An X.509 Role-based Privilege Management Infrastructure

a report by
David W Chadwick

Professor of Information Systems Security, Information Systems Institute, University of Salford

David W Chadwick is a Professor of Information Systems Security at the University of Salford. He has published widely on the topics of X.500, public key infrastructures (PKIs) and X.509, and has been running an Entrust PKI for research purposes for five years. He is the Bristish Standards Institution (BSI) representative at X.509 standardisation meetings, and was the international editor of X.518 in 1993. He is currently the author of four Internet drafts concerning the use of PKIs and Lightweight Directory Access Protocol (LDAP). He has participated in many EC and UK security related research projects including: ICE-TEL, TrustHealth 2, ICE-CAR, GUIDeS, Intelligent Computation of Trust, and the Distributed Diabetic Dietician. Professor Chadwick's current projects include: the Privilege and Role Management Infrastructure Standards Validation (PERMIS) Project, the PKI Challenge, certificate retrieval from OpenLDAP and electronic prescriptions processing.

Introduction to X.509 (2001)

Edition four of X.509[1] – published by the International Telecommunication Union Telecommunication sector (ITU-T) in 2001 – is the first edition to standardise fully the certificates of a privilege management infrastructure (PMI). Earlier versions of X.509 have concentrated on standardising the certificates of a public key infrastructure (PKI).[2]

A PMI is to authorisation what a PKI is to authentication. Consequently, there are many similar concepts in PKIs and PMIs. These are summarised in *Table 1*. While public key certificates are used to maintain a strong binding between a user's name and his or her 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 AC. The entity that signs a public key certificate digitally is called a Certification Authority (CA) and the entity that signs an AC is called an Attribute Authority (AA). The root of trust of a PKI is sometimes called the root CA,[3] while the root of trust of the PMI is called the source of authority (SOA). CAs may have subordinate CAs that 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 or her signing key revoked, a CA will issue a certificate revocation list. Similarly, if a user needs to have authorisation permissions revoked, an AA will issue an attribute certificate revocation list (ACRL).

Implementing Authorisation Schemes with X.509

Various authorisation schemes have been devised in the past. The most popular and well-known model is the discretionary access control (DAC) scheme. In the DAC scheme, users are given access rights to resources.

Typically, in traditional systems, the access rights are held as access control lists (ACLs) within each target resource. In an X.509 PMI, the access rights are held within the privilege attributes of ACs that are issued to users. Each privilege attribute within an AC will describe one or more of the user's access rights. A target resource will then read a user's AC to see if he or she is allowed to perform the action that is being requested.

Another authorisation scheme, which is popular with the military, is the mandatory access control (MAC) scheme. In the MAC scheme, every target is given a security label that includes a classification, and every subject is given a clearance that includes a classification list. The classification list specifies which type of classified target the subject is allowed to access. A typical hierarchical classification scheme that is used by the military is unmarked, unclassified, restricted, confidential, secret and top secret.

A typical security policy that is designed to stop information leakage is 'read-down and write-up', which specifies that a subject can read targets with a lower classification than his or her clearance and can write to targets with a higher classification. Under this policy, a user with clearance of confidential information who logs in as such could read from unmarked or confidential targets and write to confidential or top-secret targets. The same user could also log in with a lower clearance level and write to an unclassified target. X.509 supports MACs by allowing subjects to be given a clearance AC. The privilege attribute in the AC now holds the user's clearance. Targets can be configured securely with their own security label and the security policy that is to direct them.

1. *ITU-T Rec. X.509 (2001), ISO/IEC 9594-8,* The Directory: Authentication Framework.
2. *C Adams and S Lloyd (1999),* Understanding Public-Key Infrastructure: Concepts, Standards, and Deployment Considerations, *Macmillan Technical Publishing.*
3. *Unfortunately, X.509 did not standardise the term root CA and so disparate meanings for the term have evolved. Fortunately, this same mistake has not been made with PMIs.*

Table 1: A Comparison of PKIs with PMIs

Concept	PKI Entity	PMI Entity
Certificate	Public Key Certificate	Attribute Certificate
Certificate issuer	Certification Authority	Attribute Authority
Certificate user	Subject	Holder
Certificate binding	Subject's name to public key	Holder's name to privilege attribute(s)
Revocation	Certificate revocation list (CRL)	Attribute certificate revocation list (ACRL)
Root of trust	Root certification authority or trust anchor	Source of authority
Subordinate authority	Subordinate certification authority	Attribute authority

A more recent authorisation scheme is that of role-based access controls (RBACs). In the simple RBAC model, a number of roles are defined. Typically, they represent organisational roles such as secretary, manager, employee, etc. In the authorisation policy, each role is given a set of permissions, i.e. the ability to perform certain actions on certain targets. Each user is then assigned to one or more roles.

