A Unified Approach to Trust, Delegation, and Authorization in Large-Scale Grids

Blair Dillaway

September 2006

Technical Paper

Microsoft Corporation

One Microsoft Way

Redmond, WA 98052

Copyright

Contents

1 Introduction

This paper presents results from an on-going project investigating access control solutions for large-scale Grid Computing Environments. A primary focus of this work is to develop flexible and robust mechanisms for expressing trust relationships and constrained delegation of rights in a uniform authentication and authorization framework. Our approach offers potential improvements over existing grid security solutions (for example, [Mul05], [Wel03], [Shib05], [VOM03], [Uni]) in supporting finer-grained control and easier adaptation to different operational models, while at the same time being easier to understand, analyze, and manage.

The information in this paper reflects results from both security research and practical experience gained in developing an end-to-end prototype system. The prototype emulates a multi-domain grid environment, and uses existing Microsoft products including the Microsoft Windows® Compute Cluster Server 2003, .NET Framework, Windows Communication Foundation (formerly code-named *Indigo*), Active Directory® directory service, and Kerberos and X.509-based identity management infrastructures. It also incorporates several industry standards, such as XML and Web services protocols, for interoperability.

Section 2 describes the grid environment example used in the investigation. Section 3 describes a scenario that is the focus of this paper. Section 4 examines the core technology that forms the basis of the Microsoft approach. The core technology is a new declarative language for expressing security policies and other security-critical information: the Security Policy Assertion Language (SecPAL). Section 5 of the paper then explains how to apply SecPAL to help secure the scenario described in Section 3.

2 The Grid Environment

Large-scale grid environments are complex and involve many users, data and computational resources, network channels, and administrative domains. This complexity makes it difficult to describe all of the entities and relationships required to provide access control within such systems. This paper uses the fictitious grid environment depicted in Figure 1 to help explain the access control solution. This solution is sufficiently detailed to describe the key challenges that developers would experience when they develop such solutions, without being unnecessarily complex.

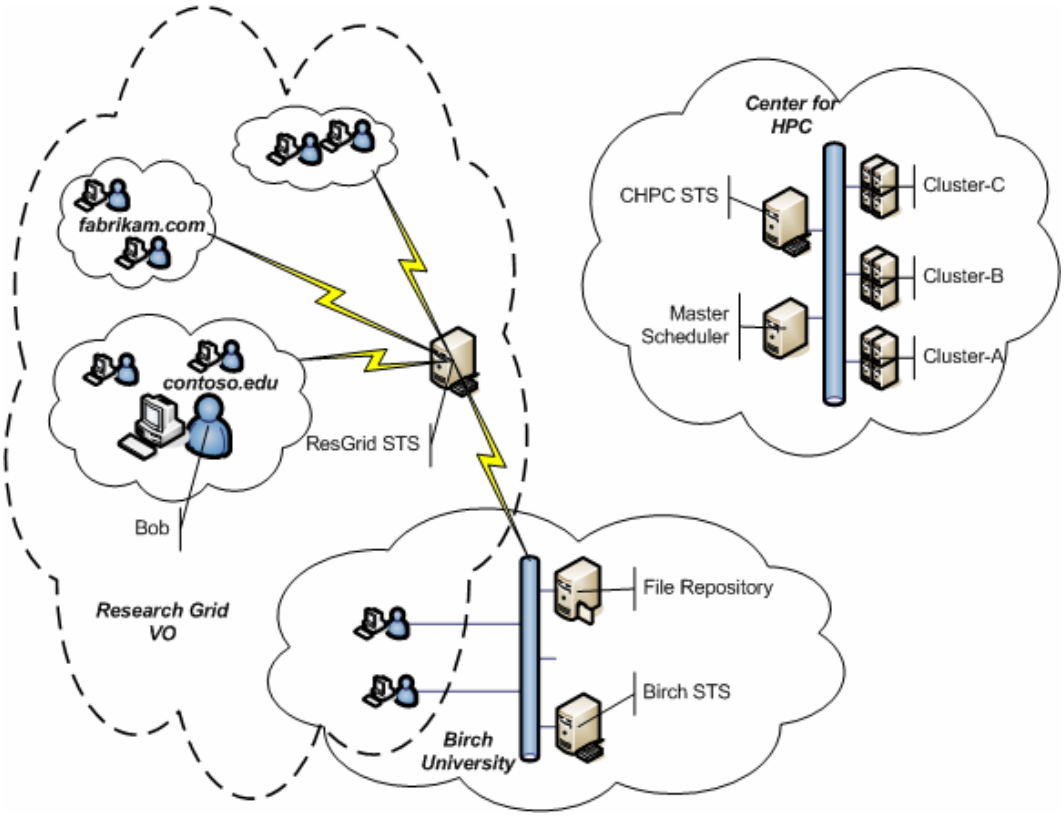

Figure 1. Large-scale Grid Computing Environment example

The grid environment consists of multiple organizations, some of which are joined into a virtual organization (VO) which handles certification of the grid users. For simplicity, the example shows compute and data resources isolated in separate organizations, although the Microsoft approach is not limited to such environments. The entities in the grid use standard protocols over the Internet to communicate. The prototype implementation uses SOAP-based Web services protocols (such as SOAP Message Security, WS-Trust, and so on).

The organizations and their associated grid entities are:

- **Center for High Performance Computing (CHPC)** – This organization (c-hpc.org) provides computational resources and job management functionality for the grid. It supports an Internet accessible master job scheduling/management service. This service is responsible for scheduling user jobs on the CHPC compute clusters, which have their own local job scheduler/manager. The scenario assumes that the clusters are not directly accessible from outside the CHPC domain. CHPC runs a domain Security Token Server (STS), which issues security tokens to CHPC domain entities.
- **Birch University** – Birch (birch.edu) runs an Internet-accessible file repository. It allows grid users to store and manage data files that are isolated based on the user and project affiliation. Birch runs a domain STS that issues security tokens to Birch-

hosted grid services (the file repository and possibly others). The Birch grid users obtain their grid security tokens from the ResGrid VO.

- **Research Grid (ResGrid) VO** – ResGrid (resgrid-vo.org) is a VO that manages grid-relevant attributes for grid users, and issues grid security tokens. It supports multiple user organizations: Birch University; Contoso College; Fabrikam, Inc., and so on.

3 The Access Control Scenario

This paper focuses on a specific grid access control scenario. In the scenario, a grid user (Bob) must delegate a subset of his access rights for data files on the Birch file repository to a grid job he would like to run. This delegation must not allow access to Bob's other resources, and it should be short lived. This type of delegation is difficult to achieve while also meeting requirements for manageability, usability, scale, and robustness.

