3

RBAC POLICIES IN XML FOR X.509 BASED PRIVILEGE MANAGEMENT

D.W.Chadwick, A. Otenko
University of Salford

Abstract: This paper describes a role based access control policy template for use by privilege management infrastructures where the roles are stored as X.509 Attribute Certificates in an LDAP directory. There is a brief description of the X.509 privilege management model, and how it can be used to implement RBAC. Policies that conform to the template are written in XML, and the template is specified as a DTD. (A future version will specify it as an XML schema). The policy is designed to be used by the PERMIS API, a Java specification for an Access Control Decision Function based on the ISO 10181 Access Control Framework and the Open Group's AZN API.

Key words: X.509, Attribute Certificates, RBAC, LDAP, Role Based Access Controls, Policy Based Access Controls, XML, DTD

1. RBAC

Role Based Access Controls have generated significant interest in the last decade, even spawning their own ACM workshop in the mid 90s [ACM]. The main entities in RBAC are the users, the roles and the permissions. A role can represent a job function, a qualification or expertise. A permission represents the right to access a target in a particular mode. Roles are assigned to users in a many to many relationship, and permissions are granted to roles, again in a many to many relationship. There is thus a level of indirection between a user and his access rights. A good description of RBAC can be found in [Sandhu]. Some of the benefits of RBAC include:

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35586-3_46

M. A. Ghonaimy et al. (eds.), *Security in the Information Society*

- Ease of understanding. Because a user's privileges are tied into the role or roles he is undertaking when making a particular access request, administrators are better able to think about the privileges needed to fulfil a role rather than think about the privileges needed by an individual to perform various tasks.
- Support for the principle of least privilege. Coupled with the above, each role only needs to be given the minimum number of privileges necessary for the tasks associated with the role to be carried out.
- Ease of management. People fulfil different roles at different times in their careers, and roles are continually being occupied by different personal. However RBAC allocates privileges to roles, and these privileges change relatively infrequently, since it is immaterial from an access control perspective who is occupying a role at any given time. Therefore the main administrative task is simply to allocate and de-allocate roles to individuals.
- Scalability. In the world of e-commerce there are typically far fewer roles than participants. Roles might include: cash buyer, seller, administrator, credit buyer etc, whereas the number of participants in each role can run into the thousands and in some cases even millions. Since RBAC allocates privileges to roles, it is much easier to administer the privileges to a limited number of roles, than in traditional discretionary access controls, where privileges are allocated to users. It is then a much simpler task to allocate roles to the large numbers of users.
- Separation of duties. This applies at two levels. Firstly, it is possible to constrain role allocation so that an individual cannot occupy mutually exclusive roles e.g. that of corporate buyer and seller. Secondly in RBAC administration, the person who determines a role's privileges can be different to the person who allocates roles to individuals.
- Privilege inheritance. Role hierarchies can be defined, e.g. Director > Manager > Employee, in which the superior roles inherit the privileges of the subordinate roles, as well as having their own additional privileges. Thus least privileges can be allocated to the junior roles, and these will automatically be inherited by the more senior roles.
- Delegation of duties. A role occupant may delegate his role to another individual, without needing to have permission to alter the privileges assigned to that individual. Furthermore, when role hierarchies are supported, a role holder can delegate just a subordinate role instead of the entire role.

2. X.509 BASED PRIVELEGE MANAGEMENT

Edition 4 of X.509 [X509], published by the ITU-T in 2001, is the first edition to fully standardise the certificates of a Privilege Management Infrastructure (PMI). Hitherto, earlier versions of X.509 have concentrated on standardising the certificates of Public Key Infrastructures (PKIs). This paper assumes the reader is already familiar with the general concepts of PKIs, and these will not be repeated here. Readers wishing to learn more about PKIs may consult books such as [Adams][Austin] or [Housley].

