Certificate-Based Authorization Policy in a PKI Environment

MARY R. THOMPSON, ABDELILAH ESSIARI, and SRILEKHA MUDUMBAI
Lawrence Berkeley National Laboratory

The major emphasis of public key infrastructure has been to provide a cryptographically secure means of authenticating identities. However, procedures for authorizing the holders of these identities to perform specific actions still need additional research and development. While there are a number of proposed standards for authorization structures and protocols such as KeyNote, SPKI, and SAML based on X.509 or other key-based identities, none have been widely adopted. As part of an effort to use X.509 identities to provide authorization in highly distributed environments, we have developed and deployed an authorization service based on X.509 identified users and access policy contained in certificates signed by X.509 identified stakeholders. The major goal of this system, called Akenti, is to produce a usable authorization system for an environment consisting of distributed resources used by geographically and administratively distributed users. Akenti assumes communication between users and resources over a secure protocol such as transport layer security (TLS) to provide mutual authentication with X.509 certificates. This paper explains the authorization model and policy language used by Akenti, and how we have implemented an Apache authorization module to provide Akenti authorization.

Categories and Subject Descriptors: D.2.11 [**Software Engineering**]: Software Architectures, Policy Languages; D.4.6 [**Operating Systems**]: Security and Protection

General Terms: Security, Languages

Additional Key Words and Phrases: Public key infrastructure, digital certificates, XML

1. INTRODUCTION

There is a significant and growing set of distributed computing environments where the resources, resource stakeholders, and users are geographically and organizationally distributed. The DOE-sponsored collaboratories [Agarwal

An earlier version of this paper was previously published as "Authorization policy in a PKI environment," M. Thompson, S. Mudumbai, A. Essiari, and W. Chin in *Proceedings of the 1st Annual NIST Workshop on PKI*, April 2002.
This work is supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Mathematical, Information and Computation Sciences office (http://www.er.doe.gov/production/octr/mics), under contract DE-AC03-76SF00098 with the University of California. See the disclaimer at http://www-library.lbl.gov/teid/tmRco/howto/RcoBerkeleyLabDisclaimer.htm This document is report LBNL-51616.
Author's address: M. Thompson, A. Essiari, and S. Mudumbai, Lawrence Berkeley National Laboratory, MS50B-2239 1 Cyclotron Rd., Berkeley, CA 94720; email: {mrthompson;aessiari}@lbl.gov, srilek@yahoo.com

et al. 1998] and various "computational grids" [Foster and Kesselman 1999] are examples of these, as well as the ubiquitous Web-controlled sets of documents and services. These systems effectively define a *virtual organization* whose members and resources span many different real organizations. These virtual organizations need a way to authenticate and then authorize their users.

One of the characteristics of a collaboratory or Grid is that both the stakeholders and users may come from many different administrative domains. Thus, the virtual organization needs to identify its users in a domain neutral manner. The traditional candidates for cross-domain identities are Kerberos [Kohl and Neuman 1993] and PKI [Arsenault and Turner 2002]. Kerberos is mostly used within a single administrative domain, although there are many examples of cross-authenticated Kerberos realms, where the Kerberos administrators have agreed to accept tokens from another realm. Negotiating cross-realm agreements is often a lengthy and complex process. Some examples of such domains are universities, where there may be multiple Kerberos realms within the university, and the DOE's ASCI-DisCom^2 program [DISCOM] that connects Lawrence Livermore National Laboratory, Los Alamos National Laboratory, and Sandia National Laboratories in a computational Grid.

Looser collaborations, such as Grids based on Globus® [Foster et al. 2001] middleware, [IPG 2000; PPDG 2002], and collaboratories [Pancerella et al. 1999; NFC 2002] have chosen to use PKI identities to authenticate members. These organizations either run a certification authority (CA) of their own and/or accept certificates from a set of trusted CAs. Establishing trusted CA relationships can also be a lengthy process, but to the extent that the trust is by an individual resource provider rather than the whole site, and that many current collaboratories and Grids are experimental in nature, the trust relations have been established on an informal basis by the researchers, rather than the system security administrators. Once a collaboration has decided to use PKI identities to authenticate users, it needs to develop an authorization system using those identities plus some additional access policy information for all of its resources.

Another characteristic of collaboratories and Grids is that their resources, such as large scientific instruments, computing resources, and data stores, may have more than one person (called a stakeholder) who needs to control access to the resource. For example, when remote control of an instrument is allowed, the instrument administrators may want assurance that any user who can control the instrument has passed a local training course, while the principal investigator may be mostly concerned that the person controlling the instrument during his allowed time is a member of his research group. An authorization system that allows access policy to be defined independently and remotely from the resource gateway is desirable.

Standard access control methods typically use a central repository located at the resource site for authorization policy. While this centralization of policy on a secure host ensures that it can be trusted, it usually requires the stakeholder to have privileged access to the resource site in order to set the policy. Also such systems, to the extent that they use the underlying operating system for actual

access control, require that all users of a shared resource must have a local account on the system. The requirement for individual system accounts on the resource machine does not scale well.

We have developed the Akenti [Thompson et al. 1999] authorization system to meet these two needs: to use a virtual organization-wide user identity (in our case an X.509 public key certificate); and to facilitate setting access policy by multiple independent stakeholders remote from the actual resource gateway. The goal of the Akenti project is to provide a practical, easy to use, authorization service that meets the needs of collaboratories and computational grids.

The rest of this paper is organized as follows. Section 2 explains the authorization model and policy language that we use and compares this to some of the other early work in authorization. Section 3 describes how we have implemented an Apache authorization module to provide the same authorization policy and mechanism for resources accessed via a Web browser as accessed by other remote methods such as Globus job submission [Foster, et al. 2001]. Section 4 presents some performance measurements of our current implementation, and Section 5 compares Akenti to some newer distributed authority systems and standards.

2. AKENTI

Akenti assumes that X.509 certificates [Housley et al. 2001] and the SSL/TLS [Dierks and Allen 1999] connection protocols have been used to securely authenticate a user that is requesting access to a resource. It represents the authorization policy for a resource as a set of (possibly) distributed certificates digitally signed by unrelated stakeholders from different domains. These policy certificates are independently created by authorized stakeholders. When an authorization decision needs to be made, the Akenti policy engine gathers up all the relevant certificates for the user and the resource, verifies them, and determines the users' rights with respect to the resource.

2.1 Authorization Model

The Akenti model consists of *resources* that are being accessed via a *resource gateway* (the policy enforcement point—*PEP*) by *users*. These users connect to the resource gateway using the SSL handshake protocol to present authenticated X.509 certificates. The *stakeholders* for the resources express *access constraints* on the resources as a set of *signed certificates*, a few of which are self-signed and must be stored on a known secure host (probably the resource gateway machine), but most of which can be stored remotely. These certificates express the attributes a user must have in order to get specific rights to a resource, who is trusted to create use-condition statements and who can attest to a user's attributes. At the time of the resource access, the resource gatekeeper (PEP) asks a trusted Akenti server (the policy decision point—*PDP*), what access the user has to the resource. The Akenti server finds all the relevant certificates, verifies that each one is signed by an acceptable issuer, evaluates them, and returns the allowed access. See Figure 1.