This paper describes a practical approach to the problem of delegated access rights. The approach is flexible enough to adapt to alternative operational requirements and system topologies. The core technology can also be used to address the full spectrum of policy and security token requirements needed to create an end-to-end access control solution. These requirements include:

- Describing explicit trust relationships
- Expressing security token issuance policies
- Providing security tokens that contain identities, attributes, capabilities, and/or delegation policies
- Expressing resource authorization and delegation policies

The approach also supports flexible revocation and auditing mechanisms; however, describing these is beyond the scope of this paper.

4 Security Policy Assertion Language

The Security Policy Assertion Language (SecPAL) represents the core technology underlying our approach to grid access control requirements. It is a declarative, logic-based, language that builds on a large body of work showing the value of such languages for flexibly expressing security policies (for example, [Bec05], [Con01], [DeT02], [Jim01], and [Bas05]). It was designed to be comprehensive and provides a uniform mechanism for expressing trust relationships, authorization policies, delegation policies, identity and attribute assertions, capability assertions, revocations, and audit requirements. This provides tangible benefits by making the system understandable and analyzable. It also improves security assurance by avoiding, or at least severely curtailing, the need for semantic translation and reconciliation between disparate security technologies.

4.1 SecPAL Features

The SecPAL design was motivated by limitations in existing and proposed approaches to grid and related distributed system security requirements. The result is a pragmatic, implementation-oriented, design that attempts to strike a balance between expressivity, usability, and ease of implementation. Some of the key features of SecPAL include:

- **Intelligibility –** It is important for any policy language to be understood by the intended audience and easy to relate to their intuitive ideas about security. SecPAL addresses this concern by using a definitional syntax that allows SecPAL assertions to be read as English language sentences. SecPAL's grammar is restrictive and requires the user to understand only a few verb phrase constructs with cleanly defined semantics. Finally, the algorithm for evaluating the deducible facts based on a collection of SecPAL assertions relies on a small number of relatively simple rules (see [SP06]).

  This approach eliminates some of the issues that have hindered adoption of other rich policy languages. Some languages (for example, XACML, XrMLv2, and SPKI/SDSI) have relied on a syntactic definition coupled with a lengthy, informal description of the intended semantics. These descriptions can be difficult to fully understand, and subsequent analysis has shown they have subtle ambiguities and complexities [Hum03, HW04, LM03a, and Aba98]. Logic-based languages (for example, Cassandra, Binder, SD3, PeerTrust, and so on) have formal, unambiguous semantics, but can be difficult to comprehend because they rely on traditional logic expression syntax or they assume knowledge of formal logic models such as Datalog.

- **Use of a common infrastructure** – Use of widely deployed system infrastructure makes it easier to implement and integrate SecPAL into existing systems. Most importantly, SecPAL uses XML syntax for the implementation, which provides a straightforward mapping from the formal model. This allows developers to use standard parsers and syntactic validation tools, along with the W3C XML Digital Signature and Encryption standards, to help ensure integrity, proof of origin, and confidentiality.
- **Support for distributed policy management** – SecPAL supports distributed policy authoring and composition. This allows it to support different operational models for authoring policies, as well as flexible separation of administrative duties. Support for digitally signed and/or encrypted policy objects allows for their secure distribution.
- **Efficient and safe evaluation** – Within SecPAL, simple syntactic checks on the SecPAL statements are sufficient to ensure that evaluations will terminate and produce correct answers. A number of research languages (Cassandra, SD3, and Binder) have also been shown to be decidable and tractable, though they may require a fairly complex safety analysis. Some other languages, such as XrML and SPKI/SDSI, may not be decidable.
- **Comprehensiveness** – SecPAL was designed to provide a complete solution for access control, supporting required policies, authorization decisions, auditing, and providing a public-key infrastructure for identity management. This is in stark

contrast to many other languages. For example, XACML focuses on authorization policies and assumes other mechanisms are used to convey identities/attributes and to establish trust relationships. SD3 and PeerTrust assume an external mechanism for establishing authenticated public key identities. Cassandra, Binder, and SPKI/SDSI define more complete solutions, but do not fully address revocation and audit requirements. This has important implications in real-world systems, because combining multiple security technologies often introduces subtle vulnerabilities and unintended behaviors due to differences in semantics and expressivity.

- **Sufficient Expressiveness** – SecPAL, like other declarative policy languages, has limits in what it can express. It was developed to handle grid security requirements, which are also applicable in many other types of distributed systems. SecPAL is adequately expressive for those purposes. SecPAL can be extended for different environments in ways that maintain the language semantics and evaluation properties. However, more general extensibility is not advisable in security-critical systems.

4.2 SecPAL Grammar (Partial)

This paper describes a simplified grammar based on the SecPAL formal model. This should allow the reader to understand the core concepts of SecPAL and follow the examples without having to read the formal specification (SP06]). The grammar presented here omits the revocation, principal *can act as* predicate, and audit constructs. It also reflects the concrete types used in the prototype implementation, rather than the generalized abstract base types used in the formal model[1].

The grammar uses the following conventions:

```
A       Constants, such as keys, uniform resource identifiers (URIs),
        strings, integers, date/time, and so on)
x       Variables that range over constants
|       Choice between alternatives
[]      Optional expression element
^, v, !  Logical AND, OR, and NOT operators
```

SecPAL statements are in the form of assertions made by a security principal. Security principals are most commonly identified by cryptographic keys so that they can be authenticated across system boundaries. This paper uses **K-A** as a shorthand for A's public key.

In their simplest form, an assertion states that the principal believes a fact is valid. They may also state that a fact is valid if one or more other facts are valid (restricted to the **existsFact** type defined subsequently) and/or some Boolean condition (c) evaluates to true. For example:

[1] SecPAL is designed to be extensible by deriving from the abstract base types in ways that don't alter the language semantics. This allows developers to define new principal identifiers, resource identifiers, attribute types, action verbs, qualifiers, and conditions that may be needed for a given environment.

```
assertion ::= Principal says fact [ if existsFact1 and ... and
  existsFactN and c] for some N ≥ 0

fact ::= principal verbPhrase

principal ::= A | x

resource ::= A | x

verbPhrase ::= can actionVerb resource [qualifiers]
  | possess attribute [qualifiers]
  | can say fact (unbounded, transitive delegation)

actionVerb :: = x | call | send | read | list | execute | write |
  modify | append | delete | install | own | actionVerb1, actionVerb2[2]

attributeType ::= rfc822Name | commonName | groupName | roleName |
  accountName | dnsName | ipAddress | deviceName | appName |
  organizationName | serviceName | accountId

attribute ::= x | attributeType=A | attribute1, attribute2

qualifiers ::= timespan
timespan ::= t1,t2 | T1,T2
```

In SecPAL, **facts** are statements about a principal. First, they can state that the principal has the right to exercise an action(s) on a resource. The supported actions in the prototype are defined by **actionVerb**. In this paper, all resources are identified by URIs and are assumed to follow a tree-based hierarchical naming convention typical of files systems, Web URLs, and so on. Additionally, SecPAL assertions can use the "principal possess attribute" fact syntax to express the binding between a principal identifier and an attribute(s). The implemented attributes are defined by **attributeType**.

