Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers

Amir Herzberg
IBM Haifa Research Lab
amir@il.ibm.com

Yosi Mass
IBM Haifa Research Lab
yosimass@il.ibm.com

Joris Michaeli
IBM Haifa Research Lab
michaeli@il.ibm.com

Dalit Naor
IBM Almaden Research Lab
dalit@almaden.ibm.com

Yiftach Ravid[1]
GammaSite Ltd.
yiftach@gammasite.com

Abstract

The Internet enables connectivity between many strangers - entities that don't know each other. We present the Trust Policy Language (TPL), used to define the mapping of strangers to predefined business roles, based on certificates issued by third parties. TPL is expressive enough to allow complex policies, e.g. non-monotone (negative) certificates, while being simple enough to allow automated policy checking and processing. Issuers of certificates are either known in advance, or provide sufficient certificates to be considered a trusted authority according to the policy. This allows bottom-up, 'grass roots' buildup of trust, as in the real world.

We extend, rather than replace, existing role-based access control mechanisms. This provides a simple, modular architecture and easy migration from existing systems.

Our system automatically collects missing certificates from peer servers. In particular this allows use of standard browsers, which pass only one certificate to the server.

We describe our implementation, which can be used as an extension of a web server or as a separate server with interface to applications.

Keywords: Authentication, key management, role based access control, trust management, logic programming, public key certificates, X.509.

1. Introduction

The Internet is quickly becoming the largest marketplace, allowing commerce and business between parties who are physically distant and do not know each other. In many (or most) business relationships, the parties need to establish some trust in each other, by receiving references from trusted intermediaries (such as letters of credit). It is recognized that, on the Internet, this trust can be facilitated using public key certificates. Indeed, the creation of recognizable and meaningful public key certificates infrastructure for Internet-wide use is long considered a critical problem for the success of electronic commerce. Unfortunately, this did not happen so far. We believe that part of the reason is that the traditional approach was to create a single, top-down, Internet-wide public key infrastructure, providing identification of subjects. We advocate a different approach, allowing bottom-up, `grass roots` buildup of the public key infrastructure, beginning from isolated `islands` (typically Intranets) gradually being connected to cover the entire Internet, and using the certificates to convey any useful reference about the subject, not necessarily its identity.

1.1 Trust Establishment and Access Control

The trust establishment problem is a variant of the well-known *Access Control (AC) problem*. A simple AC system as depicted in Figure 1 is a black box that accepts a query: "Can user U perform action A on resource R" and returns a Yes (Y) or No (N) answer. A user is typically identified to the system by username and password.

[1] Work done while at the IBM Haifa Research Lab.

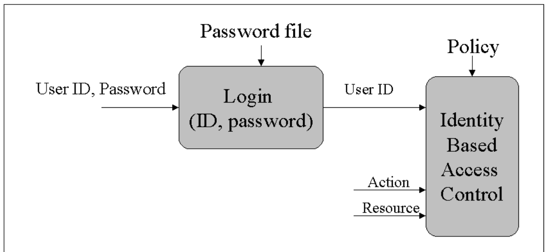

Figure 1: Identity based Access Control

A more sophisticated AC system is a *role based access control (RBAC) system* [11] as in Figure 2. A RBAC system has two phases in assigning a privilege to a user: first the user is assigned one or more roles, and then the roles are checked against the requested operation. Role based access control systems reduce the number of Access Control decisions, since they map users to roles (one/few mappings for each user), and then roles to permissions; and the number of roles is typically much smaller than the number of users. For example, the Unix OS has a list of all users who can access the system (/etc/passwd file) and a list of groups with a mapping of users to groups (/etc/group).

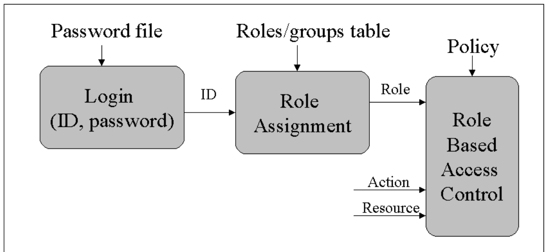

Figure 2: Role Based Access Control

Systems that require a stronger authentication can replace the login process of username, password with a public key certificate that is used to identify the accessed user, as in Figure 3. Certificates contain a public key, and properties of the owner of the corresponding secret key; in this case, the relevant property is the identity of the owner. A trusted Certification Authority (CA) digitally signs each certificate, binding the attributes with the owner – specifically, providing the identity (name) of the owner of the private key. The login process is replaced by an authentication protocol such as SSL, which verifies that the user has the secret key. This is the mechanism used, e.g., by certificate-based authentication by current browsers and servers.