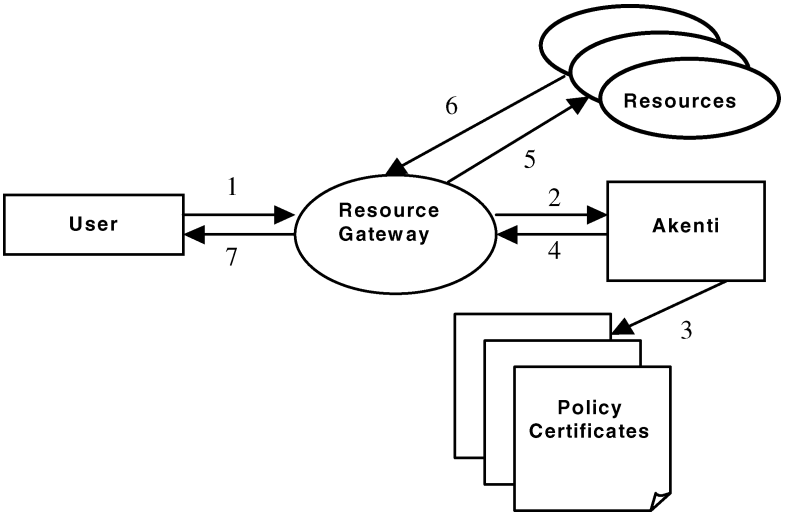

Fig. 1. Akenti authorization model.

There are several models for authorization systems. One is the *pull model* where the user presents only his authenticated identity to the gatekeeper who finds (pulls) the policy information for the resource and evaluates the user's access. A classic example of this is local file system access where the user id of the process that is attempting to reference the file is compared to the access control list of the file. The other general model is the *push model*, where the user presents one or more tokens or assertions that grant the holder specific rights to the resource. In this model, the gatekeeper has to verify that the user has the rights to use the tokens and then to interpret the rights that have been presented. The original examples of this model were capability-based operating systems where access to files and other objects was granted on the basis of unforgeable tokens, called capabilities, associated with a process [Levy 1984]. With the growing use of digitally signed certificates that can be verified for data integrity and authenticity, the push model is gaining wider usage. There are also hybrids of the two models, such as when a user presents identity information that includes restrictions on his full set of rights, or presents a handle to an authentication/authorization server from which the gatekeeper may pull information about the user and his rights.

We have mostly concentrated on the pull model in order to allow applications to use Akenti authorization over standard TLS connections that transport and verify X.509 certificates. We have also experimented with a push model where Akenti is contacted by the user and returns a signed capability certificate containing an authorization assertion consisting of a subject's distinguished name (DN), public key, the CA that signed for this name, the name of the resource, and the subject's rights. If this is presented by the user to a resource gatekeeper, along with an authenticated identity certificate, the gatekeeper needs only to verify Akenti's signature of the certificate and verify that the subject named in the capability certificate is the same as that in the identity certificate.

These capability certificates are short lived in order to avoid the problems of revocation.

2.2 Akenti Policy Language

Akenti policy is expressed in XML and stored in three types of signed certificates: *policy certificates, use-condition certificates*, and *attribute certificates*. Policy certificates specify the sources of authority for the resource. Use-condition certificates contain the constraints that control access to a resource. Attribute certificates assign attributes to users that are needed to satisfy the use constraints.

Policy certificates are self-signed, co-located with the resources to which they apply, and contain only minimal information since they are centrally located and may be administratively difficult to update. They are used to bootstrap and to provide closure for the trust chain by specifying the sources of authority for a resource. The sources of authority are the CAs who may sign X.509 certificates for all the principals involved in an authorization decision and the stakeholders who may issue use-condition certificates for the resource. Whenever a certificate is used, the Akenti policy engine will check that it has been signed by an acceptable issuer, and that the signature verifies. The CAs are represented by their X.509 certificates that provide a trusted copy of their public keys and information about where they publish certificates and certificate revocation lists. The stakeholders are represented by their distinguished names (DN) and the DN of the CA that issued a certificate for that name and a list of places, specified by URLs, where the stakeholders put the use-condition certificates that they issue. A policy certificate may optionally contain a list of URLs in which to search for attribute certificates.

Resources controlled by Akenti authorization may be grouped into a *resource realm*. A resource realm can be organized as a flat structure of resources such as instruments or compute platforms, or a hierarchical structure such as a file system or set of Web documents. Each resource realm has at least one policy certificate that must be stored in a known and secure place. Normally it is on the same machine that controls access to the resource, but it could also be on the platform where the Akenti server is running, if they are different. In the case of hierarchical resources, there must be at least one policy certificate at the top of the tree (referred to as the root policy). In addition, there may be a policy certificate at any level where there are new stakeholders, or restrictions on the allowed CAs. Levels without their own policy certificates inherit policy from higher levels.

Each stakeholder group for a resource must create at least one and possibly more use-condition certificates for the resource. A use-condition certificate consists of a constraint, which is a relational expression of the attributes a user must have to get a certain set of rights and a list of the principals who can attest to the required attributes. Components of the X.509 distinguished name can be used as attributes such as *CN = Mary R. Thompson*, or *O = Diesel Combustion Collaboratory*, or attributes can be defined in the context of the resource. For example, *role = researcher* or *group = accounting*. These attribute requirements

can be combined with the Boolean operators *&&* or ||. Negative permissions such as *group* ! = *accounting* are not supported because of the difficulty in requiring all the relevant attribute certificates to be found. It is also possible to specify real-time or system attributes such as *time<=5PM && time> =9AM*, or *system_load <2*. If Akenti is unable to evaluate such system attributes it may return them to the resource gateway for evaluation. An attribute authority (consisting of an issuer and its CA) is specified as the signing authority for each attribute-value pair. See Figure 2 for an example of a use-condition certificate.

A stakeholder may put use-condition certificates in more than one place for reliability, but each directory must contain the complete set. Since use-conditions restrict access to a resource, it is essential that either all or none of them be found. If no use-conditions are found for a stakeholder group, all access to the resource is denied. This is not the case with attribute certificates since they only serve to increase access. Thus a missing attribute certificate may limit or deny a user's access, but will never allow an access that should be denied.

Attribute certificates contain an attribute-value pair and the principal to whom it applies. They are signed by attribute authorities that have been specified in a use-condition certificate. Attributes can apply to more than one resource, although they are likely to be applicable in only a single resource realm. Akenti attribute certificates are XML documents rather than the proposed IETF ASN.1 attribute certificates (see the section on Related Work for a more detailed comparison). The complete XML schema for Akenti certificates can be found on the Akenti Web site [AKENTI.XSD 2003].

2.3 Comparison to Earlier Authorization Policy Languages

A language to describe access policy typically involves making statements about some or all of the following elements: requestor identity, grantor identity, a set of access rights, and a set of constraints [Ryutov and Neuman 2000]. The target resource to which the rights apply may be explicit in the policy statement or may be implicit in the context. The identities may be expressed as names or as public keys. The access rights are usually arbitrary strings whose meaning is agreed on between the policy creator and the PEP. Constraints can be expressed as a set of tokens, as a Boolean expression, or in a special purpose language. The names of the constraints, for example, group, role, time only need to be self-consistent within the policy, the attribute assertions, and the PEP.

