G E N I

Global Environment for Network Innovations

RBAC Requirements for ProtoGENI Spiral 1 Draft 0.5

Document ID: GENI-RBAC-REQ-0.5

January 15th, 2010

Prepared by:
Jay Jacobs, Alefiya Hussain, and Stephen Schwab
SPARTA, Inc.

Table of Contents

1. Document Scope

This document is entitled RBAC Requirements for ProtoGENI. It is a draft, intended to be a living document more in the spirit of an IETF Internet Draft rather than a Request For Comments (RFC) document that defines a protocol or standard. Over the course of development and prototyping spirals, this document will track and define the requirements for the use of role-based access control, as embodied by SPARTA's Attribute Based Access Control (ABAC) technology, primarily in the context of the control framework and slice-based facility architecture. As a reader or developer, your comments, criticisms and suggestions are welcome and essential to progress.

1.1 Purpose

The purpose of this document is to define the requirements for Attributed-Based Access Control (ABAC) extensions that allow the distinct security mechanisms of the various control frameworks to share security information within a single control framework, as well as with each other, starting with ProtoGENI and proceeding to ORBIT and ORCA in future years according to their integration readiness. The extensions will support trust management functions, including identity definitions and authentication mechanisms, and distributed authorization and access control mechanisms.

1.2 Related Documents

Some of the material in this document is drawn from the following documents listed below.

Document ID	Document Title and Issue Date
GENI-SEC-ARCH-0.55	"GENI Security Architecture", July 31, 2009
RFC 3280	Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, April 2002 http://www.ietf.org/rfc/rfc3280.txt
RFC 3281	An Internet Attribute Certificate Profile for Authorization, April 2002 http://www.ietf.org/rfc/rfc3281.txt
N/A	DETER Federation Daemon (fedd) http://fedd.isi.deterlab.net/
N/A	Access Control for Federation of Emulab-based Network Testbeds, Ted Faber and John Wroclawski, In Proceedings of the CyberSecurity Experimentation and Test (CSET) Workshop, San Jose, (July 2008) http://www.usenix.org/events/cset08/tech/full_papers/faber/faber.pdf
N/A	A DETER Federation Architecture, Ted Faber, John Wroclawski, Kevin Lahey, Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test, Boston, MA, (August 2007). http://www.usenix.org/events/deter07/tech/full_papers/faber/faber.pdf

WJ03a	Automated Trust Negotiation Technology with Attribute-based Access Control, W. Winsborough and J. Jacobs, In Proceedings of the DARPA Information Survivability Conference and Exposition, 2003, Vol. 2 pp 60-62, April 22-24, 2003.
GROUPS	Project Groups http://users.emulab.net/trac/emulab/wiki/Groups
LMW02	Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposiumon Security and Privacy*. IEEE Computer Society Press, May 2002.

2. Access Control

In a computing environment, trust management can answer questions regarding whether a process or application can perform an operation in a given situation. Before the introduction of Role-Based Access Control (RBAC), users, resources, and permissions needed to be enumerated explicitly, often in a cumbersome and error-prone manner. While early RBAC implementations avoid the need for explicit permissions to be assigned, strict hierarchical structures of resources, operations, and users were required. Designed specifically for heterogeneous, distributed computing environment, Attribute-based Access Control (ABAC) extends RBAC with the following features: decentralized attributes, delegation of attribute authority, inference of attributes, and attribute delegation of attribute authority. ABAC provides policy for sensitivity of credentials and allows organizations to maintain their own autonomy while still collaborating efficiently. Furthermore, ABAC provides an auditable, automated trust negotiation, where that capability is required.

2.1 Identity

Identity is defined as who someone or what something is, for example, the name by which something is known. Traditionally, identity requires identifiers—strings or tokens that are unique within a given domain, (that is globally or locally within a specific network, directory, application). Identifiers are the key used by the parties to an identification relationship to agree on the entity being represented. Identifiers may be classified as resolvable or non-resolvable. Resolvable identifiers, such as a domain name or e-mail address, may be referenced into the entity they represent, or some current state data providing relevant attributes of that entity. Non-resolvable identifiers, such as a person's real-world name, or a subject or topic name, can be compared for equivalence but are not otherwise machine-understandable.

Identity does *not* depend on identifiers, although identifiers depend on identity, and can be quite useful. One entity may have multiple identifiers, and an identifier may at different times (or in different scopes) be bound to different identities.