The X.509 PMI is to authorisation what its PKI is to authentication. Consequently there are many similar concepts in PKIs and PMIs. These are summarised in Table 1 below. Whilst public key certificates are used to maintain a strong binding between a user's name and his public key, an attribute certificate (AC) maintains a strong binding between a user's name and one or more privilege attributes. (In this respect a public key certificate can be seen to be a specialisation of a more general attribute certificate.) The entity that digitally signs a public key certificate is called a Certification Authority (CA), whilst the entity that signs an attribute certificate is called an Attribute Authority (AA). The root of trust of a PKI, is sometimes called the root CA or trust anchor whilst the root of trust of the PMI is called the Source of Authority (SOA). CAs may have subordinate CAs which they trust, and to which they delegate the powers of authentication and certification. Similarly, SOAs may delegate their powers of authorisation to subordinate AAs. If a user needs to have his signing key revoked, a CA will issue a certificate revocation list (CRL). Similarly if a user needs to have his authorisation permissions revoked, an AA will issue an attribute certificate revocation list (ACRL). X.509 can be used to implement discretionary access controls, by storing users' access rights/permissions in their ACs, and by allowing users to allocate ACs to each other for the resources that they control.

Table 1. A Comparison of PKI and PMI Entities

Concept	PKI entity	PMI Entity
Certificate	Public Key Certificate	Attribute Certificate
Certificate Issuer	Certification Authority (CA)	Attribute Authority (AA)
Certificate Receiver	Subject	Holder
Certificate Binding	Subject's Name to Public Key	Holder's Name to one or more Privilege Attributes
Revocation	Certificate Revocation List (CRL)	Attribute Certificate Revocation List (ACRL)
Root of Trust	Root CA or Trust Anchor	Source of Authority (SOA)
Subordinate Authority	Subordinate CA	Attribute Authority

3. RBAC USING X.509 ATTRIBUTE CERTIFICATES

X.509 supports RBAC by defining role specification attribute certificates that hold the permissions granted to each role, and role assignment attribute certificates that assign various roles to the users. Each role and name component in X.500 [X501] and LDAP [LDAP] is an attribute type, attribute value pair. Thus roles and name components are easily interchangeable. In role specification ACs, the holder is the role, and the privilege attributes are permissions granted to the role. In role assignment ACs, the holder is the (name of the) user, and the privilege attributes are the roles assigned to the user. The user is identified by either his LDAP Distinguished Name [DN] or his public key certificate (issuer and serial number). Role assignment ACs may point to their corresponding role specification AC via the role specification certificate identifier extension.

Hierarchical RBAC allows role specifications to be more compact, since a superior role does not need to enumerate the privileges it has inherited from its subordinate roles. X.509 supports hierarchical RBAC by allowing a role specification attribute certificate to contain both roles and privileges in the embedded attributes, so that the specified role inherits the privileges of the embedded role(s).

Delegation is supported through the basic attribute constraints extension. This extension holds an integer that indicates the depth of delegation that may take place; zero indicating no delegation, one indicating one level of delegation, and missing indicating no restrictions on delegation.

4. POLICY BASED PRIVILEGE MANAGEMENT

In policy based RBAC, a policy is defined which states the rules for assigning roles to users, and permissions to roles. The policy can then be used to control the accesses to all the targets within the policy domain. We have specified an API in the Java language, the Permis API [Permis], that reads in the XML policy, parses it, and then uses it to control access to targets within the policy domain. The API caller, typically an application gateway, passes the authenticated name of the user, and this is used to retrieve the user's attribute certificates (ACs) from the configured LDAP directory. Each signed AC is checked against the policy, and non-conformant ACs are discarded. Valid roles are extracted from the remaining ACs. The API caller then passes the user's requested action on his chosen target, and again this is checked against the policy. The API returns either

granted or denied to the caller. In this way, a single policy can be used to control access to all the resources in a domain.

5. THE X.509 PMI RBAC POLICY

We have specified an RBAC policy specifically designed for use with an X.509 attribute certificate based PMI. The top level X.509 PMI RBAC Policy is composed of a number of sub-policies as shown in Figure 1 below. The domain of the PMI Policy is the union of all the domains of the sub-policies. Each policy is given a unique object identifier (OID) that globally unambiguously identifies it. This OID is passed to the Permis API by the caller, in order to guarantee that the correct policy will be used in all the subsequent access control decisions made by the API implementation. The policy OID may also optionally be stored in X.509 ACs, so as to limit the scope of the ACs if desired.