Two basic types of authorization policy statements are authorization assertions, including rights delegation or capabilities, used in push model authorization systems and resource-centric access policy statements used in pull models. The paper of Lampson et al. which describes a formal theory for authentication in distributed systems defines a "speaks for" relationship that describes the delegation of rights [Lampson et al. 1992]. The paper shows how such delegation of rights can be used to describe many of the current access control methods. Neuman's paper on proxy-based authorization [Neuman 1993] gives a clear description of various kinds of proxy certificates: full proxies, restricted proxies,

```
<?xml version="1.0" encoding="US-ASCII"?>
<AkentiCertificate xmlns:ak="http://www-itg.lbl.gov/akenti/docs/AkentiCertificate" >
 <SignablePart>
  <Header Type="UseCondition" SignatureDigestAlg="RSA-MD5" CanonAlg="Ak1CanAlg"
Version="2">
    <UID>"host.lbl.gov#104b8965#Mon May 07 17:04:23 PDT 2001"</UID>
    <Issuer>
      <UserDN> /C=US/O=Lawrence Berkeley National Laboratory/OU=ICSD/CN=Mary R.
Thompson </UserDN>
      <CADN> /C=US/O=Lawrence Berkeley National Laboratory/OU=ICSD/CN=IDCG-CA
</CADN>
    </Issuer>
    <ValidityPeriod Begin="010508000421Z" End="040506080443Z"/>
  </Header>
 <UseConditionCert scope="local" critical="false">
  <ResourceName> LBL </ResourceName>
  <Condition>
    <Constraint>( o=Lawrence Berkeley National Laboratory | | ( group = distrib ) ) </Constraint>
    <AttributeInfo type="X509">
          <AttrName> o </AttrName>
          <AttrValue> Lawrence Berkeley National Laboratory </AttrValue>
          <CADN> /C=US/O=Lawrence Berkeley National Laboratory/OU=ICSD/CN=IDCG-CA
</CADN>
    </AttributeInfo>
    <AttributeInfo type="AKENTI">
          <AttrName> group </AttrName>
          <AttrValue> distrib </AttrValue>
          <Principal>
            <UserDN> /C=US/O=Lawrence Berkeley National
Laboratory/OU=ICSD/CN=Srilekha Issuer </UserDN>
             <CADN> /C=US/O=Lawrence Berkeley National Laboratory/OU=ICSD/CN=IDCG-
CA </CADN>
          </Principal>
    </AttributeInfo>
  </Condition>
  <Rights> read, write </Rights>
 </UseConditionCert>
</SignablePart>
```

Fig. 2. Use-condition certificate.

and bearer and delegated proxies. It shows how restricted proxies can be used to combine restrictions from access control lists with the ability of a trusted principal to delegate its privileges to another principal. The CRISIS security architecture [Belani 1998] implemented these ideas in *transfer certificates* in which one principal can delegate a subset of its rights to another principal. The

transfers are expressed as a list of capabilities. There can be a chain of transfer certificates delegating a more limited set of capabilities to additional principals.

Simple public key infrastructure (SPKI) is an IEFT experimental protocol that defines, among other things, a simple authorization certificate that can be used to allow one principal (or authority) to delegate rights to another principal [Ellison 1999]. An *authorization certificate* binds an authorization to a key and is expressed as a 5-tuple consisting of issuer, subject, delegationallowed (true or false), authorization (an S-expression), and validity dates. The subject is represented either by its public key, or by a name that is bound in a *name certificate* to a public key. One of the interesting ideas of SPKI and other trust management systems was to focus on the need to authorize a user's actions without first authenticating the user's name. The certificates being used form a chain of delegated rights that eventually chain back to a known authority.

The other common style of access policy is to define the access conditions for a resource. This often takes the form of an extended or generalized access control list. These lists can consist of an ordered set of elements that contain information about what users or groups have what access. In the extended ACL used by the IETF's GAA protocol [Ryutov and Neuman 2000] and the Prospero Resource Manager [Ryutov et al. 1999], each token consists of a type, a defining authority, and a value. Some token types are access identity, a set of positive rights, or the value of a constraint. In the GACL specified by Woo and Lam [1993], there is one element for each resource, but it can specify rights for groups, individuals, and inherited rights from other objects.

The emphasis in these early systems was to produce compact information structures that were easily machine parseable. The KeyNote system [Blaze et al. 1999] expanded authorization assertions to a keyword/value format to make the certificates more human readable. Here the conditions are also written as Boolean expressions. The first version of Akenti also used keyword/value pairs and Boolean constraints. We eventually switched to XML, as the complexity of the use-conditions required too much information to be hidden implicitly in the ordering of the fields, thus defeating our goal of easily understood certificates (see the Related Work section for a more detailed comparison of KeyNote and Akenti).

In the context of the previous discussion, the Akenti static certificates contain resource-centric access policy including trust relationships and attribute assertions. Akenti returns its access decision in a dynamic short-lived capability certificate, signed by the Akenti server. This certificate can be used as a delegation of rights proxy, where the bearer is the user who made the authorization request, and the grantor of the rights is the Akenti authorization server. XML was the choice for the second implementation after the desired policy expressions outgrew the simple ordered keyword/value pairs syntax. Using XML makes the certificates more human readable at the expense of compactness. Now that XML parsers and standardized methods for extracting information from DOM trees are readily available, XML can be as easily, if not as efficiently, machine parsed and interpreted. Of course, the amount of code needed to perform this is greater than that needed for simpler or more rule oriented languages.

2.4 Creating Policy

Since policy is contained in signed XML certificates, which are interdependent, a stakeholder needs some tools to assist in the creation of certificates. A stakeholder starts by creating the root policy certificate for the resource realm. The X.509 certificates of all the trusted CAs must be available from a trusted source and are placed in the root policy certificate. This certificate also contains the URLs of the locations where these CAs publish certificates and certificate revocation lists. The first stakeholder must decide if there are other stakeholders for the resource and, if so, include their DNs and CAs in the root policy certificate. In a hierarchical set of resources, only the top-level stakeholders need to be known initially. They in turn, can delegate control to other stakeholders for resources lower in the hierarchy.

Akenti certificates can either be created by a command line tool that signs an XML input certificate, or by a GUI program that steps a stakeholder through a menu of choices for each field in the certificate. The GUI program is supported by a resource definition server running on the resource host that in turn reads a resource definition file and any existing policy certificates to find stakeholder names, acceptable attributes, and actions for a resource realm. The command line method is fine for very simple policy, and for the root policy certificate, but as soon as the policy becomes hierarchical, or there are many stakeholders, the GUI interfaces that prompt the stakeholder with acceptable choices become preferable. The resource definitions file is only used to provide suggestions to the policy creation GUIs. It includes the names of the CAs, and their publishing directories, attribute names and values, the principals that are acceptable for issuing specific attribute values, and a list of actions that are relevant to the resource realm. In summary the two methods of getting started are

- Create an XML version of a root policy certificate, following one of the templates provided by the Akenti distribution, and use a command line program to sign it with the stakeholder's private key contained in a pkcs12 format file, and store it in the resource tree.
- Create a resource definition file, start the resource definition server, and then use the GUI program to create, sign, and store a policy certificate.