We also need to clarify the relationship between "entities" and "identities", beyond their syntactic similarity. The relationship between entities and identities can be thought of like the relationship between constants and variables in a program, both of which can be represented by identifiers, but named constants can be compared for more than equivalence.

ABAC policies should be crafted using entities, where an entity may have multiple credentials (from different issuers and/or using different authentication mechanisms). In a federated environment such as GENI, an identity could be a union of a principal’s, information stored across multiple distinct identity management systems. The databases could be joined together by the use of a common token. A principal's authentication process will thus occur across multiple networks or even across several organizations.

Identity certificates have standardized on X.509 certificates [RFC 3280], which are also used to facilitate secure communications. Private keys are needed for credential issuers. ABAC does not care about particular encryption mechanisms as long as the JVM (JCE or BouncyCastle) support them. Additional work will be required to enforce cryptographic policies. Acceptable cryptographic algorithms should be agreed to out-of-band at first to avoid interoperability issues, but each enclave needs control across an international consortium where different cryptographic or data privacy regulations might apply. ABAC relies on self-signed certificates and does not require a web of trust. Identity certificates may be self-signed and contain the cryptographic material needed to establish a secure network connection following the authentication process.

2.2 Authentication

Authentication verifies the identity of an entity in GENI. It is a key aspect of trust-based identity attribution, providing a codified assurance of the identity of one entity to another. Traditionally, authentication and identification mechanisms rely on maintaining a centralized database of identities, making it difficult to authenticate users in different administrative domains across federated networks. Each federated network keeps track of its users in a users account database and hence granting access to resources across networks is challenging. Each control framework may have its own mechanism of authentication at the early spiral prototypes. ABAC provides a mechanism for dynamic discovery of credentials which needs to be adapted for each control framework. This distributes the burden of user credential maintenance. In practice, remote and local credentials may be cached for performance, which needs to be balanced against the sensitivity of the operations and resources. Caching time quanta should take revocation time quanta into account.

2.3 Authorization

Authorization and availability are two distinct questions. Am I allowed to create a slice? Yes. Can I create a slice? If I'm allowed to create it and the available resources are free, then yes. Using attributes allows for policy to adapt quickly when large organizational changes take place (e.g. project termination, experiment launch, etc.). Cached attributes can also be useful in ad hoc networks, where power constraints and network bandwidth need to be conserved.

Authorization in ABAC is performed through a trust negotiation between two parties. The parties may not have a one-to-one mapping; for example, a subject may be traced back to an experiment entity rather than a specific principal entity. Credentials in ABAC are implemented as attribute certificates, which are self-signed by using the same issuer and holder [RFC 3281]. ABAC does not assume hierarchical organization identity structures (the underlying toolkits can though).

ABAC policy is constructed from sets of credentials which contain logical assertions using delegation logic. Credentials allow separation of identity from attributes. Identities change infrequently over time compared to attributes. Examples of identity changes would include name changes or security breaches, which occur infrequently but need to be processed in a timely manner. Rekeying identity should not require regeneration of all attributes except the leaf nodes in the credential inference chain. Attributes can be expected to change over varying periods of time. For instance, a student graduating from a program at Utah may advance from Utah.student attribute to a Utah.alum attribute. Identity certificate changes under normal circumstances should be infrequent while attributes changes can be more frequent. The time quantum for a Utah.student attribute might be a semester, while Utah.alum would have a much larger time quantum. Some attributes will exist for an experiment, a project's duration, or a for principal's tenure in a role such as project leader, group leader, or user.

2.4 ABAC Credentials

ABAC uses the RT language [LMW02] for specifying credentials and employs delegation logic which is used to implement role-based access control. Delegation logic is important for use in federated experiment policies, where testbeds can delegate and translate attributes among hierarchical organizations. RT has four types of policy rules: (1) simple roles; (2) simple inclusion; (3) linking inclusion; and (4) intersection inclusion. Intersections allow policies which require multiples of types 1, 2, and 3. Credentials take the form of:

$A.r_1 \leftarrow B$

which means the attributed role $A.r_1$ includes the principal entity B.