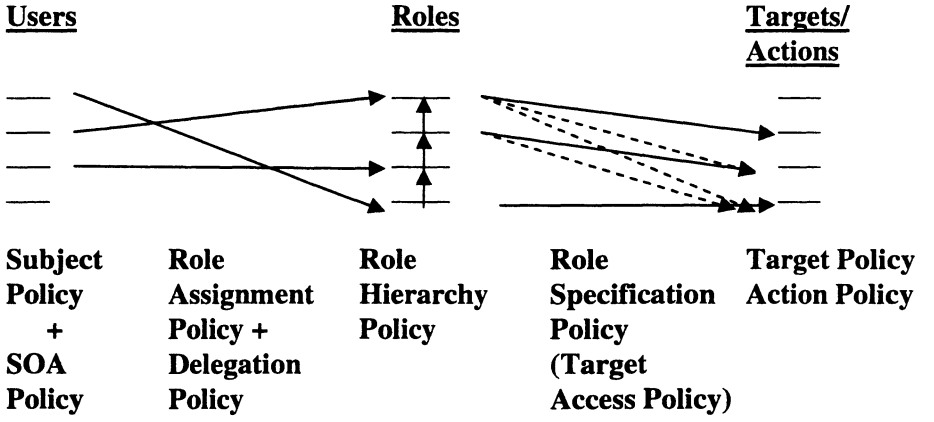

Figure 1. The X.509 PMI RBAC Policy and its Sub-Policies

5.1 Subject Policy

The Subject Policy specifies the domains of users who may be granted roles within the overall PMI policy. Each domain is specified as an LDAP subtree, using Include and Exclude statements, with optional layering. The Include statement specifies the LDAP DN of the root node of a subject domain. This subtree may be pared in two ways, by using Min and Max integers to specify which layers of the subtree to include, and by using Exclude statements to specify which subordinate subtrees to exclude from

the domain. Each excluded subtree may also be layered using Min and Max integers. The default for Min is zero, meaning the root of the subtree, and the default for Max is infinity, meaning the leaves of the subtree. Using a null LDAP DN in an Include statement specifies the domain of all users in the world.

As an example, a subject domain could comprise all users in an organisation, by using an Include LDAP DN of dc=my org, dc=com; or a particular unit could be excluded by adding an Exclude LDAP DN of ou=excluded unit, dc=my org, dc=com, or only all organisation units could be included by using Min of 1 and Max of 1. The example subject policy below, in XML, has two domains, one for UK companies and one for Salford City employees, excluding the marketing department.

```
<SubjectPolicy>
    <SubjectDomainSpec ID="UKCompanies">
        <Include LDAPDN="c=gb"/>
    </SubjectDomainSpec>
    <SubjectDomainSpec ID="Employees">
        <Include LDAPDN="dc=salford,dc=gov,dc=uk"/>
        <Exclude LDAPDN="ou=marketing,dc=salford,dc=gov,dc=uk"/>
    </SubjectDomainSpec>
</SubjectPolicy>
```

5.2 SOA Policy

The SOA Policy lists the LDAP DNs of the SOAs that are trusted to issue roles to the subjects specified in the subject policy. These DNs will match the root issuer names in published ACs. The first name in the list is the LDAP DN of the policy creator, and this name must always be present. Subsequent names refer to remote SOAs who are effectively being cross-certified by the policy creator. Every AC that is to be trusted by this policy must have been signed by one of the SOAs in this list, or by an AA rightfully delegated by one of these SOAs.

Many applications will only require the SOA list to comprise the single name of the policy creator, and all ACs will be issued by the policy creator or a delegated AA. However, we have already encountered a number of applications where multiple SOAs and externally allocated ACs are required. Those applications that do not support externally issued privileges, will usually incorporate the concept by requiring the policy creator (or his delegate) to re-issue these privileges to their subjects. But our SOA policy does not require this to happen.