The stakeholder must now create at least one use-condition certificate for the resource. Anyone can create a use-condition certificate, but it will only be used during authorization if it is issued and signed by one of the stakeholders currently listed in the resource's policy certificate.

Use-condition and attribute certificates can also be created by a command line interface or the GUI certificate generator. In creating a use-condition certificate, the stakeholder will be presented with a menu of possible stakeholders for the resource (of which he must be one), previously defined attribute/value pairs and their allowed attribute authorities, and the defined actions for the resource. The stakeholder is also asked about such details as the length of time for which this certificate should be valid; the scope of the use-condition (does it just apply to the one resource or to a hierarchy of resources); and whether it is

a critical use-condition (it must be satisfied or the user gets no access to the resource even if she satisfies other use-conditions). The use-condition certificates must be stored in a directory that is specified in the policy certificate.

When creating an attribute certificate the stakeholder will be presented with a list of defined attributes and values for the resource realm.

Once a set of policy, use-condition and attribute certificates have been stored, the stakeholder can use a Web-based interface to see what access is allowed to the resource.

2.5 Checking Access

The Akenti authorization service can be called in several ways: it can be invoked as a function call by the PEP and thus run as part of the gatekeeper. It can be contacted as a server through TCP or TLS, and it will return a signed capability certificate. If an insecure protocol is used, the gatekeeper must have a copy of the Akenti server's public key and verify the certificate, before it can trust the information. The Akenti server always returns a signed capability certificate that may include both conditional and unconditional rights. Conditional rights are rights that may have some conditions attached that only the PEP can evaluate, such as current machine load, disk availability, or the state of some related system variable. We provide an API wrapper that will extract the unconditional actions and return them as strings, and will parse and evaluate the runtime conditions calling an evaluator function provided by the PEP.

As has been mentioned previously, the Akenti policy engine finds all the use-conditions by searching in the URLs specified in the policy certificates and verifying the issuer and signature on each certificate. If a use-condition certificate cannot be found for each stakeholder group, access to the resource is denied. Attribute certificates are searched by following URLs in either the policy certificates and/or use-conditions. Again, the issuer and signature of each certificate is verified. This signature verification requires that the Akenti policy engine be able to find the X.509 certificates for each issuer. If the CAs who issue certificates publish them in an LDAP server, Akenti will look there. Otherwise, there must be some setup actions taken to put all the expected certificate issuers' X.509 certificates in a file system or at a location specified by a URL. Akenti caches all the certificates that it finds in order to reduce subsequent search time. It also caches the authorization decision as a capability certificate that contains the access rights of a user for a resource, so that subsequent requests for the same resource by the same user require no repeated decisions. The lifetime of the cached certificates is set in the policy certificate for the resource.

3. MOD_AKENTI MODULE FOR APACHE WEB SERVER

Web-controlled sets of documents and services have rapidly grown from collections of read-only documents that are centrally administered to a vast array of remotely managed documents and services. In the scientific community such Web-based systems are increasingly used to provide a common interface to

static documents, to allow shared authoring of documents, to allow access to legacy data bases, to allow execution of codes on shared server machines, and practically anything else an inventive scientist can think of. Authentication to perform such access is usually implemented by the http *basic authentication* mechanisms, (e.g., user/password or domain based) or by ad hoc scripts based on the username. In the standard HTTP protocol these passwords are passed across the Internet in clear text and are thus deemed insecure.

In order to make Akenti authorization available for the widest range of distributed resources, we wanted to make it available to Web-accessed resources. There were several ways to accomplish this: referencing resources through CGI scripts that called Akenti, referencing resources through Java servlets or JSPs that called Akenti, or building Akenti authorization into a Web server. The first two methods involve an indirection between the request and response that is both less efficient and requires more complicated URLs to refer to documents. Since the Apache Web server makes it straightforward to include new functionality, we decided to build an Akenti module for Apache.

The Apache [APACHE 2002a] Web server is a widely used, high-performance freeware server which is built around an API [Thau 2002] that allows third-party programmers to add new server functionality. Indeed, most of the server's visible features (logging, authentication, access control, CGI, and so forth) are implemented as modules, using the same extension API available to third parties. The modules can be statically or dynamically linked to the server [Wainwright 2001].

3.1 How Apache Modules Work

Apache divides the handling of requests into different phases:

- URI to file name translation
- Authentication and access checking
- Determining the MIME type of the requested entity
- Returning data to the client
- Logging the request

Each module can contribute to any of these phases. For each phase, a module can completely replace an existing module or can be added to a list of existing modules. The list of modules acts as a queue in which control is passed from one module to another. Each module can return one of three values: OK, DECLINE, and FORBID. If a module returns OK, then the server passes the request on to other modules in the queue. A module returns DECLINE when it wishes to ignore a specific request. A FORBID return overrides other module's replies and causes the server to forbid access to the resource requested. Each module can declare a set of handlers to handle specific types of URI requests. The interface between the server core and the extension modules is through a module structure that consists of a vector of callback routines. A module provides a callback for each phase that it wishes to handle and NULL for the rest. The module structure for Apache 1.3.x allows a module to define a number of different callback routines. Mod_Akenti defines only three of the possible procedures:

two to handle reading directives from the configuration file and one to check access.

Apache allows each module to read directives from the configuration file by specifying a command table structure. The entries in the command table include the name of the command, a pointer to the command handler, and an argument that is passed to the command handler.

3.2 How Mod_akenti Works

Mod_akenti is an Apache module that provides Akenti authorization capabilities for the Apache Web server. Mod_akenti is implemented as a dynamic shared object module which can be loaded into the server at start-up or restart time. It currently works in Apache 1.3.x. Mod_akenti does not define any handlers as it serves as an access control mechanism for all requests to the Web server unless otherwise specified.

Mod_akenti defines two global directives inside the server configuration, and defines a check access callback. Thus, its interface consists of a call for per-directory configuration, a command table, and a callback for the check access routine.

The two Akenti directives are AkentiConf, which supplies the name of the configuration file used to configure the Akenti policy engine, and AkentiResources, which is used to specify what part of the document tree should be controlled by Akenti. The second directive is of interest as it allows other authorization mechanisms to coexist with that of mod_akenti. It accepts a set of resource names to be controlled, or "ALL" to control the whole hierarchy or an empty argument to control none of the resources. The Akenti module requires a secure Apache Web server (Apache + mod_ssl) with client-side certificates required. Thus, the Web server authenticates the client's X.509 certificate before mod_akenti is called. The mod_akenti distribution package [Mudumbai 2002] provides detailed information about how to build and configure the Akenti module.