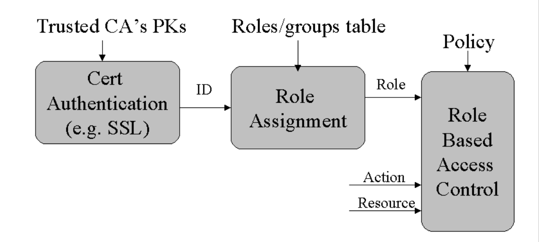

Figure 3: RBAC with Identity Certificates

The mechanisms described assumed that every user is known to the system in advance, with an entry in either the password file or the Roles/groups table. As previously explained, for many applications we need to control access also to users and entities not known in advance. The certification authentication system simply outputs the entire certificate, rather than extracting and forwarding just the identity field from the certificate. This is input to a *Trust Establishment* system, which identifies a role, based on a policy mapping from certificates to roles. See Figure 4.

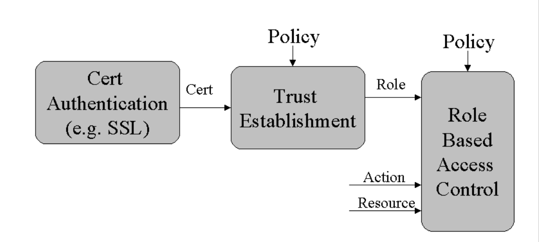

Figure 4: Trust Establishment and RBAC

Very simple Trust Establishment modules are available as part of some systems, such as the IBM S/390 Resource Access Control Facility (RACF) [5] and Policy Director[20]. However, these modules just perform simple mappings from the distinguished names of the issuer and subject of the certificate to a role (e.g. if the distinguished name ends with ORG=/IBM, the role is IBM employee).

We show a more powerful Trust Establishment (TE) system, with a strong language to define the trust policy. Our TE system can handle multiple certificates for the subject, collecting some of them itself, as discussed later. The system maps the subject of the certificates to a role, based on the subject's certificates, on a given role-assignment policy set by the owner of the resource and on the roles of the issuers of the certificates.

1.2 Potential Applications

Trust Establishment may be applicable wherever entities want to engage in sensitive (trust-requiring) transactions, without sufficient pre-established direct trust. Such applications involve essentially every aspect of e-Business, such as electronic marketplaces, e-government, banking and securities trading. For example, in auction sites such as e-Bay, buyers need to trust sellers (to actually deliver) and sellers need to trust buyers (to pay). Currently, the only mechanism to establish trust is through a very limited history of previous transactions kept by the site.

Another example, which will be used throughout this paper, is a hospital's policy to enable access to large databases of anonymous medical data for research purposes, while limiting the access only to authorized people. The hospital may allow access to cardiology records only to cardiologists, to oncology data only to oncologists, etc. A cardiologist is a doctor presenting a certificate from a recognized hospital. A recognized hospital can be either known locally or certified by at least two already recognized hospitals. In this example there is no need to have a root authority that certifies all hospitals. Hospitals from different countries can cross-certify each other to create a web of trust, enabling doctors to share data. This simple Trust Policy may be described visually as in Figure 5, and the rule for adding new hospitals, in Trust Policy Language, is in Figure 6 shown later on.

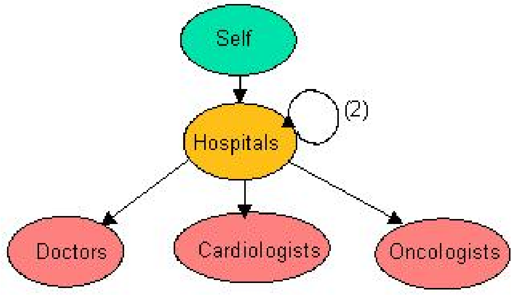

Figure 5 - Policy for medical data access

The group of recognized hospitals (i.e. public key in Hospitals role) may grow dynamically and is not taken from an a priori list of authorized hospitals. Thus, if a certificate is issued for a new hospital by two certified hospitals, the new hospital is also rendered a certified hospital, which now can issue certificates to e.g. cardiologists. Hospitals from different countries can cross-certify each other to create a global web of trust.

Yet another example would be the internal access control policy to files and documents places on the internal web sites of a large company (Intranet). Large companies often have complex matrix organizations, organized e.g. by geography, business lines (e.g. marketing, sales, research and development), and product lines (e.g. storage products and personal computers). Employees may have several roles from the point of view of every particular intranet web site.