Applications that we are building that benefit from the support of externally allocated privileges include electronic tendering and electronic prescribing. In electronic tendering, the organisation issuing the Request for Tenders might require that the tenderers be ISO 9000 certified or similar. ISO 9000 certificates are typically issued by a standards body (in the case of the UK this is the British Standards Institute (BSI)). Thus a policy can be specified that trusts BSI to issue ISO 9000 certificates to organisations. The certificates can then be electronically issued by BSI in the form of X.509 attribute certificates, and stored in their publicly accessible LDAP directory server, allowing the tendering application to download them. In electronic prescribing, GP (general practitioner) roles are allocated to doctors by the General Medical Council, pharmacist roles are allocated to qualified pharmacists by the Royal College of Pharmacists, and NoFee Patient roles are allocated by various government agencies to members of the public entitled to free prescriptions for one reason or another.

The example SOA policy below allows for two SOAs, the policy creator and BSI.

```
<SOAPolicy>
    <SOASpec ID="PolicyOwner" LDAPDN="cn=David Hunter,
                        ou=computing, dc=salford,dc=gov,dc=uk"/>
    <SOASpec ID="BSI" LDAPDN="o=bsi,c=gb"/>
</SOAPolicy>
```

5.3 Role Hierarchy Policy

The Role Hierarchy Policy defines the role hierarchies that are supported by this RBAC policy. Each role hierarchy (RoleSpec in the DTD) is a directed graph, rather than a simple tree, as the former supports multiple superior roles inheriting the privileges of a common subordinate role. For example, the Employee role may have superiors of Administrator and Project Leader, both of whom inherit the privileges given to an employee. The role hierarchy also supports multiple inheritance whereby a superior role inherits all the privileges of a set of subordinate roles. For example, Managing Director may have subordinate roles of General Manager, Company Secretary and Chief Technical Officer.

Each role is named using an Attribute Type, Attribute Value pair, for example AT=isoCertification, AV=ISO9000; AT=permisRole, AV=CTO; AT=healthProfessional, AV=GP. The attribute types will typically be the LDAP attribute type names (these are a single ASCII string usually starting with a lower case letter [LDAP]). However, the attribute types in X.509 ACs are identified using their globally unique object identifiers, so each role

hierarchy (RoleSpec) holds the mapping of the LDAP attribute type name to its OID.

Each role hierarchy is specified as a set of Superior-Subordinates attribute values. (Consequently it is not possible to define a role hierarchy containing different attribute types.) Each superior role can have multiple subordinate roles, and each subordinate role may also be a superior. This allows any arbitrary directed graph to be described (when a directed graph is described as a tree, some subordinate will occur multiple times in the tree beneath different superiors). Leaf subordinate roles are specified as superiors without any subordinates. A set of roles that do not form a hierarchy, but rather are equal roots of trees comprising single entries, is specified as a set of Superiors, as in the example below where ISO9000 and ISO17799 are not hierarchically related. Also in the example below, the permisRole of map reader is subordinate to the role of architect.

```
<RoleHierarchyPolicy>
  <RoleSpec Type="permisRole" OID="1.2.826.0.1.3344810.1.1.14">
      <SupRole Value="Architect">
          <SubRole Value="MapReader"/>
      <SupRole Value="MapReaders"/>
  </RoleSpec>
  <RoleSpec Type="isoCertification" OID="1.2.826.0.1.3344810.1.1.15">
      <SupRole Value="ISO9000"/>
      <SupRole Value="ISO17799"/>
  </RoleSpec>
</RoleHierarchyPolicy>
```

5.4 Delegation Policy

The Delegation Policy has been taken from the X.509 standard. This allows the depth of delegation to be specified as an integer. (In X.509, the integer is specified in the path length constraints field of the basic attribute constraints extension.) A depth of zero indicates that no delegation is allowed, a depth of one indicates that the entity assigned a role assignment attribute certificate may also assign the latter to other entities. If the depth is missing, infinite delegation is allowed. We have included the delegation policy as an integral part of the role assignment policy, since delegation is a function of each role assignment.

5.5 Role Assignment Policy

The Role Assignment Policy specifies which roles can be assigned to which subjects by which SOAs. For each role assignment, we also specify whether the assigned roles can be delegated or not (see above), and whether there are any time constraints on the assignment. Both subjects and SOAs are referenced using their IDs from their respective policy definitions. Roles may be specified as either their type only (implying all values of a particular type may be assigned), by a specific combination of type and value (meaning only a single role may be assigned), or by null (meaning any role of any type from the role hierarchy policy may be assigned).