3.3 Web Authentication and Authorization Methods

Standard Web authentication and access control are based either on the domain in which the request originated, or something called basic authentication [Franks et al. 1999], where the user provides a user name and password that the Web browser matches against user information stored on the server machine. There are many authentication modules for Apache based on this mechanism [APACHE 2002b]. Mod_auth is the basic module that matches a user and password with an entry in Web specific password and group files. Modules such as mod_auth_dbm and mod_auth_db provide greater scalability by looking up users in a database. There are also modules available for authenticating users in ldap directories, Oracle, and mysql databases and Kerberos users. In all of these schemes the user name and password are passed over the network in plain text. There is one other form of user authentication, called digest authentication, which is implemented by mod_auth_digest. This protocol has the server send a nonce to the browser who then returns an MD5 hash of the nonce,

```
<Directory /foo>
    SSLRequireSSL
    SSLRequire %{SSL_CLIENT_S_DN_O} eq
                                "LBNL" and
        %{SSL_CLIENT_S_DN_OU}
                                in      {"DSD",      "ICSD",
"NERSC"}
    </Directory>
```

Fig. 3. Example of SSLRequire directive.

the user name, password, http request, and the URI. Thus, the password is not sent in the clear. Unfortunately this method is not supported by many browsers and so is not in common use.

When mod_ssl [MODSSL 2002] is added to the Apache Web server, the communication between the client and server becomes HTTP over SSL. In the typical commercial use of SSL, only the server is required to have an identity certificate and private key that is used to establish the encrypted communication channel. However, SSL can run in a mode that requires the client to present a certificate and demonstrate its possession of the private key. When this mode is used mod_ssl can provide access control based on the client certificate.

Mod_ssl can implement a *FakeBasicAuth* option where it uses the subject of the client's X.509 certificate as a user name, but no password needs to be obtained from the user since the SSL handshake has verified the client certificate. It also provides a directive called SSLRequire (see Figure 3) that specifies constraints that need to be fulfilled in order to allow access. The requirement specification is an arbitrarily complex Boolean expression containing any number of access checks. The variables used in the expression include all the standard CGI/1.0 and Apache variables, plus a large number of variables defined by mod_ssl that refer to parts of both the server and client certificates: for example, client subject's DN, the client issuer's DN, and most components of the client's certificate. The syntax also allows an expression to be used from an arbitrary file. This method is used to match portions of distinguished name compared to the FakeBasicAuth where the whole DN is used.

While the SSLRequire directive is very powerful and can express many of the same constraints that Akenti does, it is limited by the fact that constraints are specified as part of the server's configuration file. If many resources need to be controlled, the server configuration file will expand to the point where it becomes difficult to manage. In distributed environments where policies for resource access are managed by multiple owners, a centralized access control list does not scale well.

For example, WebDAV [Goland et al. 1999] has been implemented as Apache module, mod_dav, which allows extensions to the HTTP protocol in order to provide a shared file system. If several projects need to be managed by one server, there should be a way to limit the writing of access policy for a set of resources to the project manager. But since all the policy is in one file, this is not possible.

Table I. Average Times to Fetch a Document from a Secure Akenti Server

	No Caching			Caching		
File (bytes)	Akenti (s)	SSL/Network (s)	Total (s)	Akenti (s)	SSL/Network (s)	Total (s)
Minimum Policy						
1 KB	0.142 (0.0033)	0.481	0.623 (0.007)	0.00630 (0.000145)	0.458	0.464 (0.0101)
1 MB	0.142 (0.001)	1.221	1.363 (0.0047)	0.00588 (0.000044)	1.193	1.199 (0.0125)
Typical Policy						
1 KB	0.233 (0.014)	0.486	0.720 (0.019)	0.00587 (0.000026)	0.455	0.461 (0.015)
1 MB	0.228 (0.004)	1.220	1.448 (0.007)	0.00613 (0.000032)	1.202	1.208 (0.016)

Mod_akenti, on the other hand, stores all of its policy information outside of the Web server configuration file in the normal Akenti certificate set. The only information in the Apache configuration file is the name of the resources that mod_akenti wishes to control and a pointer to Akenti's own configuration file. Thus, the same access policy certificates can be used for resources referenced via the Web or by another remote method. Certificate caching is especially important for Web accessed resources which tend to be accessed in clusters.

4. PERFORMANCE MEASUREMENTS

The performance of Akenti authorization is typically dominated by the time it takes to find and retrieve the policy certificates. Nevertheless, it is valuable to measure the system performance in order to characterize and to optimize it.

Two sets of measurements are of interest to potential users of Akenti. The first is the time required to make an authorization decision. As mentioned above, this time is strongly dependent on the number of credentials that are required and where they are stored. The second metric of interest is to compare the time to access a file via a secure server, such as SSL-Apache/Akenti, versus the access time using an unsecure server such as plain Apache. In the latter case, additional variability arises from the network transmission time between the client and server including time associated with establishing an SSL connection and encrypting the data transfer.

The measurements in this paper are for file fetches between a simple command-line client and an Akenti/Apache Web server. The client, server and all the certificate servers are on a 100 Mb/s LAN. The document sizes varied between 1 KB and 1 MB in order to evaluate the overhead of the SSL encryption. The Akenti code was run on both a dual processor 450 Mhz RedHat Linux box and a 450 Mhz Sun Ultra 60 running Solaris 5.7. The numbers in Table I are for the Linux system. The Sun times were about 10% higher.

We measured the access times for two different access policies. The first was a minimum policy consisting of one policy certificate, one use condition certificate, one attribute certificate, and two identity certificates. This represents the case of one stakeholder for a resource who limits access to the resource to members

of a group. The stakeholder is also the issuer of the group attribute certificates. The second policy that we measured represents a more typical policy, where there are two stakeholders, and a separate attribute authorizer. This requires one policy certificate, two use-conditions, two attribute conditions, and four X.509 certificates. We also took measurements with and without Akenti server certificate caching enabled.

On the server side, Akenti does extensive logging of each logical step in the policy engine. This measurement excludes the server side time spent in the Apache server and SSL encryption. The times in Table I are the times in the Akenti policy engine (Akenti), the total socket read time the client saw (total) and the difference between the two which can mostly be accounted for by SSL overhead (SSL/network).

The test program fetched the same file five times and then calculated the mean fetch time. The standard deviation of the measurements is shown in parentheses. The data from the client program was combined with the matching server log entries to determine the values in Table I.

Several observations can be made from this data. As the files get bigger, the SSL encryption times tend to dominate. However, SSL can be configured to do only authentication and message integrity checking if encryption is not needed, which would reduce this time. As more certificates are required to grant access, the times in the Akenti policy engine increase. We can see from the Akenti log files that the major categories of time in the policy engine are fetching certificates and verifying signatures. In the minimum certificate case 89.2 ms or about 62% of the total time was spent fetching certificates. In the more typical case, 156 ms or about 67% of the time was in fetching certificates. In each case about 18 ms was spent in certificate verification. The rest of the time was spent parsing the use-conditions, signing the final capability certificate and general program overhead.

In the case when caching is enabled and a valid capability certificate is found, the time in the policy engine is about 6 ms. The caching lifetimes of cached use-condition and identity certificates is generally longer than that for capabilities, so there is an intermediate case that we did not measure where cached versions of those certificates are used to reestablish the capability.