More application examples can be community of suppliers/providers, such as the car or electronic components industries, travel planning where certificates are evaluations by tourist agencies, consumer discount clubs, loan applications, peer reviews, and more.

1.3 Related Works on Trust Management

The non-hierarchical, `web of trust` model for public key certificates which we advocate in this paper, was first deployed in the popular Pretty Good Privacy (PGP [16]) secure e-mail system. However, PGP supports only a very restricted policy, based on each user defining fully trusted and partially trusted users and rules for how many `references` from trusted users are enough to make a new user (partially) trusted.

Our work is more closely related to KeyNote [9], Policy Maker [4] and REFEREE [10]. All three works suggest a programming language to define a policy based on certificates and an engine to answer the question: "Can the holder of a certificate X perform action A on resource R?" Hence, they provide an integrated solution to Trust Establishment and Access Control. This makes the solution more complex and more difficult to integrate with existing systems. Another important drawback in PolicyMaker, and even more in REFEREE, is that the policy is defined as a complex, fully programmable language, hard to define for a non-programmer. While KeyNote's policy language is substantially simpler, it is still hard for non-programmers. All three systems do not provide mechanisms to collect missing certificates (except in a very limited, `manual` way in REFEREE) and are 'assertion monotone', which means that only positive certificates are considered with no support to negative certificates.

Another related work is the IETF's Simple Public Key Infrastructure (SPKI) [7]. This work defines two relevant notions: one is a method to define global identifiers for entities and the second is a suggestion for trust management embedded in certificates. They claim that the notion of an identity certificate which binds a key to some name requires a global name space with unique names, such as the X.500 DN. However, not only did this concept fail, but this binding is not necessary since all we want to know is the privileges of the key holder and not its name. Therefore, they suggest that the unique global name should be the key itself. We adopt their observation and also use only public keys to identify entities. Their other suggestion is to embed a "Tag" field in a certificate that will contain the privileges of the key holder. We object to this approach, as the issuer of the certificate may be unaware of the access control needs of the owner of the resource; in our work we separate the data in the certificate from the policy itself.

Winslett et al. [8] discuss the issue of using certificates with attributes to map users to roles, but they do not describe a policy language. The paper also discusses a policy for certificate collection but only through naive certificate chains.

Seamons et al. [13] discuss a concept of mapping users to role based on certificates using the prolog programming language. The drawback in this solution is that there is no policy language, but a separate program has to be written for each application, which poses undue constraint on the user. In addition, their proposed approach relies on simple certificate chains which lead to root CA.

Trust establishment is only a part of the security requirements for enabling business between e-strangers; a complete solution will need to address the complete access control question as well as other issues, e.g. contract enforcement and dispute resolution. There are several attempts at designing complete designs for securing business between e-strangers. In particular, Gladney [14] proposes such a design, with a very limited policy for mapping from certificates to roles – but with a specific proposal for integrating access control (based on ticket granting servers), contract enforcement and dispute resolution protocols.

A work closely related to ours is [15], which extends a logic, programming language with constructs to support trust. The constructs described there, such as **threshold** and **delegation,** can be expressed in our policy language through the **repeat** tag and the **membership** certificate. While using a general-purpose logic programming can solve a more general problem, we believe that our language, which is focused toward solving a specific problem, is easier for the system administrator who needs to define her business policy, and can be more efficient. In addition the TE system collects missing certificates, and therefore is able to solve goals even if there are not enough assertions in the local engine.

2. Design Principles

We now describe some of our guidelines and principles, motivating the Trust Establishment approach and the Trust Policy Language.

2.1 Principles (entities) and names/identifiers.

We consider, essentially, three types of entities: the *owner* of a resource defining the policy on access to it; the *subject* requesting specific action (access) on the resource; and *issuers*, which issue certificates with some properties to the subject or to other issuers.

We focus on trust based on cryptographically signed or authenticated statements. The name or identity of the entity making the statement is not really meaningful and for our purposes, the only relevant identifiers are the public keys (or cryptographic one-way hashes of them), which simply identify the corresponding private keys. Names, including distinguished names in X.509 certificates, and other identifiers can only be interpreted as specific attributes associated to the owner of the private key.