When accessing a target, a user presents his or her role and the target reads the policy to see if this role is allowed to perform this action. X.509 supports simple RBAC by defining role-specification ACs that hold the permissions granted to each role, and role-assignment ACs that assign various roles to the users. In the former case, the AC holder is the role, and the privilege attributes are permissions that are granted to the role. In the latter case, the AC holder is the user, and the privilege attributes are the roles that are assigned to the user.

The hierarchical RBAC model is a more sophisticated version of the simple RBAC model. With this model, the roles are organised hierarchically and the senior roles inherit the privileges of the more junior roles. Therefore, for example, there might be the following hierarchy:

employee > programmer > manager > director.

If a privilege is given to an employee role, for example, that the person can enter main buildings, each of the superior roles can also enter the main building, even though their role specification does not state this explicitly. If a programmer is given permission to enter the computer building, managers and directors would also inherit this permission.

Hierarchical roles mean that role specifications are more compact. X.509 supports hierarchical RBAC by allowing both roles and privileges to be inserted as attributes in a role-specification AC so that the latter role inherits the privileges of the encapsulated roles.

Privilege and Role Management Infrastructure Standards Validation (PERMIS) Project

The European Commission (EC)-funded Privilege and Role Management Infrastructure Standards Validation (PERMIS) Project has been given the challenge of building an X.509 role-based PMI that can be used by different applications in three cities of Europe. The project has members from Barcelona (Spain), Bologna (Italy) and Salford (UK). All three centres already have experience of running pilot PKIs and it was therefore natural for them to want to add a PMI capability in order to complete the strong authentication and authorisation chain. The chosen applications of the three cities are significantly different in character, so it will be a good test of the generality of the developed PMI if it can cater for each of them.

In the case of Bologna, the city wants to be able to allow architects to download road-maps of the city, to update the maps with their proposed plans and to upload the new building plans and requests for building licences to the city planning office's server. This should improve the efficiency of the current system significantly as, currently, the plans and requests are sent by post as paper documents to the city hall.

Barcelona is a major tourist and commercial centre and has many car hire locations throughout the city and at the airport. However, parking in Barcelona is restricted considerably and many parking tickets are issued frequently to hired cars. By the time that the car hire companies receive the parking tickets, the hirers have left the country.

The plan is to provide the car hire companies with online access to the city's parking ticket database so that when cars are returned at the end of their hire period, the company can check instantly to see if any parking tickets have been issued for the car. The company will be able to send the details of the driver to the city, thereby transferring the fine to the individual. Data protection legislation requires that a car hire company can only access the tickets that are

issued to its own cars and not to those of other car hire companies and so authorisation needs to be controlled strictly.

Finally, Salford intends to implement an electronic tendering application. This will start when the city places the request for proposal (RFP) documents on its website, allowing them to be downloaded by anyone. However, in some restricted tendering instances, only companies that have been authorised by Salford previously will be able to submit tenders.

In other cases, it may be a requirement that a company has International Organization for Standardization (ISO) 9000 or other certification in order to submit a tender. Once the tenders have been submitted, they must remain anonymous until the winner has been chosen. The city tender officers must not be given access to the electronic tender store before the closing date of the RFP, and tenderers must not be allowed to submit tenders after the closing date of the RFP.

The challenge for the PERMIS Project is to build a role-based X.509 privilege management infrastructure that can cater for these different applications and, in so doing, indicate that it will be useful to a much wider range of applications.

The PERMIS Privilege Management Infrastructure (PMI) Implementation

There are three main components to the PMI implementation – the authorisation policy, the privilege allocator (PA) and the PMI application programming interface (API).

Authorisation Policy

The authorisation policy specifies who has what type of access to which targets and under what conditions. Domain-based policy authorisation is far preferable than having separate ACLs configured into each target. The latter is hard to manage, duplicates the effort of the administrators – since the task has to be repeated for each target – and is less secure – since it is difficult to keep track of which access rights any particular user has across the whole domain.

Policy-based authorisation, on the other hand, allows the domain administrator – the SOA – to specify the authorisation policy for the whole domain, and all targets will then be controlled by the same set of rules.

