to other federated (multi-domain) distributed systems. Many federated systems in use today are deployed with similar trust structures. Often the trust structure is "baked in" to the software, or is partly declarative in ad hoc configuration files. We cite three common examples already discussed in this paper:

- Establishing trust in certifying authorities (CAs) in classical PKI hierarchies.
- Federated identity systems require a means to establish mutual trust among identity providers (IdPs) and service providers (SPs). The InCommon identity federation described in Section 2.4 is an example of a trust structure extended with another level of delegation: the SP delegates trust to the InCommon root, and the SP adds (in essence) a policy rule to accept credentials from any IdP endorsed by the root, even if the IdP is not known in advance.
- Grid systems require some means to broker trust among resource providers and VOMS (or CAS) instances that control virtual organizations and user privileges. International Grid Trust Federation (IGTF) is an international trust fabric of endorsing parties has grown up around grid deployments.

RT/ABAC logic enables us to represent such structures and other trust structures for a deployment declaratively, and change them without modifying the software.

6 Conclusion

This paper gives an overview of a trust framework suitable for federated community clouds serving user communities spanning multiple identity domains. The architecture has been adopted as the basis for managing identity and trust in the NSF GENI project. The work is in prototype stage; work to incorporate it into public GENI deployment is ongoing.

Our approach meets the goals of many previous systems: integration with external identity services such as web sign-on systems, flexible delegation and assignment of roles and attributes, groups spanning multiple identity domains, and declarative policy that combines policies and assertions from multiple sources.

A key element of the work is its grounding in an off-the-shelf logic-based authorization framework with solid theoretical underpinnings (RT/ABAC) and a portable implementation (*libabac* from ISI). The logic framework generalizes PKI-based delegation mechanisms in many previous systems. It also supports flexible, declarative trust structures for evolving federations, such as the NSF GENI deployment. It also provides a powerful foundation for managing access and accountability for long-lived networked execution contexts (slices), and reasoning about trust in hosted software entities.

Note. More details on our use of ABAC and our prototype can be found in references [14, 11], and on *http://groups.geni.net/geni/wiki/AuthStoryBoard*.

7 Acknowledgements

Andrew Brown and Muzhi Zhou contributed to this work. We thank RENCI, NSF, IBM, and the GENI Project Office (GPO) at BBN for their support. Many of the ideas in this paper have evolved in GENI

discussions with many participants, notably Tom Mitchell and Marshall Brinn of GPO, and others on the GENI Architects panel. Thanks also to Steven Carmody and Ken Klingenstein for discussions involving Shibboleth and federated identity. Thanks also to Ilia Baldine and the ORCA/ExoGENI team for their patience and assistance with the prototype.

References

[1] R. Alfieri, R. Cecchini, V. Ciaschini, L. dellAgnello, . Frohner, A. Gianoli, K. Lrentey, and F. Spataro. VOMS, an Authorization System for Virtual Organizations. In F. Fernndez Rivera, M. Bubak, A. Gmez Tato, and R. Doallo, editors, *Grid Computing*, volume 2970 of *Lecture Notes in Computer Science*, pages 33–40. Springer Berlin / Heidelberg, 2004.

[2] I. Baldine, Y. Xin, A. Mandal, C. Heermann, J. Chase, V. Marupadi, A. Yumerefendi, and D. Irwin. Autonomic Cloud Network Orchestration: A GENI Perspective. In *2nd International Workshop on Management of Emerging Networks and Services (IEEE MENS '10), in conjunction with GLOBECOM'10*, Dec. 2010.

[3] I. Baldine, Y. Xin, A. Mandal, P. Ruth, A. Yumerefendi, and J. Chase. ExoGENI: A Multi-Domain Infrastructure-as-a-Service Testbed. In *TridentCom: International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, June 2012.

[4] T. Barton, J. Basney, T. Freeman, T. Scavo, F. Siebenlist, V. Welch, R. Ananthakrishnan, B. Baker, M. Goode, and K. Keahey. Identity Federation and Attribute-based Authorization through the Globus Toolkit, Shibboleth, Gridshib, and MyProxy. In *5th Annual PKI R&D Workshop*, April 2006.

[5] J. Basney, M. Humphrey, and V. Welch. The MyProxy online credential repository. *Software Practice and Experience*, 35(9):801–816, July 2005.

[6] M. Blaze, J. Feigenbaum, and A. D. Keromytis. Keynote: Trust management for public-key infrastructures. In *Infrastructures (Position Paper). Lecture Notes in Computer Science 1550*, pages 59–63, 1998.

[7] A. Brown and J. S. Chase. Trusted Platform-as-a-Service: A Foundation for Trustworthy Cloud-Hosted Applications. In *Proceedings of the ACM Cloud Computing Security Workshop (CCSW '11)*, October 2011.

[8] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, 1:146–166, 1989.

[9] D. Chadwick, G. Zhao, S. Otenko, R. Laborde, L. Su, and T. A. Nguyen. PERMIS: a modular authorization infrastructure. *Concurrency and Computation: Practice and Experience*, 20(11):1341–1357, August 2008.

[10] D. W. Chadwick and A. Otenko. The PERMIS X.509 role based privilege management infrastructure. In *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 135–140, June 2002.

[11] J. Chase. Authorization and Trust Structure in GENI: A Perspective on the Role of ABAC. http://www.cs.duke.edu/~chase/geni-abac.pdf, June 2011.

[12] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI Certificate Theory. RFC 2693 (Experimental), September 1999.

[13] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222, 2001.

[14] P. Jaipuria. Building GENI Authorization with a Distributed Trust Management Framework. MS Project Report, Duke University, May 2012.

[15] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: theory and practice. *ACM Trans. Comput. Syst.*, 10(4):265–310, Nov. 1992.

[16] B. Lang, I. Foster, F. Siebenlist, R. Ananthakrishnan, and T. Freeman. A flexible attribute based access control method for grid computing. *Journal of Grid Computing*, 7:169–180, November 2008. 10.1007/s10723-008-9112-1.

[17] N. Li, B. N. Grosof, and J. Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. *ACM Transactions on Information and System Security*, 6:2003, 2000.

[18] N. Li and C. Mitchell. Understanding SPKI/SDSI using first-order logic. *International Journal of Information Security*, 5(1):48–64, January 2006.

[19] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust-management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–, Washington, DC, USA, 2002. IEEE Computer Society.

[20] P. Mell and T. Grance. The NIST Definition of Cloud Computing. Special Publication 800-145, Recommendations of the National Institute of Standards and Technology, September 2011.

[21] R. L. Morgan, S. Cantor, S. Carmody, W. Hoehn, and K. Klingenstein. Federated Security: The Shibboleth Approach. *EDUCAUSE Quarterly*, 27:12–17, 2004.

[22] L. Peterson, A. Bavier, M. E. Fiuczynski, and S. Muir. Experiences Building PlanetLab. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)*, November 2006.

[23] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 255–270, December 2002.