Qualifiers might be included as part of these facts. Qualifiers allow the assertor to indicate environmental conditions (such as time, principal location and so on) that they believe should hold if the fact is to be considered valid. SecPAL cleanly separates an assertor's qualifier statements from a relying party's checks that are based on these values. This paper refers to time span qualifiers only, and shows how they can be used to control system behavior.

The final type of fact is defined by the **can say** verb. This provides a very powerful mechanism for expressing trust relationships and delegations (a form of Lampson's **speaks-for** operator [Lamp]). It allows one principal (A) to state its willingness to believe certain types of facts asserted by a second principal (B). For example, given the assertions "A says B can say fact0" and "B says fact0," you can conclude that A believes

---

[2] We allow writing facts of the form 'principal can actionVerb1, actionVerb2,…,actionVerbN resource' for compactness. This is semantically equivalent to writing N facts with each containing one of the actionVerbs. Similarly, facts containing multiple attributes are equivalent to multiple facts where each has a single attribute.

fact0 to be valid, and you can deduce that "A says fact0." This paper discusses the unbounded, transitive, form of such assertions only.

SecPAL also supports a bounded form that can control the number of allowed steps in a delegation chain. Note that it was not deemed necessary to support qualifiers for these types of facts, and their omission significantly simplifies the SecPAL semantics and evaluation safety properties.

Within SecPAL, you can either state concrete facts or use variables to write policy expressions. Variables in SecPAL are strongly typed and can be unrestricted (they can bind to any concrete value of the correct type) or restricted to a subset of concrete values based on a specified pattern. The prototype supports specification of patterns as regular expressions. In the simplified grammar, you use an identifier such as ***x*** to write an unrestricted variable, and you use a pattern such as **^ResGrid/\w+$** (that is, the string **ResGrid/** followed by one or more characters in [a-zA-Z0-9_]) to indicate a restricted variable.

A fact's validity can be based on an evaluation of other facts and conditions. Allowed conditional facts are defined by **existsFact**, as shown in the following code example.

```
existsFact ::= principal actionVerb resource [qualifiers]
      | principal possess attribute [qualifiers]

c ::= durationCondition
  | temporalCondition
  | x matches pattern (regular expression)
  | !c
  | c1 ^ c2

temporalCondition ::= t1 ≤ Current-Time() ≤ t2
durationCondition ::= t2-t1 ≤ Duration
```

Conditions are expressions that use SecPAL variables and/or functions that depend on the evaluation environment. This paper uses time-based conditional expressions only.

SecPAL decisions are based on evaluation of an authorization query against a collection of SecPAL assertions from applicable policies and security tokens. Queries are complex logical expressions that combine facts and conditions as shown in the following code example.

```
query ::= principal says existsFact
  | query1 ^ query2
  | query1 v query2
  | !query
  | c
```

This provides a very flexible approach to defining what is required for a given action to be authorized. Query templates form part of the SecPAL policy. They allow the policy author to declaratively state the appropriate query for different types of access requests.

5 Grid Access Control Using SecPAL

This section explores how SecPAL can be used to provide a comprehensive security solution for the access control scenario described in Section 3. The SecPAL assertions that would be encoded in the required security policies and security tokens are covered in some detail. These reflect a number of operational assumptions made about the scenario environment.

5.1 Federated Trust Relationships

Section 3 introduced the grid access control scenario used in this paper. Within this scenario, the first issue to address is the trust relationships between the three grid administrative domains (ResGrid VO, Birch U., and CHPC). You can assume that each administrative domain is independent and can decide which entities to trust and for what purpose. You can also assume that the domain members rely on the domain trust policy, although SecPAL is capable of expressing finer-grained trust distinctions within a domain. Figure 2 shows the domain STS relationships and trust policies described in this section.

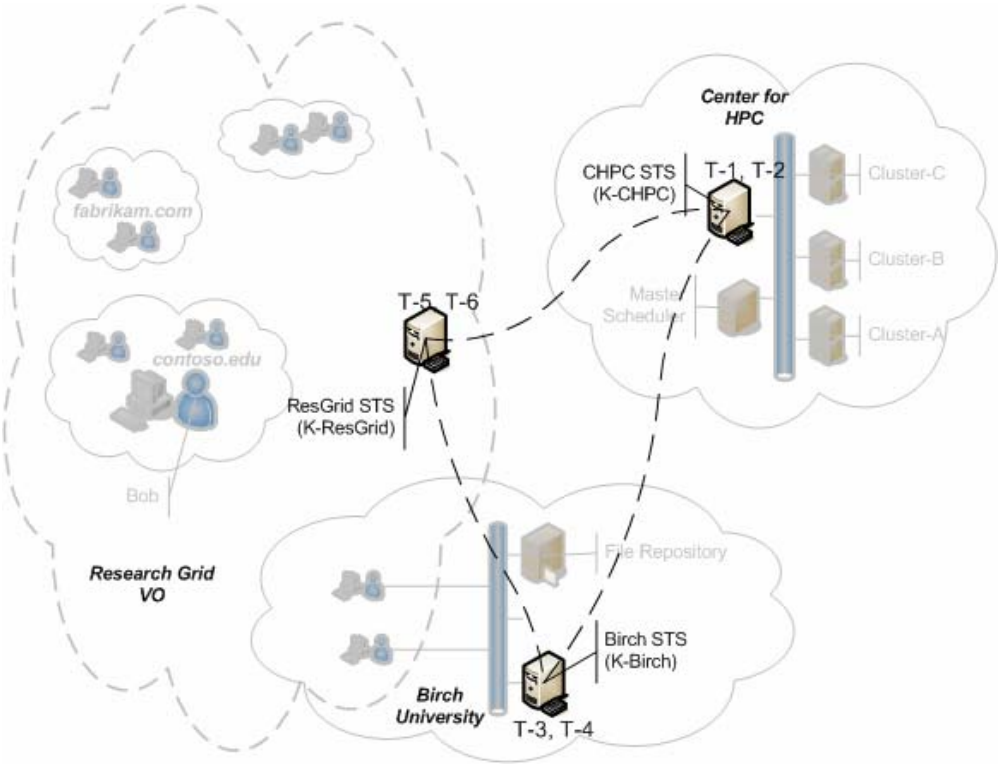

Figure 2. Grid domain relationships and federated trust policy

The CPHC domain trusts the Birch STS (K-Birch) for asserting Web service identities in the birch.edu domain. It also trusts the ResGrid STS (K-ResGrid) for asserting user identities and namespace-scoped attributes. It requires that any trusted assertions are used

only during the time span indicated by the asserting authority. CHPC federated trusts can be expressed as follows:

(T-1) K-CHPC says K-ResGrid can say *x* possess rfc822Name=^[-_.a-zA-Z0-9]+@ [-_.a-zA-Z0-9]+$, groupName=^ResGrid/\w+$, roleName=^ResGrid/\w+/\w+$ [*t1*,*t2*] if *t1* ≤ Current-Time() ≤ *t2*

(T-2) K-CHPC says K-Birch can say *x* possess serviceName=^http(s?)://\w+\.birch\.edu/\w+$ [*t1*,*t2*] if *t1* ≤ Current-Time() ≤ *t2*

As stated in section 4.1, SecPAL uses regular expressions to indicate a restricted variable. The patterns are kept simple in these examples and convey the expected form and namespace scoping of various attribute types (for example, group names of the form **ResGrid/<GroupName>,** similar to the approach described in [VOM04]).

This approach to specifying the specific assertion types that an external authority is trusted to issue differs significantly from the common practice of fully trusting certificate authorities and other servers that provide security-critical information. Other approaches to making fine-grained trust distinctions within grids were explored—such as use of X.509 name constraints—but they were not satisfactory.

Similarly, you can assume that the Birch domain trusts the ResGrid STS for asserting user information. It also trust the CHPC STS for asserting machine identities (domain name server [DNS] names in the c-hpc.com domain) and application identities associated with jobs. Birch federated trusts can be stated as follows:

(T-3) K-Birch says K-ResGrid can say *x* possess rfc822Name=^[-_.a-zA-Z0-9]+@ [-_.a-zA-Z0-9]+$, groupName= ^ResGrid/\w+$, roleName=^ResGrid/\w+/\w+$ [*t1*,*t2*] if *t1* ≤ Current-Time() ≤ *t2*

(T-4) K-Birch says K-CHPC can say *x* possess appName=.+, dnsName=^\w+\.c-hpc\.com$ [*t1*,*t2*] if *t1* ≤ Current-Time() ≤ *t2*

Users in the ResGrid VO trust the CPHC and Birch STSes to assert the identities of grid services in their respective domains. ResGrid federated trusts can be stated as follows:

(T-5) K-ResGrid says K-Birch can say *x* possess serviceName=^http(s?)://\w+\.birch\.edu/\w+$ [*t1*,*t2*] if *t1* ≤ Current-Time() ≤ *t2*

(T-6) K-ResGrid says K-CHPC can say *x* possess serviceName=^http(s?)://\w+\.c-hpc\.com/\w+$ [*t1*,*t2*] if *t1* ≤ Current-Time() ≤ *t2*

5.2 Principal Identification

In the prototype grid access control scenario, the SecPAL security tokens are issued by the previously identified domain STS, although this approach easily scales to a set of hierarchically structured authorities.

The SecPAL STSes must identify the entities that request tokens and decide which assertions should be encoded in the issued tokens. This is fundamentally the same problem faced by other token issuing authorities (such as X.509 certificate authorities and SAML authorities), and similar techniques can be employed. You could identify a token requestor by using out-of-band identity verification, administratively established accounts at the authority, or an existing identity management system. You could then use that identity to look up the appropriate attributes or capabilities for the entity in a secured directory.

For example, the prototype ResGrid STS uses X.509 identity credentials, which grid users receive from their primary organization. Figure 3 shows how grid security tokens are acquired.

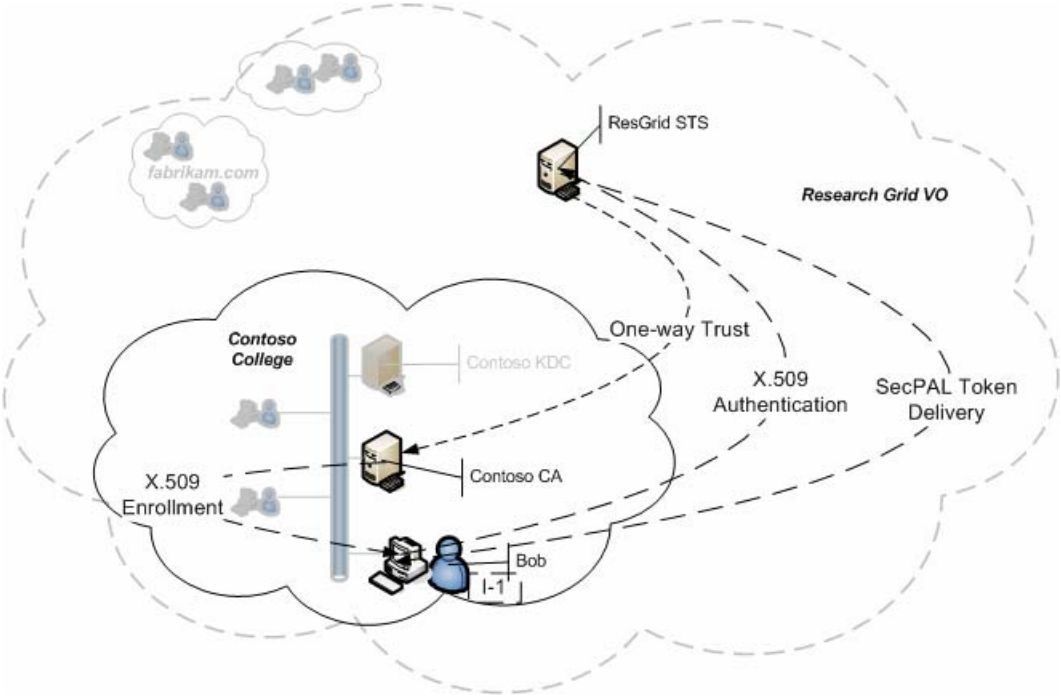

Figure 3. SecPAL Token acquisition using X.509 certificate authentication

The STS relies on the user attributes encoded in the X.509 certificate, along with attributes retrieved from a local directory (identified by K-Dir). SecPAL policy is used to control which attributes are encoded in a user's token. For example, the policy below uses e-mail addresses from the user's organization and group and role information from the local directory:

K-ResGrid says K-Contoso can say *x* possess rfc822Name=^[-_.a-zA-Z0-9]+@contoso.edu$

K-ResGrid says K-Dir can say *x* possess groupName=.+, roleName=.+

Note: You would need an assertion similar to the first one for each member organization