Type 1 credentials are used to assign a principal (an experimenter, experiment, organization, etc.) to an attributed role. For example, the following RT statements assert that three testbeds are being federated:

GENI.aggregate $\leftarrow$ DETER

GENI.aggregate ← Emulab

GENI.aggregate ← Cobham

Type 2 credentials translate attribute roles across organizational domains where the included role is at least a subset as shown in this RT example:

Utah.researcher ← Emulab.researcher

For example, all Utah researchers are Emulab researchers although Utah may have other researchers not associated with Emulab. This statement also lets Utah (as the entire campus) delegate policy to Emulab testbed operations for membership and policy. Since this federated experiment occurs within defined hierarchical organizations, Type 3 credentials can be used by a federation organization, GENI, to define federation participants as follows:

GENI.researcher ← GENI.university.researcher

GENI.researcher ← GENI.company.researcher

GENI.university ← Utah

GENI.company ← Cobham

GENI asserts that both university and corporate researchers map to the role of GENI.researcher. Universities and companies may join and leave GENI but this is likely to be infrequent; however, each university (Utah) and company (Cobham) has the autonomy to define researchers using their own policies.

Cobham.researcher ← Alice

Emulab.researcher ← Robert

In this example, Utah only needs to know that someone (Alice) is a Cobham researcher or that someone is Utah researcher (Robert). These are credentials issued directly to Alice or Robert, respectively. Robert also needs to supply the credential defining Utah.researcher. RT delegation logic frees the Emulab and Cobham IT staff from maintaining each other's membership policy. This degree of autonomy in policy is similar to the autonomy of sliver resources in a slice. The previous set of RT credentials assign Alice and Robert attributes directly; however, Emulab could delegate approval of credentials to a Utah graduate officer as follows:

Emulab.researcher ← Utah.graduateOfficer.gradStudent

Utah.graduateOfficer ← James

Now, James, in the role of graduate officer can designate researchers (Ann) by issuing credentials rather than having the credentials coming from Utah.

James.gradStudent $\leftarrow$ Ann

Discovery of ABAC credentials is crucial to the success of a trust negotiation without caching. In a ProtoGENI environment, discovery should be facilitated by a set of distributed clearinghouses. Similar to public key certificates, discovered credentials should be considered freely available. If some disclosure sensitivity is required, ABAC has access control and acknowledgment policies, which are consulted during the automated trust negotiation.

2.5 Trust Negotiation

ABAC uses a directed, acyclic graph, called a trust target graph (TTG), to facilitate a trust negotiation. Each negotiator maintains a separate copy of the TTG, where negotiators take turns at processing the graph until success or failure occurs. Sets of RT credentials and TTGs can be thought of as directed graphs. We use the term forward search to describe an inference chain from a principal entity to an attributed role. Backwards searches are inference chains mapping from an attributed role to included principals. Both types of searches are needed for dynamic discovery; however, discovered credentials cannot not be sensitive to disclosure. For sensitivity, ABAC has two forms of local policy: access control (AC policy) and acknowledgement policy (Ack policy), which we describe in the following subsections. TTGs can be cached for reuse during subsequent negotiation as long as the underlying RT credentials are still valid.

2.5.1 Trust Targets

Access control negotiations start with a resource service provided TTG. Nodes in the TTG are called trust targets which are connected via directed edges described in [WJ03a]. The root node of a TTG is the primary trust target which typically would look like:

V: $A.r_1$ <--?—S

Where, V is the verifier, presumably of a service providing a resource which requires the attributed role $A.r_1$, and S is the subject entity requesting the resource. In this example, V is the initial negotiator responsible for initializing the TTG. The authenticated, opponent negotiator may be S or someone authorized to act on the behalf of S.

2.5.2 Access Control Policy

AC policies are associated with credentials as well as with resources. When Alice requests a resource, the access mediator transmits to Alice the AC policy of that resource. This is in effect a query asking Alice whether she holds credentials that satisfy the AC policy. In ABAC, this operation is performed when the verifier initializes the TTG and transmits it to the opponent. AC Policy is local policy for the service provider and does not need to be publicly available.

2.5.3 Acknowledgement Policy

Ack policy is local policy, whose goal is to protect a negotiator's sensitive credentials. Ack policies ensure that no one is able to learn through negotiation whether a negotiator satisfies an attribute without first satisfying an Ack policy. Using the earlier GENI examples, GENI.aggregate attributes could be required before any credentials issued by James are released in a negotiation.

3. ProtoGENI

This section describes the current access controls available in ProtoGENI and gives examples of policy usage for the control framework.

3.1 Current Access Controls

ProtoGENI uses standard X.509 identity certificates from Emulab in order to facilitate secure communications over SSL. We propose extending the ProtoGENI decision points to make ABAC web services calls and extending ABAC to support ProtoGENI credentials. The ProtoGENI control framework already supports SSL authentication and implements a comprehensive set of control framework interfaces based on the SFA. The following sections briefly cover the framework and how it can benefit from ABAC.