The PERMIS Project decided early on to use the hierarchical RBAC model for specifying authorisations. RBAC has the advantage of scalability over DAC and can handle large numbers of users easily as, typically, there are far fewer roles than users.

The PERMIS Project wanted to specify the authorisation policy in a language that could be both parsed easily by computers and read by the SOAs with or without software tools. Various pre-existing policy languages – for example, Ponder – were examined, but none were found that were ideally suited. It was decided that XML was a good candidate for a policy specification language, since there are many tools available that support XML. It is fast becoming an industry standard and raw XML can be read and understood by the majority of technical people, as opposed to ASN.1,[5] for example, which uses a binary encoding.

First, a document type definition (DTD) was specified for the X.500 PMI RBAC policy. The DTD is a metalanguage that holds the rules for creating the XML policies. The DTD comprises the following components.

- SubjectPolicy – specifies the subject domains, i.e. only users from a subject domain may be authorised to access resources that are covered by the policy.
- RoleHierarchyPolicy – specifies the different roles and their hierarchical relationships to each other.
- SOAPolicy – specifies which SOAs are trusted to allocate roles.
- RoleAssignmentPolicy – specifies which roles may be allocated to which subjects and by which SOAs, whether delegation of roles may take place or not, and for how long the roles may be assigned.
- TargetPolicy – specifies the target domains that are covered by this policy.
- ActionPolicy – specifies the actions or methods that are supported by the targets, along with the parameters that should be passed along with each action, for example, action 'open' with parameter 'filename'.
- TargetAccessPolicy – specifies which roles have permission to perform which actions and on which targets and under what conditions. Conditions are

4. *N Damianou, N Dulay, E Lupu and M Sloman, "The Ponder Policy Specification Language",* Proceedings of Policy 2001, Workshop on Policies for Distributed Systems and Networks, *Bristol, UK, 29–31 January 2001, Springer-Verlag LNCS, 1995, pp. 18–39.*
5. *Abstract Syntax Notation One (ASN.1) is the syntax in which X.509 certificates are specified.*

specified using Boolean logic and may contain certain constraints. All of the actions that are not specified in a target access policy are denied.

Table 2 shows a portion of the DTD that specifies the rules for the role assignment policy, and below it is an example for the Salford application.

The SOA creates the authorisation policy for the domain using his or her preferred XML editing tool and stores this in a local file to be used later by the PA.

Privilege Allocator

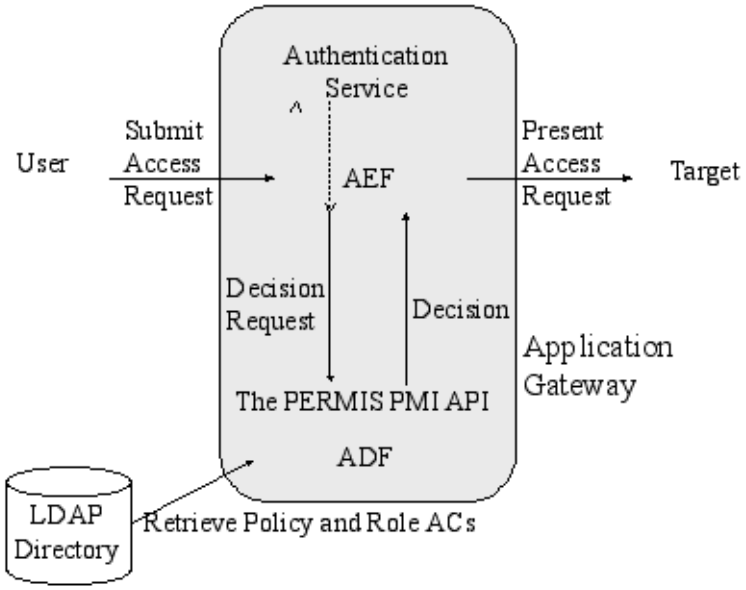
The PA is a tool that is used by the SOA or an AA to allocate privileges to users. Since PERMIS is using RBAC, the SOA uses the PA to allocate roles to users in the form of role-assignment ACs. These will be given to all users of the various applications in the different cities.

In the case of Bologna, there are two roles – map readers and architects. Map readers can download any maps that are produced by the municipality, whereas architects are allowed to download and upload maps that are modified digitally. In the case of Barcelona, there are also two roles defined – generalised and authorised. Any citizen or business can be allocated the generalised role. Anyone with the generalised role has permission to read their own pending car parking fines. Businesses that have signed an agreement with the city council are given the authorised role. Authorised roles can read their own pending fines and also may modify the details of them, for example, update the driver's name and address.