The STS uses the requestor's X.509 certificate to authenticate the request and validates it by using standard X.509 chain building techniques. If Bob is the requestor, the system would synthesize the e-mail and name attribute assertion based on the X.509 data, as follows:

K-Contoso says K-Bob possess rfc822Name=bob@contoso.edu, commonName='Bob Jones', organizationName='Contoso College'

The system would then retrieve Bob's attributes from the directory:

K-Dir says K-Bob possess groupName=ResGrid/ProjectX, roleName=ResGrid/ProjectX/Researcher

The STS can then evaluate a standard query for all known attribute types:

K-ResGrid says *x* possess rfc822Name=*e* ∨ K-ResGrid says *x* possess groupName=*g* ∨ K-ResGrid says *x* possess roleName=*r* ∨ K-ResGrid says *x* possess commonName=*c* ∨ K-ResGrid says x possess organizationName=*o*

The evaluation of this query returns all valid variable bindings that satisfy the query. Based on the input assertions from K-Contoso and K-Dir, the query would return valid bindings for ***x, e, g***, and ***r,*** with ***c*** and ***o*** unsatisfied. Depending on the input, there could be multiple valid bindings for a given attribute. If the ResGrid STS issues assertions that are valid for 1 month, then the token returned to Bob would contain the following:

(I-1) K-ResGrid says K-Bob possess rfcName=bob@contoso.edu, groupName=ResGrid/ProjectX, roleName=ResGrid/ProjectX/Researcher [9/12/2006, 10/11/2006]

The CHPC and Birch domains can issue security tokens to services in a similar manner. In the prototype, their key-based identities are entered into the STS by a system administer. Kerberos must authenticate the administrator. Typically, service identity assertions are valid for a longer time than those for users because of their more stable relationship with the grid. The example scenario uses a one-year time period.

5.3 Job Scheduling

Now that you can identify and authenticate the grid entities, you should understand how to authorize Bob to schedule a job. Figure 4 shows the entities involved in this activity and where the policies described in this section are stored.

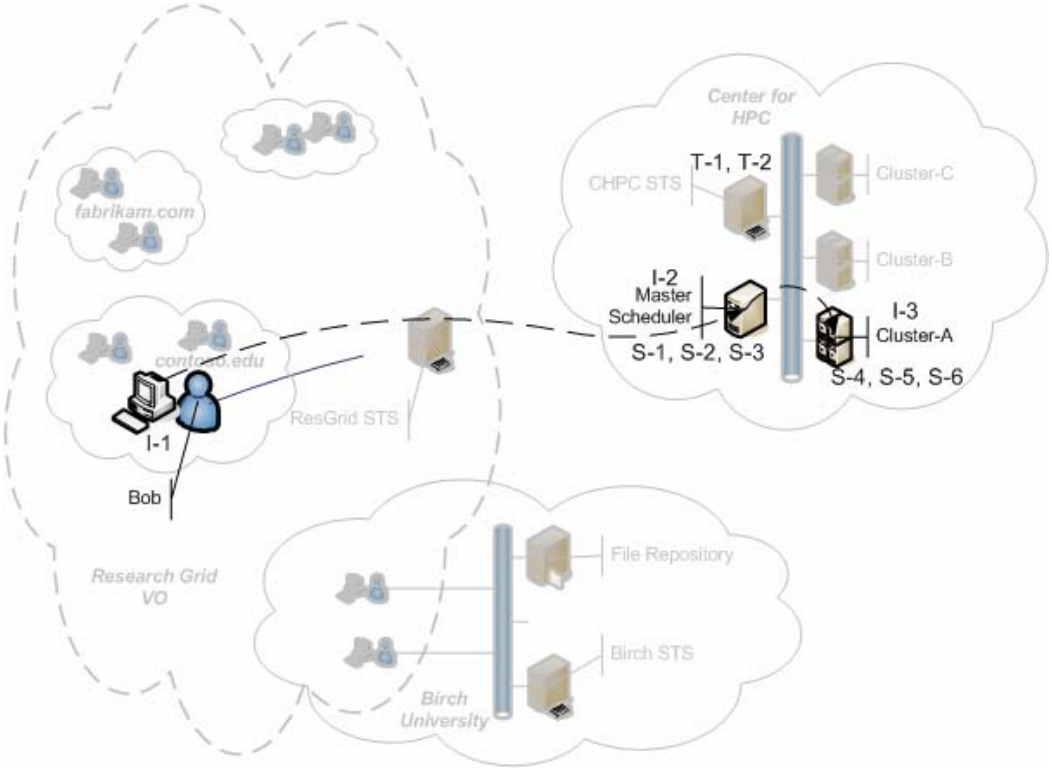

Figure 4. Grid job scheduling entities and security policies

In the example scenario, there is a single CHPC master scheduler (K-Sched), which Bob can access. This scheduler implements a simple authorization policy. Grid users who have a valid project group name and an associated role of Researcher can schedule jobs. Job information is isolated based on queues associated with each project. The policy governing access to each queue can be established based on known projects, or it can be dynamically managed based on incoming requests and completed jobs.

As discussed previously, the master scheduler has an identity security token that contains a **serviceName** attribute which is valid for 1 year:

> (I-2) K-CHPC says K-Sched possess serviceName=http://scheduler.c-hpc.com/job-manager [1/1/2006, 12/31/2006]

An example scheduler policy has the following elements:

- The scheduler trust policy, which consists of an explicit trust of the CHPC federated trust policy for attributes it cares about and an import of that policy. Together, these define the schedulers trust in ResGrid asserted group and role names.

> (S-1) K-Sched says K-CHPC can say *x* can say *y* possess groupName=.+, roleName=.+ [*t1*,*t2*]
>
> (T-1) K-CHPC says K-ResGrid can say *x* possess rfc822Name=^[-_.a-zA-Z0-9]+@ [-_.a-zA-Z0-9]+$, groupName=^ResGrid/\w+$, roleName=^ResGrid/\w+/\w+$ [*t1*,*t2*] if *t1* ≤ Current-Time() ≤ *t2*

- An additional trust policy for accepting CHPC STS-asserted service identifies, which is used to authenticate CHCP compute clusters.

> (S-2) K-Sched says K-CHPC can say *x* possess serviceName=.+

- An authorization policy that designates who can operate on the job queue for ProjectX, with similar policies for other active projects. (The example does not use the policy that would typically be present to allow for administrative access to all queues.)