3.1.1 Control Framework

Narrowly interpreted for this project, the control framework includes the slice authority and component manager. Permission to use a resource does not guarantee availability. In production systems, ABAC needs to verify credentials transmitted during the automated trust negotiation (ATN). The authenticated requestor should either be the subject of the negotiation or a delegate such as an authority or manager acting on a principal's behalf.
The Component Manager abstraction is used for contributing resources to ProtoGENI. We propose the following roles for delegated access control: Infrastructure (clearing house required functions), User (slice, sliver, ticket), and Admin(slice, sliver, ticket).

Engineering issues:

- Is Resolve sensitive? If there are credentials retrieved, then yes. If this operation is strictly for identity (authentication) then our interpretation would be no, since this is similar to discovery.
- For a given experiment, who are the likely users (PI, experimenter) and what is a likely distinction?

- Ack policy is not immediately necessary, but as more users and projects are added to the ProtoGENI environment, some restrictions should be anticipated.
- Is it necessary to have a users and administrator role? Do users actually need to use slices, slivers, or tickets?

Table 1 - Component Manager Features

Function Name	Description	ABAC Feature
Resolve	Lookup an object (by URN) and return information about it.	Gets information about a principal. This is where AckPolicies need to be inserted.
DiscoverResources	Return information about available resources.	Possibly use this with the intersection of a forward search
CreateSliver	Add a sliver to an existing slice.	Sliver access control
GetTicket,UpdateTicket	Request a ticket from the Component Manager. A ticket is another form of a credential, which in this case, is an rspec that has been signed, indicating a promise to deliver the resources specified in the rspec when the ticket is redeemed.	Ticket access control
RedeemTicket	Redeem a ticket obtained via the `GetTicket`() or `UpdateSliver`(). Note that the ticket is effectively destroyed, and no longer usable or accessible, if the operation succeeds.	Ticket access control
UpdateSliver	Request a change of resources for an existing sliver.	Sliver access control
RenewSlice	Request a change of expiration time for an existing sliver.	Slice access control
ReleaseTicket	Release a ticket obtained via the `GetTicket`() or `UpdateSliver`().	Ticket access control
StartSliver	Request that resources associated with a sliver be *started*.	Sliver access control
StopSliver	Request that resources	Sliver access control

	associated with a sliver be *stopped*.	
RestartSliver	Request that resources associated with a sliver be *restarted*.	Sliver access control
DeleteSliver	Request that a sliver be shutdown, returning a ticket for whatever remains of the original promise of resources.	Ticket access control
DeleteSlice	Request that all resources associated with a slice be released, and the sliver and slice records deleted.	Slice access control
GetSliver	Request a sliver credential for any resources that are currently allocated to a slice. Put another way, this call allows a user to get a duplicate sliver credential, if they have a valid slice credential.	Sliver access control
BindToSlice	Bind to a slice so that the caller may manipulate the slice.	Slice access control
SliverStatus,WaitForStatus	Return a description of the status for all of the resources that have been allocated to the slice on the component.	Sliver access control
ListUsage	Return a list of all resources in use. This is used by the ClearingHouse to get a global sense of usage. Currently, only the ClearingHouse will be allowed to make this call, but it might perhaps be made available to other principles, e.g. GMOC	This is a sensitive operation and should require AckPolicy.
Shutdown	Shuts down a slice completely.	Slice access control
ListHistory	This operation is reserved for the Clearing House (which authenticates itself with the credential	This is a sensitive operation and should require AckPolicy.

	parameter).	
GetVersion	Return the revision number of this API.	N/A

The Slice Authority (SA) coordinates resource sharing and mediation. Table 2 - Slice Authority Functions, lists the functions of the SA and where they fit into the ABAC framework. The functions can almost completely be divided into identity/credential functions {GetCredential, Resolve, Register, Remove, DiscoverResources, GetKeys} and slice operations {BindToSlice, RenewSlice, Shutdown}.