For a Web server that is mainly fetching documents, caching by the Akenti policy engine provides a big performance benefit, since there are usually several clustered accesses to documents in the same general protection domain. If Akenti is being used by a server where the pattern of accesses is isolated, the caching may actually be a disadvantage, since cache misses and subsequent cache updates are relatively costly

In order to put the Akenti/Apache Web server in context, we measured the fetch times in our environment from a standard, non-SSL-enabled Apache server. The corresponding fetch times for 1 KB and 1 MB files using the same client program but a standard Apache server with no access control was about 4 ms for the 1 KB file and about 213 ms for the 1 MB file. (See Table II.) For Web servers, that most meaningfully compares to the 0.462 and 1.21 s times for an average set of access constraints from a caching Akenti server. Since Akenti is caching, most of this time is due to using SSL. Obviously, the target

Table II. Document Fetch with and Without Akenti Access Control

	With Akenti/SSL	Without Akenti/SSL
1 KB	0.461	0.00375 (.0003)
1 MB	1.21	0.213 (.0337)

applications for Akenti access control are ones where there is something important to protect and the granularity of the access checking is fairly large, for example, a large document to be fetched, or a substantial process is to be started on the resource machine.

Another case where the Akenti overhead is not too severe is accessing a Web document that requires the parallel fetching of many secure components. For example, a document where all its parts are in the same protected tree. In this case, the browser and the server fetch in parallel, and since Akenti has no trouble working in parallel and sharing the same cache, the net result of such a clustered fetch is not too much worse than the secure fetching of one document.

5. RELATED WORK

5.1 Existing Systems

KeyNote [Blaze et al. 1999] is a trust management system, which provides a simple language for describing and implementing security policies, trust relationships, and credentials. Like Akenti, it provides a practical tool for authorization in a distributed environment. It differs from Akenti by using authorization delegation instead of resource access policy. KeyNote defines a *principal* as any convenient string that may include a cryptographic public key. Authorization policy is contained in *assertions* that consist of a sequence of fields. Each field is represented by a keyword and value. A *credential* asserts some attribute about a principal and is signed by a trusted authority. Both assertions and credentials are represented by the same keyword policy language. Akenti and KeyNote both provide a function call API for compliance checking for a resource gatekeeper (PEP) to call when making an access decision. Both systems return a list of trusted actions. KeyNote is less tied to one form of authentication than Akenti, since a KeyNote principal may be represented by a cryptographic key, or it may just be an opaque string. They deliberately did not require X.509 certificates in order to separate the issues of secure naming and authorization. While this removes the need for maintaining a PKI, it means that the principals named in the authorization policy may be opaque making it harder for a stakeholder to read and evaluate the policy of a resource.

The mechanisms for creating and storing policy assertions and storing and marshaling certificates are left up to the installer of a KeyNote system. In contrast, one of the emphases of the Akenti system is to support remote creation and storage of policy certificates. Akenti thus provides several tools to help in their creation and signing, while the policy engine supports gathering certificates from file systems, LDAP servers, or Web servers.

Other systems rely on the user being able to edit policy files on the resource gateway machine that does not meet our goal of accommodating distributed stakeholders.

In our original implementation of Akenti, we chose a simple keyword language for our certificates similar to that used by KeyNote. Eventually, expressing the constraints and trust relationships for all the attributes became increasingly awkward, with too much information being implicit in the ordering of fields or in relationships between fields. For our second implementation we switched to XML for greater flexibility and more precise definition of the semantics. We were also encouraged by the availability of XML parsing tools in a variety of languages and have made use of the Xerces parsers from the Apache XML project [APACHE 2002c].

Akenti is the only system of those surveyed that uses a pure pull model. The main motivation for this decision was to require no changes to the existing client server interfaces. Most other distributed authorization models require the requestor to present some credentials, in some cases to simplify the implementation of the PDP, but in other cases to give the user more control. The most serious feature lacking in Akenti is the ability of a user to limit his access during a specific session. This lack is most noticeable in role-based access control systems where the user would like to specify the role to use when making an access.

Several recent authorization services implement a hybrid model of authorization. In this model, the requestor contacts an authorization server before accessing the resource to get a proxy credential. The proxy credential may be handed back to a trusted authorization server or may contain the authorization assertions. Internet2's Shibboleth, Microsoft's Passport, and Liberty Alliance's open standards for federated network identity are examples of the first style; the Globus project's Community Authorization Service (CAS) and the European Data Grid's VOMS server are examples of the second.

Shibboleth [Erdos and Cantor 2002] is a cross-institutional authentication and authorization service for access control to Web-accessed resources. It is being specified by the Internet2 middleware architecture committee. It has many of the standard goals of distributed authorization with one additional twist. It wants to be able to grant access to a user who can still maintain anonymity at the resource site. The major motivation for this goal is access to library materials by academics. Shibboleth's authorization model entails a user making a request to a Web server and providing an identity handle back to his home institution. The Web server then asks that institution for attributes about the user. It then checks those attributes against its local policy to allow or deny access. The user need only to authenticate his host site and may use whatever type of credentials that site recognizes. One difference between this trust model and that used by Akenti is that in Akenti, the resource provider can separately specify authentication authorities and attribute authorities. Whereas in Shibboleth, the subject's home institution must provide both the authentication information about a user and all his attributes.

Microsoft's Passport and Liberty Alliance's federated identity servers [Hodges and Wayson 2003] use a similar model. A user authenticates once

with an identity server, and then relying sites contact that server to get authentication or attribute information about the user.

The Community Authorization Service (CAS) [Perlman et al. 2002] is a new authorization service being developed by the Globus project [Foster and Kesselman 1999] for Grid environments. Their authorization model allows a resource site to grant a community access to resources and the authorization server for that community to grant access to the community members. This is implemented by having the user contact the CAS server to get a delegated proxy certificate [Tuecke et al. 2002], which includes a rights restriction extension that limits what resources can be accessed. The resource gatekeeper must interpret the restricted rights extension and verify that the community has such rights to the resource. Since the delegated proxy is a short-lived X.509 certificate it can be passed between the user and the resource gateway as part of the GSI/SSL connection. There is no additional information that needs to be conveyed, as is the case when a user needs to pass attribute certificates to the gatekeeper. CAS differs from Akenti in that the examination of policy and granting of rights is done before the gatekeeper is contacted. This means the user must ask for all the rights she will need in advance of referencing the resource. In Akenti, all the gathering and checking of policy is done after the call to the gatekeeper to perform a certain action. Akenti does cache the rights that the user was granted, to deal with the common case of several calls in rapid succession for resources in the same realm.

Policy about resources is stored and managed by the CAS servers and so far mainly consists of lists of objects and allowed rights. This information is included in the rights restriction extension of the delegated proxy. The intent of the CAS project is to extend the policy language as the need arises. The CAS administrator is responsible for adding each community member to the appropriate groups. The CAS administrator may also delegate administration of subsets of the objects to additional people. In contrast, in Akenti, a new user would need to contact the stakeholder for the resource to be added to the policy files.

VOMS is another ad hoc solution to authentication in a GSI-enabled Grid. [Alfieri et al. 2003]. It is similar to the CAS model, but the VOMS server is run by a virtual organization and supplies authorization information about its own members. This service is implanted and used within the European Data Grid.