Policy time constraints, which are optional, over-rule any validity times in the attribute certificates. Policy time constraints can be specified in either or both of two ways, absolute and relative.

Absolute time constraints comprise absolute start and end times, with each time being specified as a date/time integer string in ISO8601 format i.e. ccyy-mm-ddThh:mm:ss. The actual validity time of an AC is taken to be the intersection of the policy absolute validity time and the validity time field in the AC. If the policy start time is missing, the AC is valid from the beginning of its notBefore validity time. If the policy end time is missing, the AC is valid up until its notAfter validity time (or until revoked). For example, a policy absolute start time of 2002-02-01 with an AC notBefore validity time of 1 Jan 2002, would mean that the AC will not be accepted during the month of January 2002.

Relative time constraints specify the maximum Age of a certificate and the Maximum and Minimum life spans, relative to the evaluation time, that the AC must be valid for, in order for it to be accepted. These relative times are specified as date/time integer strings of the form +yy-mm-ddThh:mm:ss, meaning this amount of time from now. Trailing (zero) dates and times can be omitted i.e. to specify 2 months from now would be +00-02. Whilst Age goes back in time from now, both Maximum and Minimum go forward in time from now. For example, if the Age is specified as 02, then an AC that has a notBefore validity time of more than two years from the evaluation time will be discarded. If the Maximum time is specified as +00-02, then an AC that has a notAfter validity time more than two months from the evaluation time will be discarded. If Minimum time is +00-00-01, then an AC that has a notAfter validity time less than one day from the evaluation time will be discarded. If the Age, Maximum and Minimum times are missing, the AC does not have any policy time constraints imposed on it.

The example role assignment policy below allows three types of AC to be trusted. The first one is role assignment ACs for tender officers. This role can only be assigned to employees by the policy owner, and they are not

valid before 5pm on 21st September 2001 (the close of the electronic tendering process), and delegation of the role is not permitted. The second one is role assignment ACs for tenderers. The latter must be UK companies, and the policy owner must make the assignment. They are only valid up to 5pm on 21st September 2001 (the close of the electronic tendering process). Delegation of the role is not permitted. The final role assignment is for ISO9000 certification. The assignment is made by BSI, but only certificates for UK companies are valid. Furthermore, since companies have to be annually re-certified, certificates older than one year are not acceptable, neither are certificates valid for more than one year, or certificates that expire in less than one day from now. Delegation is not permitted.

```
<RoleAssignmentPolicy>
  <RoleAssignment>
      <SubjectDomain ID="Employees"/>
      <Role Type="permisRole" Value="TenderOfficer"/>
      <Delegate Depth="0"/>
      <SOA ID="PolicyOwner"/>
      <Validity>
        <Absolute Start="2001-09-21T17:00:00"/>
      </Validity>
  </RoleAssignment>
  <RoleAssignment>
      <SubjectDomain ID="UKCompanies"/>
      <Role Type="permisRole" Value="Tenderer"/>
      <Delegate Depth="0"/>
      <SOA ID="PolicyOwner"/>
      <Validity>
          <Absolute End="2001-09-21T17:00:00"/>
      </Validity>
  </RoleAssignment>
  <RoleAssignment>
      <SubjectDomain ID="UKCompanies"/>
      <Role Type="isoCertification" Value="ISO9000"/>
      <Delegate Depth="0"/>
      <SOA ID="BSI"/>
      <Validity>
          <Age Time="01"/>
          <Maximum Time="01"/>
          <Minimum Time="00-00-01"/>
      </Validity>
  </RoleAssignment>
```

</RoleAssignmentPolicy>

5.6 Target Policy

The Target Policy specifies the target domains covered by this policy. Target domains are specified as LDAP subtrees, using Include, Exclude, Max and Min statements as for subject domains. Include specifies the LDAP DN of the root node of a domain, and optional Exclude statements specify subordinate subtrees to be excluded from the domain. Min and Max specify which layers of the subtree are being targeted. Using a null LDAP DN in an Include statement specifies the domain of all targets in the world. At least one domain must be specified, even if it is the whole world. A domain may optionally be refined by specifying a set of object classes. Only targets with the full set of object classes are included in the domain. (A null set implies all targets in the domain are included.) The reason we allow target domains to be refined using object classes, but do not allow subject domains to be similarly refined, is that targets are trusted to provide their object classes to the Permis API, since the API is acting on their behalf. Subjects on the other hand, are trying to gain access to the targets via the Permis API, are not trusted to provide their own object class. However, a similar functionality can be introduced for subjects, by allocating object class bearing attribute certificates to them, and requiring these object classes (which can be treated as roles) to be present in the target access policy (see below).