Salford is different to the other sites in that, while it will allocate two roles – that of tenderer and tender officer – it will also rely on an external SOA – in this case, the British Standards Institution (BSI) – to allocate the role of ISO 9000 certified to users. In the project, the plan is to set up a proxy BSI SOA to allocate these roles as BSI is not a project partner.

Once the role-assignment ACs have been created by the PA, they are stored in a Lightweight Directory Access Protocol (LDAP) directory. Since ACs are signed digitally by the AA who issued them, they are tamper-resistant and, therefore, there is no modification risk from allowing them to be stored in an LDAP directory that is accessible publicly. This also means that authorities who issue digital ACs can store them locally but give global access to them. This may be particularly useful in the case of ISO 9000 certificates, for example.

Figure 1: The PERMIS API Architecture



Anyone wishing to know if an organisation has ISO 9000 certification will access the BSI LDAP directory and retrieve the organisation's X.509 AC. The ACRLs of revoked certificates, if any, will also be stored here. Therefore, in general, there is little advantage to distributing the ACs to their holders, since a relying party will still need to access the issuing authority's LDAP directory to retrieve the latest ACRL.

Another function of the PA is to create an authorisation policy that is signed digitally as a policy AC. The policy AC is a standard X.509 AC with the following special characteristics – the holder and issuer name are the same, i.e. that of the SOA, the attribute type is pmiXMLPolicy and the attribute value is the XML policy created as mentioned previously. The PA prompts the SOA for the name of the policy file and then it copies the contents into the attribute value. After the SOA has signed the policy AC, the PA stores it in the SOA's entry in the LDAP directory.

The PMI Application Programming Interface (API)

A standard authorisation API has already been defined by the Open Group. It is called the AZN API[6] and is specified in the C language. It is based on the ISO 10181-3 access control framework[7] and specifies the interface between the access control enforcement function (AEF) and the access control decision function (ADF) (see *Figure 1*).

6. *The Open Group,* Authorization (AZN) API*, January 2000, ISBN 1-85912-266-3.*
7. *ITU-T Rec X.812 (1995), ISO/IEC 10181-3:1996,* Security Frameworks for Open Systems: Access Control Framework.

PERMIS has drawn on this work and made the following changes to it. First, it has specified the PERMIS API in Java™ rather than in C and, second, it has simplified the AZN API significantly by assuming that the target and the AEF are either co-located or can communicate with each other across a trusted local area network (LAN). It has also been assumed that only a single authorisation service will be available and that the API does not need to support the export of authorisation tokens as ACs are already in an exportable format.

Figure 1 shows the PERMIS API architecture. A user accesses resources via an application gateway. The AEF authenticates the user and then asks the ADF if the user is allowed to perform the required action on the particular target resource. The ADF accesses one or more LDAP directories to retrieve the authorisation policy and the role ACs for the user, and bases its decision on these.

The PERMIS API comprises four simple calls – 'initialise', 'get creds', 'decision' and 'shutdown'. The functionality of the calls is as follows. Initialise is called to tell the ADF to read in the policy AC. The AEF passes the name of the trusted SOA and a list of LDAP Uniform Resource Identifiers (URIs) from where the ADF can retrieve the policy AC and, subsequently, role ACs. Initialise is called immediately when the AEF starts up. After initialise has completed successfully, the ADF will have read in the XML policy that will control all of the future decisions that it makes.

When a user initiates a call to the target, the AEF authenticates the user and then passes the LDAP distinguished name (DN) of the user to the ADF through a call to get creds.

In the three cities, the user will be authenticating in different ways. In Salford, the user will be sending a Secure/Multipurpose Internet Mail Extension (S/MIME) e-mail message to the AEF. In Barcelona and Bologna, the user will be opening a secure socket layer (SSL) connection. In both cases, the user will be signing the opening message digitally and verification of the signature will yield the user's DN. The ADF uses this DN to retrieve all of the role ACs of the user from the list of LDAP URIs that are passed at initialisation time. The role ACs are validated against the policy, for example, to check that the DN is within a valid subject domain and to check that the ACs are within the validity time of the policy, etc. Invalid role ACs are discarded, while the roles from the valid ACs are extracted and kept for the user.