5.2 Proposed Standards

While there has been a great deal of work in formulating standards for authorization protocols or data structures, no single standard has emerged. There is an IETF proposed attribute certificate profile [Farrell and Housley 2001] to associate attributes with an X.509 identity. While the contents and purpose of this certificate are basically the same as an Akenti attribute certificate, we chose not to use it in our implementation because it is difficult for users and applications to deal with ASN.1 structures. A major goal of Akenti was to make the policy as easy to understand as possible, so using a consistent ASCII format to represent all our policy certificates was preferable to using a proposed ASN.1

standard for attribute certificates in addition to our own XML format for policy and use-condition certificates.

A recent XML standard specification for security assertions named security assertion markup language (SAML) [Hallam-Baker and Maler 2002] has been published by the OASIS consortium [OASIS 2002]. This standard defines both protocols and assertion structures in XML. Assertions come in three types *authentication*: the specified subject was authenticated by a particular means at a particular time; *attribute*: the specified subject is associated with the supplied attributes; *authorization decision*: a request to allow the specified subject to access the specified resource has been granted or denied. The SAML authorization request message provides the authentication assertion for the subject along with additional *evidence* that can include other assertions about the subject. This request is more general than Akenti in two respects: it handles multiple authentication methods, and it allows the requestor to push attributes to the PEP.

The attribute assertion/certificate has the same purpose as the PKIX attribute certificate and the Akenti attribute certificate: namely, a subject name, an associated attribute-value pair and the authority that attests to this and could be used in place of an Akenti attribute certificate.

The capability certificate returned by the Akenti server differs from the authorization decision assertion in that it does not contain the reasons (evidence) of why it made the decision. Both may contain unresolved conditions on the actions, so that the PEP can do further checks. The SAML assertion schema specifies a few conditions such as audience restriction, but the conditions element can be extended which might enable it to cover the types of runtime conditions that Akenti capability certificates include.

The SAML standards are focused on letting various peers report security decisions. The SAML authorization assertion could probably replace the Akenti capability certificate, and adapting the SAML authorization request message would considerably generalize the Akenti authorization model. However, the focus of Akenti is on defining, gathering, and interpreting policy (use-condition) statements about the resource. SAML does not address the expression of access policy.

eXtensible access control markup language (XACML) is another recently proposed XML language for an extensible access control policy [Godik and Moses 2003]. It is complementary to SAML as it focuses more on expressing the resource-centered access policy than on user credentials or authorization assertions. It does overlap SAML a bit, in its specification of authorization requests and responses. XACML has a rich policy information model consisting of *targets* (subject, resource, action), which are used in *rules* (target, conditions, (allow/deny)) which in turn are combined into *policies* that include user obligations to be enacted when accessing the resource. The XACML policy language is more complex than the Akenti certificate schema and could be used to express Akenti-style access constraints. However, there is an underlying assumption that all policy about a resource domain is managed by a trusted policy authority point and can be accessed securely by the PDP. Subject attributes are provided by a trusted policy information point. Thus, the XACML policy does not contain

any explicit trust relationships. In contrast, Akenti assumes no trusted servers, other than Akenti itself, and puts all the trust relationships explicitly into the policy statements.

Like the SAML request, the XACML authorization request includes *context* about the subject that can be used to pass attribute, role, or authentication information. The authorization responses return a value of permit, deny, or indeterminate if it needs to know more about the subject to make a decision. The response may also contain obligations, actions that the PEP must perform when the access is granted to the subject. XACML does not explicitly include the signing of any of its messages or policy files, but remarks that the XML digital signatures may be used.

Both of these XML languages have recently appeared in the OASIS standards body. SAML was accepted as an OASIS standard in Nov. 2002, and XACML was accepted as a standard in Feb. 2003. Whether the next version of Akenti should try to adopt the SAML authorization request and assertion, and the XACML policy language will depend on how widely these standards are used. In the case of XACML policy, it will depend on whether rules and conditions can be extended to include trust information.

6. CONCLUSIONS

Akenti is an authorization service (PDP) that uses authenticated X.509 certificates to establish identity and distributed digitally signed authorization policy certificates to make access decisions about distributed resources. It supports authorization decisions based on policy that it gathers from many sites. It returns an authorization decision as a signed capability certificate that can be used directly by a PEP to grant access or could be used by the subject of the certificate as a rights-granting authorization assertion. It supports Globus proxy identity certificates, and could easily be extended to handle restricted delegation credentials. We have implemented an Apache Web server module that allows the same authorization policy to be used to control access to Web-accessed resources as well as resources accessed by other remote methods.

Akenti differs from most of the other authorization services that we have surveyed in its emphasis on using easily read policy statements that are independently created and signed by multiple stakeholders. This policy can be stored on the resource host or local to the stakeholder and is gathered and evaluated by the trusted authorization server (PDP) at the time of resource access. The Akenti distribution also includes several tools for displaying the combined authorization policy for a given resource and for tracking the steps in a user's authorization or rejection.

Akenti was developed for use in distributed environments that rely on X.509 certificates and TLS to establish authenticated secure connections between the users and the resources. Hence, it was natural to rely on X.509 certificates for identity and to implement a pull authorization model. With the advent of the Web services model for distributed computing environments, the underlying security mechanisms are changing. TLS will be replaced with secure connections being made at the SOAP message level. The suggested protocols

for secure connections support a number of different authentication methods. With the communication protocol consisting of XML messages, it is much easier to extend the security protocols to push attribute or authorization information to a PEP. It is anticipated that there will be a standard interface for a Grid authorization service that will standardize the authorization request and response messages. The request message will contain additional assertions which can include roles as well as a generalized authentication token. As Akenti evolves to fit into the Grid services environment, it will need to address the issue of multiple authentication tokens and additional attribute assertions. Akenti should be able to conform to new authorization interfaces while keeping intact its fundamental goal of accommodating access policy statements that are independently created by stakeholders from unrelated domains.

The Akenti authorization has been used as part of the diesel combustion collaboratory [Pancerella et al. 1999] to control access to Web-based documents and remote execution and is now being integrated with the Globus job manager to control access to legacy applications in the National Fusion Grid [Keahey et al. 2001].

The code is freely available as C++ source code, or Linux and Solaris executables (http://www-itg.lbl.gov/Akenti).

ACKNOWLEDGMENTS

The original idea for Akenti came from William Johnston. Case Larsen did a large part of the original implementation. Wille Chin, Maria Kulick, Guillaume Farret, and Xiang Sun have also contributed to the current implementation.

REFERENCES

AGARWAL, D. A., SACHS, S. R., AND JOHNSTON, W. E. 1998. The reality of collaboratories. *Computer Physics Communications, 110*, 134–141.

AKENTI.XSD. 2003. Akenti certificate schema. http://www-itg.lbl.gov/Akenti/docs/AkentiCertificate.xsd.

ALFIERI, R., CECCHINI, R., CIASCHINI, V., DELL'AGNELLO, L., FROHNER, A., GIANOLI, A., LORENTEY, K., AND SPATARO, F. 2003. VOMS, an authorization system for virtual organizations Presented at the *1st European Across Grids Conference*, Santiago de Compostela, February 13–14, 2003.

APACHE. 2002a. Apache software foundation. http://www.apache.org.

APACHE. 2002b. Apache module registry. http://modules.apache.org/.

APACHE. 2002c. Apache XML project. http://xml.apache.org/.