The following example comprises two target domains, the first is all the printers in the Salford City domain, and the second is the tender store application at Salford.

```
<TargetPolicy>
    <TargetDomainSpec ID="Printers">
        <Include LDAPDN="dc=salford,dc=gov,dc=uk"/>
        <ObjectClass Name="Printers"/>
  </TargetDomainSpec>
  <TargetDomainSpec ID="TenderStore">
      <Include LDAPDN="cn=Tender Store,dc=salford,dc=gov,dc=uk"/>
  </TargetDomainSpec>
</TargetPolicy>
```

5.7 Action Policy

The Action Policy specifies the actions that are supported by this policy. An action is the smallest granularity of access to a target. Each action has a name, and zero or more arguments. The policy creator will need to refer to

each target's Reference Manual for the names of the actions/methods the target supports, plus the sequence of arguments that each method requires. The sequence of the arguments in the Action Policy is very important and must be the same as those passed by the access enforcement function (AEF) to the Permis API at run time.

The reason why we separately declare the actions is purely one of policy specification efficiency. Several targets may support the same action, so the latter only needs to be specified once in the Action Policy.

The following example lists the actions that can be applied to the Tender Store application. It supports submission, retrieval and deletion of tender documents, given the unique number of a particular tender.

```
<ActionPolicy>
        <Action Args="TenderNo" Name="Write" />
        <Action Args="TenderNo" Name="Read" />
        <Action Args="TenderNo" Name="Delete" />
</ActionPolicy>
```

5.8 Target Access Policy

The Target Access Policy comprises a set of target access clauses. Each target access clause grants an initiator with a specified set of roles permission to carry out the specified actions on the specified list of targets, but only if the conditions specified by the optional IF clause are true. Note that the policy implicitly operates the Deny All Unless Explicitly Granted rule, so that only those permissions in the target access clauses will ever be granted.

An initiator must possess all of the roles in a target access clause in order to gain the specific access/privilege. Other RBAC schemes typically assign actions (privileges) to each role. But we wanted to cater for the case where an initiator may be required to possess several ACs before any privilege is granted. Hence the ability to specify a set of roles in a target access clause, if required.

Each target in the target list can be a target instance (which must be within a previously specified target domain) or a reference to a previously specified target domain. Each target has its own list of granted actions. Actions are referred to by their action name. If no actions are specified then all actions that the target supports are granted.

The IF clause specifies the conditions which must be satisfied in order for the actions to be granted. It is an enhanced version of the privilege policy specified in clause D.2 of Annex D of X.509. A condition comprises:

- a comparison (logical) operator

- the LHS operand(variable), described by its source, name and type, and
- a series of one or more variables or constant values against which the LHS operand is to be compared.

The operator is chosen from the following set: PRESENT | EQ | GT | LT | LE | GE |Subordinate | Substrings | Subset | Superset | NonNullIntersection | ApproxEQ | and Operator, where Operator is an extensibility mechanism to allow policy setters to define new operators for their condition statements. The meaning of any new operator and the number of operands it operates on is application specific. The Permis API will support the calling of new Java objects that implement the new operators and their operands.

Two possible sources are currently specified for an operand/variable: either an argument from the action specified by the initiator, or the environment. In the former, the variable name is the action's argument name, and the variable type is the type of the argument. Examples might be: for a Read action specified by the initiator, the argument is the filename, and its type is string; for a tender submission action by an initiator, the argument is the tender number, and its type is integer. The environment represents the Contextual ADI in the ISO 10181-3 Access Control Framework. Environmental variables are application specific, and AEF implementers specify their names and types, and pass them to the Permis API at initialisation time. The policy creator will need to refer to the AEF Reference Manual for a complete list of environmental variables. Examples of environmental variables are the time of day and the number of previous accesses. We may add additional operand sources in the future if there is an identified need for them.