> (S-3) K-Sched says *x* read, write, list, delete //queue/ProjectX if *x* possess groupName=ResGrid/ProjectX, roleName=ResGrid/ProjectX/Researcher

This will allow Bob to use his ResGrid issued token (I-1) to create a new ProjectX job or to manipulate an existing job. If you assume that the time condition on T-1 is satisfied, evaluation would combine I-1 with T-1 to conclude the following:

> K-CHPC says K-ResGrid can say K-Bob possess rfc822Name=bob@contoso.edu, groupName=ResGrid/ProjectX, roleName=ResGrid/ProjectX/Researcher [9/12/2006, 10/11/2006]

This can then be combined with S-1 to conclude the following:

> K-Sched says K-CHPC can say K-ResGrid can say K-Bob possess groupName=ResGrid/ProjectX, roleName=ResGrid/ProjectX/Researcher [9/12/2006, 10/11/2006]

This result combined with I-1 concludes the following:

> K-Sched says K-Bob possess groupName=ResGrid/ProjectX, roleName=ResGrid/ProjectX/Researcher [9/12/2006, 10/11/2006].

This deduction combined with S-3 \concludes that K-Bob is a valid binding for the variable '*x*' in S-3 and therefore:

K-Sched says K-Bob read, write, list, delete //queue/ProjectX

Bob could have his own trust policy, similar to S-1 and T-1, which would allow him to determine that K-Sched represents a valid scheduler based on scheduler authentication using I-2, as shown above.

In the scenario, the master scheduler needs to schedule a job with one of the cluster schedulers that will actually manage the execution. The scenario assumes that the cluster schedulers use a queue management and authorization approach that is similar to one that the master scheduler uses; however the cluster scheduler approach accepts requests from the master scheduler only. An appropriate policy would be the following:

(S-4) K-ClusterA says K-CHPC can say *x* can say *y* possess groupName=.+, roleName=.+ [*t1*,*t2*]

(T-1) K-CHPC says K-ResGrid can say *x* possess rfc822Name=^[-_.a-zA-Z0-9]+@ [-_.a-zA-Z0-9]+$, groupName=^ResGrid/\w+$, roleName=^ResGrid/\w+/\w+$ [*t1*,*t2*] if *t1* ≤ Current-Time() ≤ *t2*

(S-5) K-ClusterA says K-CHPC can say *x* possess serviceName=^http://\w+\.c-hpc\.com/job-manager$

(S-6) K-ClusterA says *y* read, write, list, delete //queue/ProjectX if *x* possess groupName=ResGrid/ProjectX, roleName=ResGrid/ProjectX/Researcher and *y* possess serviceName=^http://\w+\.c-hpc\.com/job-manager$

To assign Bob's job to ClusterA, the master scheduler uses its identity token (I-2) to authenticate and include the token used to create Bob's job (I-1). Evaluation proceeds as described previously, resulting in the following:

K-ClusterA says K-Sched read, write, list, delete //queue/ProjectX.

Assume that the cluster uses its token (I-3) to authenticate with the master scheduler, as follows:

(I-3) K-CHPC says K-ClusterA possess serviceName=http://clusterA.c-hpc.com/job-manager [1/1/2006, 12/31/2006]

You can combine this authenticated token with the policy S-2 to validate that K-ClusterA is a valid cluster.

5.4 Data Access and Constrained Delegation

This section explains how to delegate required data access rights to the executing job. Figure 5 shows the entities, security tokens, and policies described in this section.

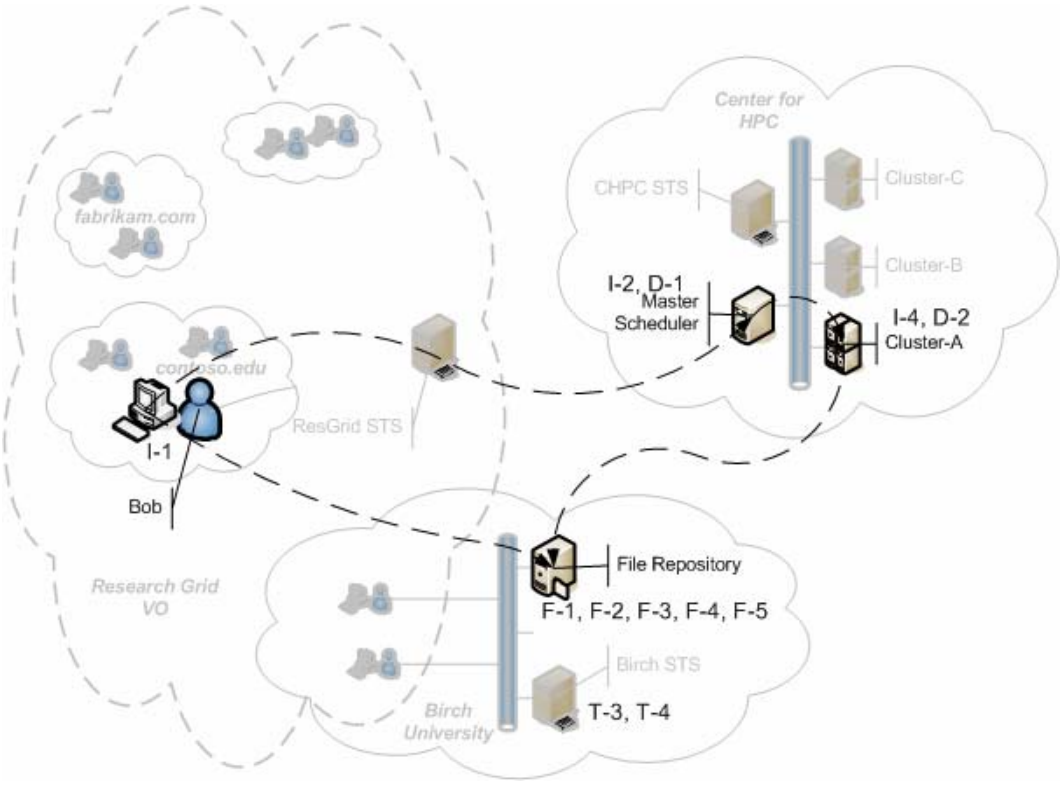

Figure 5. Delegated data access and associated security policies

The scenario assumes that Bob stores his job data on the file repository (K-FR) hosted by Birch. This repository stores data in a hierarchical file system, and uses standard URL notation to refer to a given file or directory (e.g., file://fs/ProjectX/foo.txt). The SecPAL evaluation engine understands resource hierarchies, making it is easy to define policies that apply to a node and all of its children.

The prototype uses a simple mechanism to control policy inheritance. A node (directory or file) may have a defined SecPAL policy. If no specific policy is defined, it finds the appropriate policy by going up through the ancestor chain until it finds the first node with an associated policy.