ARSENAULT, A. AND TURNER, S. 2002. Internet X.509 public key infrastructure: roadmap internet draft, draft-ietf-pkix-roadmap-09.txt July 2002. http://www.ietf.org/internet-drafts/draft-ietf-pkix-roadmap-09.txt.

BELANI, E., VAHDAT, A., ANDERSON, T., AND DAHLIN, M. 1998. The CRISIS wide area security architecture. In *Proceedings of the USENIX Security Symposium*, San Antonio, Texas, January 1998.

BLAZE, M., FEIGENBAUM, J., IOANNIDIS, J., AND KEROMYTIS, A. 1999. The KeyNote trust management system, version 2. IETF RFC-2704. http://www.crypto.com/papers/rfc2704.txt.

DIERKS, T. AND ALLEN, C. 1999. The TLS protocol, version 1. IETF RFC 2246. http://www.ietf.org/rfc/rfc2246.txt.

DISCOM^2. ASCI DisCom^2. http://www.llnl.gov/asci/discom/.

ELLISON, C. 1999. SPKI requirements. IETF RFC 2692. http:// www.ietf.org/rfc/rfc2692.txt.

ERDOS, M. AND CANTOR, S. 2002. Shibboleth-architecture draft, v05. http://middleware.internet2.edu/shibboleth/docs/draft-internet2-shibboleth-arch-05.pdf

FARRELL, S. AND HOUSLEY, R. 2001. An Internet attribute certificate profile for authorization. draft-ietf-pkix-ac509prof-09.txt June 2001. http://www.ietf.org/internet-drafts/draft-ietf-pkix-ac509prof-09.txt.

FOSTER, I. AND KESSELMAN, C. (eds.). 1999. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Mateo, CA.

FOSTER, I., KESSELMAN, C., AND TUECKE, S. 2001. The anatomy of the Grid: Enabling scalable virtual organizations. *International J. Supercomputer Applications 15*, 3. http://www.globus.org/.

FRANKS, J., HALLAM-BAKER, P., HOSTETLER, J., LAWRENCE, S., LEACH, P., LUOTONEN, A., AND STEWART, L. 1999. HTTP authentication: Basic and digest access authentication. IETF RFC2617. http://www.ietf.org/rfc/rfc2617.txt

GODIK, S. AND MOSES, T. 2003. eXtensible access control markup language (XACML), version 1.0, oasis-xamcl-1.0.pdf. http://www.oasis-open.org/committees/xacml/repository/oasis-xamcl-1.0.pdf.

GOLAND, Y., WHITEHEAD, E., FAIZI, A., CARTER, S., AND JENSEN, D. 1999. HTTP extensions for distributed authoring—WEBDAV. IETF RFC2518. http://www.ietf.org/rfc/rfc2518.txt.

HALLAM-BAKER, P. AND MALER, E. (eds.). 2002. Assertions and protocol for the OASIS security assertion markup language (SAML), draft-ssct-core-31. http://www.oasis-open.org/committees/security/docs/ draft-ssct-core-31.pdf.

HODGES, J. AND WAYSON, T. 2003. Liberty architecture overview. http://www.projectliberty. org.

HOUSLEY, R., POLK, W., FORD, W., AND SOLO, D. 2001. Internet X.509 public key infrastructure certificate and CRL profile, draft-ietf-pkix-new-part1-12.txt. http://www.ietf.org/internet-drafts/draft-ietf-pkix-new-part1-12.txt.

IPG. 2002. NASA's information power grid. http://www.ipg.nasa.gov/.

KEAHEY, K., FREDIAN, T., PENG, Q., SCHISSEL, D. P., THOMPSON, M., FOSTER, I., GREENWALD, M., AND MCCUNE, D. 2001. Computational grids in action: The National Fusion Collaboratory. *Future Generation Computer System*. http://www.fusiongrid.org.

KOHL, J. AND NEUMAN, B. C. 1993. The Kerberos network authentication service (version 5). Internet request for comments RFC-1510.

LAUNCH. PAD. 2002. Portal to the IPG. http://www.ipg.nasa.gov/launchpad/servlet/launchpad.

LAMPSON, B., ABADI, M., BURROWS, M., AND WOBBER, E. 1992. Authentication in distributed systems: Theory and practice. *ACM Trans. Computer Systems 10*, 4 (Nov.), 265–310.

LEVY, H. M. 1984. *Capability-Based Computer Systems*. Digital Press, Bedford, MA. On-line at http://www.cs.washington.edu/homes/levy/capabook/.

MODSSL. 2002. http://www.modssl.org/.

MUDUMBAI, S. 2002. mod_akenti: Akenti access control module for Apache. http://www-itg.lbl.gov/Akenti/docs/mod_akenti.html.

NEUMAN, B. C. 1993. Proxy-based authorization and accounting for distributed systems. In *Proceedings of 13th International Conference on Distributed Computing Systems*, 283–291.

NEUMAN, B. C. AND TS'O, T. 1994. Kerberos: An authentication service for computer networks. *IEEE Communications*, *32*, 9, 33–38.

NFC. 2002. National Fusion Grid. http://www.fusiongrid.org/.

PANCERELLA, C., RAHN, L., AND YANG, C. 1999. The diesel combustion collaboratory: Combustion researchers collaborating over the Internet. In *Proceedings of ACM/IEEE SC99 Conference*, November 13–19, 1999, Portland, Oregon, USA. http://www-collab.ca.sandia.gov/dcc/.

PEARLMAN, L., WELCH, V., FOSTER, I., KESSELMAN, C., AND TUECKE, S. 2002. A community authorization service for group collaboration. In *Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*. http://www.globus.org/research/papers.html#CAS-2002.

PPDG. 2002. Particle Physics Data Grid. http://www.ppdg.net/.

OASIS. 2002. www.oasis-open.org.

RYUTOV, T., GHEORGHUI, G., AND NEUMAN, B. C. 1999. An authorization framework for metacomputing applications. *Cluster Computing. 2*, 2, 165–175.

RYUTOV, T. AND NEUMAN, B. C. 2000. Access control framework for distributed applications, IETF draft work-in-progress draft-ietf-cat-acc-cntrl-frmw-05.txt.

THAU, R. 2002. Apache API notes. http://modules.apache.org/doc/API.html.

THOMPSON, M., JOHNSTON, W., MUDUMBAI, S., HOO, G., JACKSON, K., AND ESSIARI, A. 1999. Certificate-based access control for widely distributed resources. In *Proceedings of the 8th Usenix Security Symposium,* Aug. 1999.

TUECKE, S., ENGERT, D., FOSTER, I., WELCH, V., THOMPSON, M., PEARLMAN, L., AND KESSELMAN, C. 2002. Internet X.509 public key infrastructure proxy certificate profile. IETF draft. http://ww.ietf.org/internet-drafts/draft-ietf-pkix-proxy-03.txt.

WAINWRIGHT, P. 2001. *Professional apache*, Wrox 2001. http://www.apache.org/.

WOO, T. Y. C. AND LAM, S. S. 1993. A Framework for distributed authorization. In *Proceedings of the ACM conference on Computer and Communications Security*, Fairfax, VA.

Received October 2002; revised February 2003, March 2003, June 2003; accepted August 2003