Table 2 - Slice Authority Functions

Function Name	Description	ABAC usage
GetCredential	Request a credential from the Slice Authority.	Add credential to local cache
Resolve	Lookup a UUID and return information about it.	Acknowledge policy needed here?
Register	Register a principal object, returning a new credential to manipulate that object later.	Add credential to local cache
Remove	Remove a principal object from the Slice Authority.	Check for ABAC slice privilege
DiscoverResources	Discover available components*	Forward search on principal entity?
GetKeys	Get all the SSH keys for the caller.	Used for verifying a credential signature
BindToSlice	Bind a local user to a slice so that the user may manipulate the slice. This call can be invoked by any user with a valid slice credential.	Check for ABAC slice privilege.
RenewSlice	Request a change of expiration time for an existing slice.	Check for ABAC slice privilege
Shutdown	Perform an emergency shutdown on a slice, by asking the SA (for that slice) to do an emergency shutdown. Operationally, the request is forwarded to the ClearingHouse which knows the full set of Component Managers. The call returns once the ClearingHouse is notified; the ClearingHouse will process the request asynchronously.	Check for ABAC slice privilege
GetVersion	Versioning for compatibility	N/A

*Are all published components available to any authenticated user? Using a forward search (including discovery) would return a set of all authorized resources. This is likely a research topic.

The slice authority uses Emulab certificates for authentication. The key pair used for authentication should be used for the ABAC trust target's issuer and subject if they are equivalent. The SA needs to initiate access control requests for slice privileges and credential management.

3.1.2 Clearinghouse

The clearinghouse functions include the features of the keystore in the reference ABAC implementation. Dynamic discovery in ABAC facilitates retrieval of distributed credentials in order to guarantee the timeliness of a trust negotiation. ProtoGENI credentials and principal identity certificates will need to be retrieved. *Currently, ProtoGENI credentials can only support RT type 1 credentials*. We expect to reuse the discovery API call described briefly in Appendix A.

Ideally, access control should be performed on the {Register, Remove, List, PostCRL} set of principal modifying functions. Care should be used during policy creation to ensure that at least some principals can register and remove other principals. A complete description of all clearinghouse features can be found in table 3. The slice related Shutdown function should also be a privileged function.

Table 3 - Clearinghouse Functions

Function Name	Description	ABAC Usage
GetCredential	Request a credential for accessing certain protected parts of the clearinghouse API.	Access control point to determine whether this is allowed for the callee.
Register	Register a principal object, returning success of failure.	Dynamic addition of identity certificate.
Resolve	Lookup a URN and return information about the corresponding object.	ABAC will need this for dynamic discovery search path.
Remove	Remove a principal object from the clearinghouse database. Note that only users and slices may be deleted from the clearinghouse at this time.	Dynamic removal of identity certificate.
Shutdown	Perform an emergency shutdown on a slice.	Possible access control point to determine whether this is allowed for the callee.
ListComponents	Return a list of all component managers.	TBD. *Does this need to be a trusted operation?*
PostCRL	This is a ProtoGENI specific call that is used by the federants to post their	Must be consulted before verification is complete.

	Certificate Revocation Lists.	
List	Return a list of all objects of the specified type, which should be one of "Authorities", "Components", "Slices" or "Users".	Authorities and Users will be needed for verifying credential signatures.
GetVersion	The revision of this API supported by the clearing house.	N/A

3.2 ProtoGENI Attributes

This section describes policy examples specific to ProtoGENI and how they can be used in slices and slivers. Using the Emulab access control model as a starting point [GROUPS], ProtoGENI can define four privilege levels: project leader, group leader, local administrator, and user. Users have the least amount of privilege and they should only have enough privilege to invoke the services in a slice. Local root users can administer slices and use the services provided by a slice. Group leaders can designate other group leaders, local administrators, and users, etc. Project leaders can in turn create groups and group leaders. Using attributed roles, the potential sliver locations are defined below:

ProtoGENI.aggregate ← PrimoGENI

ProtoGENI.aggregate ← Emulab

ProtoGENI.aggregate ← Cobham

These aggregate credentials should only be created when an out of band agreement between ProtoGENI and the subjects have been completed. Next, the privileges might use RT type 3 credentials to delegate aggregation:

ProtoGENI.user←ProtoGENI.aggregate.user

ProtoGENI.localRoot← ProtoGENI. aggregate.localRoot

ProtoGENI.groupLeader← ProtoGENI. aggregate.groupLeader

ProtoGENI.projectLeader ←ProtoGENI.aggregate.projectLeader

Now Cobham, Emulab, and PrimoGENI are free to designate principals to fill roles. Note that within small organizations, group delegation may not be as crucial (e.g. researchers) while roles such as user may have multiple levels or types of delegation (e.g. university classes, training classes, etc.).