In the scenario, the file repository access rules allow users to create and manipulate files and directories scoped to their project affiliation. When a user creates a new directory, the system creates a new policy specific to their directory. This assigns ownership to the original user, and, by default, gives that user sole access to the contents.

The file repository uses the following general trust policy:

(F-1) K-FR says K-Birch can say *x* can say *y* possess rfc822Name=.+, groupName=.+, roleName=.+ [*t1*,*t2*]

(T-3) K-Birch says K-ResGrid can say *x* possess rfc822Name=^[-_.a-zA-Z0-9]+@ [-_.a-zA-Z0-9]+$, groupName=^ResGrid/\w+$, roleName=^ResGrid/\w+/\w+$ [*t1*,*t2*] if *t1* ≤ Current-Time() ≤ *t2*

(F-2) K-FR says K-Birch can say *x* can say *y* possess appName=.+, dnsName=.+

(T-4) K-Birch says K-CHPC can say *x* possess appName=.+, dnsName^\w+\.c-hpc\.com$ [*t1*,*t2*] if *t1* ≤ Current-Time() ≤ *t2*

This is combined with the authorization policy for the ProjectX directory node, as follows:

(F-3) K-FR says *x* read, write, list file://fs/ProjectX if *x* possess groupName=ResGrid/ProjectX

This is similar to the scheduler policies discussed in the section 6.

Based on this policy, Bob can request the creation of a new directory named file://fs/ProjectX/Bob, and can authenticate with his token (I-1). This action is authorized as follows: I-1 and T-3 indicate that K-Birch believes Bob's group affiliation, which combined with F-1, indicates that K-FR believes Bob's group affiliation, which combined with F-3, indicates that Bob has the right to write to file://fs/ProjectX.

When the new directory is created, a new policy that grants access rights to Bob only is created and attached to the node. This policy combines the trust policy statements (F-1, F-2, T3, T-4) above with the authorization policy:

(F-4) K-FR says *x* read, write, list, delete, own file://fs/ProjectX/Bob if *x* possess rfc822Name=bob@contoso.edu, groupName=ResGrid/ProjectX

Bob can now make a request to write his data files into this directory with F-4 defining the default authorization policy for all such files.

Bob can allow a job running on his behalf to access data files in file://fs/ProjectX/Bob. However, approaches that require modification of the policy for this file system node are impractical. Bob does not have a good way to uniquely identify his job before it runs, and requiring Bob to grant access, and then rescind it after the job completes does not scale.

A more practical approach is to allow specific types of delegation to occur within a general policy. This approach has the following prerequisites:

- The FR must agree to allow delegated access and must encode that as a policy.
- There must be a straightforward way for Bob to specify the rights he wants to delegate.

SecPAL addresses these prerequisites in an elegant manner. For example, if the file repository allows any user to delegate his or her file access rights to a job principal for at most 5 five days, an appropriate delegation policy would be:

> (F-5) K-FR says *x* can say *y v r* [*t1*,*t2*] if *x v r* and *y* possess appName=.+, dnsName=.+ and *t2*-*t1*<5Days ^ *t1*<Current-Time()<*t2*

This assertion would be added to each of the node specific policies in the file system. This policy says that a principal (***x***) can say that some other principal (***y***) can perform the actions (*v*) on a resource (*r*), provided *x* has the right to perform those actions on the resource. The file repository further restricts the delegation by requiring the principal ***y*** to have a trusted **appName** and **dnsName** attribute (used in the prototype to identify a job). This ensures that the file repository has identifying information about the delegate that it can audit and prevents arbitrary delegation to any grid principal if the system is compromised. The policy also says that a specific delegation is limited to 5 days and must be used during the asserted time span. Alternatively, the file repository—or an authority trusted by the file repository—could give each user a token that contains the policy which governs that user's ability to delegate. This can provide very fine-grained control without having to store extremely complex policies in the file repository.

Given F-5, how does Bob express delegation of his rights to files in his file repository directory when he cannot know when or where his job will run? One approach is to pass explicit delegation rights to a trusted intermediary. Assume that the master scheduler fulfills this role. (An alternative is for ResGrid to run a trusted service for this purpose. This operational mode has also been implemented in our prototype system). To do this, Bob sends a signed token that encodes the policy governing the delegation with the job scheduling information, as follows:

> (D-1) K-Bob says K-Sched can say *x* read,write,list file://fs/ProjectX/Bob [*t1*,*t2*] if *t2*-*t1*<5Days

Assume that the job is assigned to ClusterA. When ClusterA is ready to start Bob's job, it collects the required security tokens. It generates a key-pair (K-Job) to serve as an identifier for the job, and then requests the CPHC to provide a security token that asserts the application's name and the cluster machine identity (which is also valid for 5 days), as follows:

> (I-4) K-CHPC says K-Job possess dnsName=clustera.c-hpc.com, appName= 'Blast, Version=1.0' [9/12/2006, 9/16/2006]

It then asks the master scheduler to provide the necessary delegation(s):

> (D-2) K-Sched says K-Job read, write, list file://fs/ProjectX/Bob [9/12/2006, 9/16/2007]

The job would also be passed Bob's delegation to the scheduler (D-1) and to Bob's identity token (I-1).

The job can now send a request to read or write data in Bob's file repository directory, authenticating the request with its token (I-4) and including the delegation information (I-1, D-1, and D-2). The file repository will then try to answer the query, as follows:

```
K-FR says K-Job read,write,list file://fs/ProjectX/Bob
```

The evaluation proceeds by using Bob's token (I-1) in combination with the trust policy (F-1, T-3) to conclude that K-FR trusts Bob's attributes, and therefore Bob is authorized read, write, list, delete, own file://fs/ProjectX/Bob based on policy F-4. Using trust policy F-2 and T-4, you can deduce that K-FR believes the **dnsName** and **appName** attributes associated with K-Job. Combining these deductions with F-5, you can conclude the following:

```
K-FR says K-Bob can say K-Job read, write, list, delete, own file://fs/ProjectX/Bob
[t1,t2] if t2-t1<5Days ^ t1<Current-Time()<t2.
```

D-1 and D-2 allow you to conclude the following:

```
K-Bob says K-Job read, write, list file://fs/ProjectX/Bob [9/12/2006, 9/16/2007]
```

This when combined with the previous deduction allows you to conclude that K-FR believes K-Job has read, write, and list rights if the time condition is satisfied. Therefore, the query is satisfied and the access is allowed.

While SecPAL requires a somewhat complex explanation, use of SecPAL for constrained delegation is easier to comprehend than existing solutions that rely on multiple technologies. A similar explanation for commonly used X.509 proxy certificates requires a description of X.509 trusted root keys, validation and chain building rules, and proxy certificate chaining rules, followed by name/attribute extraction and their use in evaluation against some local authorization policy.