Once the user has been authenticated successfully, he or she will attempt to perform certain actions on the target. At each attempt, the AEF passes the target name and the attempted action, along with its parameters to the ADF via a call to decision. Decision checks whether the action is allowed for the roles that the user has, taking into account all of the conditions that are specified in the target access policy. If the action is allowed, decision returns 'true'; if it is not allowed, it returns 'false'. The user may attempt an arbitrary number of actions on different targets and a decision is called for each one.

In order to restrict the user from keeping the connection open for an infinite amount of time, for example, until after his or her ACs have expired, the PERMIS API supports the concept of a session time out. On the call to 'get creds', the AEF can specify how long the session may stay open before the credentials should be refreshed. If the session times out, 'decision' will throw an exception, telling the AEF to either close the user's connection or call 'get creds' again.

Shutdown can be called by the AEF at any time. Its purpose is to terminate the ADF and cause the current policy to be discarded. This could happen when the application is shut down gracefully or if the SOA wants to impose a new authorisation policy on the domain dynamically. The AEF can follow the call to shutdown with a call to initialise and this will cause the ADF to read in the latest authorisation policy and be ready to make access control decisions again.

Progress to Date

Version two of the PERMIS X.500 PMI RBAC policy DTD has been published and version one of the PA tool has been released to the PERMIS Project participants. Version 0.2 of the PERMIS PMI API has also been released to the PERMIS Project participants, and public releases will be made available after the signing of an appropriate licence agreement.

The generality of the PERMIS API has already proven its worth. The author is the leader of another research project at Salford that is designing an electronic prescription processing system and has found that the PERMIS API can be incorporated easily into the electronic dispensing application. With a suitable policy, the ADF is able to make decisions about whether a doctor or a pharmacist is allowed to issue a prescription and whether a patient is entitled to free prescriptions. It is expected that many more applications will use the API in due course. ■

Table 2: The Role Assignment Policy DTD and an Example of Salford's Role Assignment Policy

```
<!ELEMENT RoleAssignmentPolicy (RoleAssignment)+ >
<!ELEMENT RoleAssignment (SubjectDomain,Role,Delegate,SOA,Validity) >

<!ELEMENT SubjectDomain EMPTY>
<!ATTLIST SubjectDomain ID IDREF #REQUIRED>

<!ELEMENT Role EMPTY >
<!ATTLIST Role Type IDREF #IMPLIED
       Value IDREF #IMPLIED >

<!ELEMENT SOA EMPTY>
<!ATTLIST SOA ID IDREF #REQUIRED>

<!ELEMENT Validity (Absolute?, Maximum?, Minimum? ) >
<!ELEMENT Absolute EMPTY>
<!ATTLIST Absolute Start CDATA #IMPLIED
        End CDATA #IMPLIED >
<!ELEMENT Maximum EMPTY>
<!ATTLIST Maximum Time CDATA #IMPLIED >
<!ELEMENT Minimum EMPTY>
<!ATTLIST Minimum Time CDATA #IMPLIED >

<!ELEMENT Delegate EMPTY >
<!ATTLIST Delegate Depth CDATA #IMPLIED >
```

```
<RoleAssignmentPolicy>
  <RoleAssignment>
<!-- Role assignment for tender officers.
They must be employees of Salford City Council. Valid only from close of tender.
Delegation not permitted -->
    <SubjectDomain ID="Employees"/>
   <Role Type="permisRole" Value="TenderOfficer"/>
   <Delegate Depth="0"/>
   <SOA ID="Salford"/>
   <Validity>
    <Absolute Start="2001-09-21T17:00:00"/>
   </Validity>
  </RoleAssignment>
  <RoleAssignment>
<!-- Role assignment for tenderers.
They must be dotcom or co.uk companies. Valid only until close of tender.
Delegation not permitted -->
    <SubjectDomain ID="Companies"/>
   <Role Type="permisRole" Value="Tenderer"/>
   <Delegate Depth="0"/>
   <SOA ID="Salford"/>
   <Validity>
    <Absolute End="2001-09-21T17:00:00"/>
   </Validity>
  </RoleAssignment>
  <RoleAssignment>
<!-- Role assignment for companies who are ISO 9000 Certified.
They must be dotcom or co.uk companies. Valid only for a maximum of one year as companies have
to be re-accredited annually. Certificates are issued by BSI.
```

```
Delegation not permitted -->
    <SubjectDomain ID="Companies"/>
   <Role Type="ISOCertified" Value="ISO9000"/>
   <Delegate Depth="0"/>
   <SOA ID="BSI"/>
   <Validity>
    <Maximum Time="+01"/>
   </Validity>
  </RoleAssignment>
</RoleAssignmentPolicy>
```