3.2.1 Control Framework Policies

The component manager and slice authority access control are concerned with sliver and slice usage privileges. The main distinction is between users who do not need administrative privileges and developer and managers who will be (de)allocating slices and slivers. Here is a sample policy for slices (slivers would have an analogous set):

ProtoGENI.sliceUser ←ProtoGeni.user

ProtoGENI.sliceUser ←ProtoGeni.sliceControl

ProtoGENI.sliceControl←ProtoGENI.localRoot

ProtoGENI.sliceControl←ProtoGENI.groupLeader

ProtoGENI.sliceControl←ProtoGENI.projectLeader

These slice policies would be agreed to by all the aggregate members and are public knowledge. The access control point would use a trust target similar to the following:

Cobham: ProtoGENI.sliceUser ←? Alice

In this case, Cobham is asking whether Alice is a ProtoGENI slice user. Since Alice has a researcher credential from Cobham, she must either supply a credential stating that Cobham researchers or Cobham may publish a credential with the clearinghouse similar to the following:

Cobham.user ←Cobham.researcher

Discovered credentials do not allow for sensitivity. If this policy information is confidential to Cobham, then Alice needs to hold the credential in order to verify the other negotiator meets Cobham's Ack policy.

3.2.2 Clearinghouse

The clearinghouse is a repository for publicly available information which can be discovered. The operations Register,Remove,PostCRL are used to for credential management, which needs access control. Using the previous examples in section 3.2.1, clearinghouse policy could be similar to the following:

ProtoGENI.clearinghouseAdmin ← ProtoGENI.localRoot

ProtoGENI.clearinghouseAdmin ← ProtoGENI.groupLeader

ProtoGENI.clearinghouseAdmin ← ProtoGENI.projectLeader

Policy decisions for the clearinghouse include can a user add a self-signed identity certificate and which issuers may add credentials. Credentials should be added by the issuer signing them. Identity credentials should be added by their principals or on their behalf.

4. Attribute Extension Requirements

The goal of this effort is to extend the ABAC reference implementation to support multiple, distinct control frameworks. This section currently describes the necessary features for ProtoGENI to fully support and use all ABAC features and proposes further tasks. Task sections are listed in order of importance.

4.1 Decision Points

The access control decision points should be extended to reuse existing ABAC web service function *access*. Aggregate and component management are the access control decision points which can rely on this operation.

The clearinghouse is essential for performing dynamic discovery of credentials. The decision points need to guard against adding unauthorized users or removing authorized users.

4.2 Credentials

ABAC needs to support for ProtoGENI identity certificates. Propose and document ProtoGENI credentials for all four types of RT credentials. Currently only type 1 credentials can be used.

Add attribute credentials generation to ABAC for existing ProtoGENI credentials. Modify ProtoGENI credential formats to support types 2, 3, and 4. Type 4 (intersection) credentials are not immediately needed.

4.3 Discovery

Credentials used in ABAC need to use the clearinghouse for looking up public keys to verify credentials. Indexing of credentials should be done on the following keys: issuer (for auditing), required role (for backward searching), and subject role (for forward searching). The clearinghouse can also provide URNs to resolve name spaces for principals and attributes.

4.4 Revocation

Identity certificate revocations are done through the clearing house. Credential revocations need to be added to the clearinghouse. ABAC uses a trust target graph, which can be cached. The duration of the caching should be coordinated with the revocation update quantum. Credentials should also have an expiry date which should limit their usage.

4.5 Policy Features

The reference ABAC implementation does not restrict cryptographic algorithms. We propose adding enforcement points within ABAC for cryptographic policies. For example, if multiple credentials for a principal exist a particular organization may trust one authentication mechanism or require a minimum cryptographic strength or elliptic curve cryptography may be more desirable in low power environments.

Policy tools for auditing within an organization and between organizations. Modifying ABAC policies is simpler than non-role based policies; however, audit tools for a production environment will be needed as policy for projects and experiments change over time. Migrating users and administrator during project and group creation are a particularly critical time.

Appendix A: ABAC Web Service (WS-ABAC)

The reference ABAC web services interface supplies the following functions:

- negotiate – internal function used between negotiators to pass messages
  - in: negotiation request (message)
  - out: negotiation response (message)

- createContext – negotiation context unique to set of opponents
  - contextInfo (name)
  - contextInfo (id)

- access – an access request in the form of a primary trust target
  - in: access request (trust target, negotiation context)
  - out: access response (result, provenance, trust target)

- credentialUpdate – dynamic method for adding credentials to a specific context
  - in: credential[]
  - out: result code[]

- discovery -- discovery service which allows a search of credentials
  - in: op {issuer, required, subject}, credential
  - out: credential[]