The approach described in this paper supports very precise control over what is delegated. In the example, K-Job can only exercise the explicit capability granted to Bob's files located in //fs/ProjectX/Bob. It does not implicitly inherit rights to other files or resources because its requests are associated with Bob's identity and attributes.

6 Conclusions and Future Work

This paper provides an introduction to a new security policy language, SecPAL. It explains how it can provide the foundation for a comprehensive solution that helps to secure access control in large-scale grid computing environments. This includes expressing fine-grained trust relationships, authorization policies, delegation policies, and

constrained delegation. The paper describes an example of how the language can support separation of duties, allowing independent authoring of domain trust policy and resource authorization policy, which are combined to create the effective resource policy. The paper also describes SecPAL's ability to serve as the basis for a public key infrastructure for authenticating principals, establishing their attributes, and communicating delegated capabilities.

The relatively simple scenario explored in this paper presents a number of real-world challenges not adequately addressed by currently deployed security mechanisms. SecPAL provides a straightforward solution that is easy to use and understand. It allows very powerful and flexible policies to be compactly encoded. The prototype system has proven to be easy to implement and integrate with mainstream technologies, such as XML, Web services, and compute clusters.

Microsoft's ongoing work in this area is focused on expanding the existing prototype to investigate additional security requirements and explore a number of important usability issues. This includes investigating the use of SecPAL for:

- Policy-directed renewal of short-lived delegations
- Explicit delegation of job management rights by a job owner
- Allowing an authority to delegate constrained rights to manage trust relationships
- Policy controlling the installation and execution of mobile code

In the usability area, Microsoft is investigating the safe automation of delegation generation. This is an important enabler, as few users are expected to have expertise in writing SecPAL delegations or analyzing allowed delegations. It should be possible to handle this for most cases based on the user's intent and knowledge of the resources involved. For example, given the knowledge a user is scheduling a job that must access a particular data source, you can query the data source for the appropriate delegation policy. It should be possible to inspect this policy and then generate a compatible delegation assertion(s)

Microsoft will report the results of this work in future papers.

Acknowledgements

Gregory Fee, Brian LaMacchia, and Jason Mackay of the Microsoft Advanced Technology Incubation group developed the SecPAL implementation and prototype grid environment upon which this paper is based. They also made significant contributions in evaluating alternative approaches to applying SecPAL to address grid security requirements.

Moritz Becker, Cédric Fournet, and Andrew Gordon of Microsoft Research made significant contributions to the definition of SecPAL and defined the underlying formal model and evaluation safety properties.

References

1. [SP06] Moritz Y. Becker, Cédric Fournet, Andrew D. Gordon. "SecPAL: Design and Semantics of a Decentralized Authorization Language." *Technical Report MSR-TR-2006-120*. Microsoft Research, September 2006.
2. [OG05] "Use of SAML for OGSI Authorization." *Global Grid Forum*, August 2005.
3. [Mul05] Mulmo, Olle. "Grid Security." The Globus Alliance, Presentation at SC|05.
4. [Wel03] Welch, V., et al. "Security for Grid Services." Presented at the Twelfth International Symposium on High Performance Distributed Computing. IEEE Press, June 2003.
5. [Fos98] Foster, I., et al. "A Security Architecture for Computational Grids." *Proceedings of the 5th ACM Conference on Computer and Communications Security Conference*, 1998.
6. [Shib05] "Shibboleth Architecture, Technical Overview, Working Draft 2." *Internet2/MACE*, June 8, 2005.
7. [VOM03] Alfieri, R., et al, "VOMS, an Authorization System for Virtual Organizations." *Proceedings of the 1st European Across Grids Conference*. Santiago de Compostela, Geb. 2003.
8. [VOM04] Frohner, Akos and Vincenzo Ciaschini. "VOMS Credential Format." *DataGRID*, IST-2000-25182, February 5, 2004.
9. [Bas05] Basney, J., W. Nejdi, D. Olmedilla, V. Welch, and M. Winslett. "Negotiating Trust on the Grid." *Proceedings of Semantic Grid: The Convergence of Technologies*, 2005.
10. [EGA05a] "Enterprise Grid Security Requirements v1.0." Enterprise Grid Alliance, Security Working Group, 8 July 2005.
11. [EGA05b] "Reference Model v1.0." Enterprise Grid Alliance, April, 13 2005.
12. [Aba98] Abadi, Martin. "On SDSI's linked local name spaces." *Journal of Computer Security*, 6(1-2):3–22, 1998.
13. [Bec05] Becker, Moritz Y. "Cassandra: Flexible trust management and its application to electronic health records." (Ph.D. thesis). *Technical Report UCAM-CL-TR-648*. University of Cambridge Computer Laboratory, 2005. http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-648.html.
14. [Con01] ContentGuard. "eXtensible rights Markup Language (XrML) 2.0 specification part II: core schema, 2001." *XRML Specifications*.
15. [DeT02] DeTreville, John. "Binder, a logic-based security language." *IEEE Symposium on Security and Privacy*, 2002: 105–113.
16. [HW04] Halpern, Joseph Y. and Vicky Weissman. "A formal foundation for xrml." *CSFW '04: Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW'04)*, 2004: 251.
17. [Jim01] Trevor, Jim. "SD3: A trust management system with certified evaluation." *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, 2001: 106–115.
18. [LM03a] Li, Ninghui and John Mitchell. "Understanding SPKI/SDSI using first-order logic." Computer Security Foundations Workshop, 2003.
19. [OAS] "Security Assertion Markup Language." *OASIS*, www.oasis-open.org/committees/security.

20. [WP05] Wu J. and P. Periorellis. "Evaluation of authorization-authentication tools: PERMIS, OASIS, XACML, and SHIBOLETH." *Technical Report CS–TR–935*. University of Newcastle upon Tyne, 2005.
21. [WS05] "Web Services Trust Language." Actional, BEA, CA, IBM, Layer 7, Microsoft, Oblix, OpenNetwork, Ping, Reactivity, RSA, VeriSign. February 2005.
22. [WS04] "Web Services Security: SOAP Message Security." *OASIS*, March 15 2004.
23. [Lamp] Lampson, B., M. Abadi, M. Burrows, and E. Wobber. "Authentication in distributed systems: Theory and practice." *ACM Trans. Computer Systems* 10, 4. November 1992: 265–310.
24. [Uni] "UNICORE Plus Final Report – Uniform Interface to Computing_Resources." *Joint Project Report for the BMBG Project*. UNICORE Plus, 2003.
25. [Hum03] Humenn, Polar. "The formal semantics of XACML (draft)." Syracuse University, 2003. http://lists.oasis-open.org/archives/xacml/200310/pdf00000.pdf.