Constants comprise types and values. Typically the LHS variable is compared against a right hand constant using the specified operator, and the result is evaluated to true or false. The IF clause also supports complex boolean logic when individual conditions are combined through the inclusion of ANDs, ORs and NOTs.

The following example allows tender officers to remove tenders from the tender store but only between 9am and 5pm, Mon-Fri, June to October 2001. (Note. The time period is adapted from RFC 3060 [Policy] and may contain any or all of the following Start, End, MonthsOfYear, DaysOfMonth, DaysOfWeek, TimeOfDay and LocalOrUTC)

```
<TargetAccess>
    <RoleList>
      <Role Type="permisRole" Value="TenderOfficer"/>
    </RoleList>
    <TargetList>
```

```
        <Target Actions="Delete">
            <TargetDomain ID="TenderStore"/>
        </Target>
    </TargetList>
    <IF>
    <EQ>
      <Environment Parameter="TimeOfAccess" Type="Time"/>
      <Constant Type="TimePeriod" Value= "DaysOfWeek=0111110
      End=2001-10-00 LocalOrUTC=local Start=2001-06-00
      TimeOfDay=T090000/T170000"/>
    </EQ>
    </IF>
</TargetAccess>
```

6. XML DTD/SCHEMA

Version 7 of the PERMIS X.500 PMI RBAC Policy DTD has been published at http://www.xml.org, and is also available from our web site at http://sec.isi.salford.ac.uk/permis/Policy.dtd. The reason why we used a DTD rather than a schema, is that the XML tools that we were using e.g. IBM's Xeena and MS Internet Explorer, support the use of DTDs and not Schemas. As soon as Schema using tools become readily available we will produce the XML Schema corresponding to our published DTD.

7. CONCLUSIONS

We have described a comprehensive RBAC policy for use with X.509 attribute certificates. Policies are specified in XML, and the DTD has been published by xml.org. The X.509 RBAC policy defines the subject and target domains governed by the policy, the role hierarchies supported by the policy, which roles may be assigned to which subjects by which trusted SOAs, and which roles are needed to perform which actions on which targets under which conditions. We believe the policy is widely applicable to many different types of application, and we are already using it for electronic tendering, electronic prescribing, and several database access applications.

REFERENCES

[ACF] ITU-T Rec X.812 (1995) | ISO/IEC 10181-3:1996 "Security Frameworks for open systems: Access control framework"
[ACM] ACM Workshop on Role Based Access Control, 1996-2001. See http://portal.acm.org/portal.cfm for proceedings.
[Adams] Adams, C., Lloyd, S. (1999). "Understanding Public-Key Infrastructure: Concepts, Standards, and Deployment Considerations". Macmillan Technical Publishing, 1999
[Austin] Austin, T. "PKI, A Wiley Tech Brief", John Wiley and Son, ISBN: 0-471-35380-9, 2000
[DN] Wahl, M., Kille, S., Howes, T. "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", RFC2253, December 1997.
[Housley] Housley, R., Polk, T. "Planning for PKI: Best Practices Guide for Deploying Public Key Infrastructure". John Wiley and Son, ISBN: 0-471-39702-4, 2001
[LDAP] Wahl, M., Howes, T., Kille, S. "Lightweight Directory Access Protocol (v3)", RFC 2251, Dec. 1997
[Permis] The latest version of the Permis API can be downloaded from http://sec.isi.salford.ac.uk/permis
[Policy] B.Moore, E. Ellesson, J. Strassner, A. Westerinen. "Policy Core Information Model -- Version 1 Specification". RFC 3060, February 2001.
[Sandhu] Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E. "Role Based Access Control Models". IEEE Computer 29, 2 (Feb 1996), p38-43.
[X501] ISO/ITU-T Rec. X.501(1997) The Directory: Models
[X509] ISO/ITU-T Rec. X.509(2001) The Directory: Authentication Framework

ACKNOWLEDGEMENTS

This work has been 50% funded by the EC Information Society Initiative For Standardization (ISIS) programme, as part of the pan-European PERMIS project (see http://www.permis.org).