2.2 Certificates.

A certificate is a statement signed by *the issuer's public key*, identifying a *subject's public key* and properties of the holder of the corresponding subject's private key. This implies that we use an internal, generic certificate object, transcoding from specific certificate formats such as X.509v3 [1], SPKI [7], PGP [16] or KeyNote [9]. Currently we have implemented transcoding only from X.509v3 certificates; adding additional transcoders should be easy. Namely, we consider our system to be *certificate format independent.*

We find it convenient to categorize certificates, by defining a special attribute of the generic certificate object which we call *certificate type*. Notice that the certificate type does not necessarily have to be an actual attribute of the `real` certificate, but instead may be added by the transcoder. Certificate types help us identify the semantic and syntax of certificates from different issuers, which is essential to ensure interoperability. Every certificate must have a *certificate type*, where the set of all certificate types is dynamic and easily extendible. The *certificate type* must be unique, so either it is registered through some central organization or it becomes unique by adding to it the originator prefix (e.g. ibm.employee).

Each certificate type has a *certificate profile*, which defines the certificate structure; namely, which fields it is composed of, and for each field what types of values it can admit. The certificate profile must address two issues:

- Syntax - A listing of all fields, types of values each field can admit and the mandatory fields. The syntax is expressed in XML to ease interoperability.
- Semantics - A free text explanation of the meaning of each field and its corresponding values.

The issuer is likely to be different from the owner of the resource. Therefore, the properties in the certificate represent beliefs of the issuer with respect to the subject, rather than the policy of access to the resource, which is defined by its owner. This is the *principle of separation of issuers from authorization.*

When decisions are made on the basis of certificates, an obvious question is: do we have all the necessary certificates to decide? Almost all existing systems require the subject to provide all the relevant certificates; this makes some sense as the subject is making the request. However, in many cases, this requirement is not realistic. In particular, the subject may be limited in ability to store multiple certificates (e.g. on smartcard), or to transfer multiple certificates (e.g. on wireless link or when using SSL, which sends only one certificate chain). To solve this, we allow certificates to contain a special attribute, which is the address of a repository where additional certificates may be found.

In real life, trust decisions are influenced by negative opinions and reviews as well as by positive ones. Traditional approaches to certificates focus on the relatively simple case of revocation of a certificate, as a negative indication – rendering a certificate null. We extend this to support general *negative certificates*. We cannot trust the subject to point us at the repositories of negative certificates; the owner should define, as part of the policy, which repositories should be searched for negative certificates.

To summarize, the following are mandatory components in TE certificate object:

1. **Issuer's public key –** as identifier of the issuer.
2. **Subject's public key –** an identifier of the subject.
3. The certificate type
4. Version of the certificate.
5. **profileURL** - URL that describes the certificate type; namely, its structure and semantics.
6. **issuerCertRepository** - Provides addresses where to look for more certificates for the issuer (in order to map the issuer to some group), as well as requesting most updated CRL's to verify certificate validity.
7. **subjectCertRepository** - Provides addresses where to look for more certificates for the subject. This is important mainly in cases where the subject can present only one certificate (because it has a limited capacity on her accessing device or is limited by the accessing protocol such as SSL).

Each certificate may include more fields, where a field is identified by its name (a string), and its value can be numeric, a string, a range of values, or a set of strings or numbers.

2.3 Trust Policy Language.

The TE system enables a business to define a flexible policy for role assignment, which supports dynamic ad-hoc relationships and a `web of trust` - allowing complex networks of trust rather than requiring a pre-defined tree with a fixed `root certification authority`.

Processing of the policy is essential, to ensure reasonable efficiency (e.g. in handling a new certificate or revocation), to check policy (e.g. for conflicts), to collect missing certificates, to compose policies, and to allow subjects to select which certificates to present. We also expect policy (fragments) to be shipped around, e.g. policies of a large company defined centrally with limited local refinements. For this reason, we used XML [3] to define the policy language, which helps portability and provides several automated processing tools. In particular, we recently developed a visual tool for editing graphs, and, in particular, TE policies, which can produce the XML representation or accept XML representation as input.

2.4 System considerations.

The TE system does not require replacing or re-engineering existing role-based access control systems. Instead, it extends them by mapping unknown users to roles.

3. The Trust Policy Language

The main purpose of the Trust Policy Language (TPL) is to map *entities* to *roles,* using well defined logical rules. A *role* in TPL is a group of entities that can represent a specific organizational unit (e.g. employees, managers, auditors). Entities are identified by their public keys. A special *role* is 'self', which includes the key of the policy owner. Each *role* has one or more *rules* defining how a certificate holder can become a member in the *role*. The *rules* are OR(ed); namely, it's enough that one *rule* holds for mapping an entity to a *role*. Note that we use the terms *role* and *group* interchangeably.

The language is defined using XML [3] where *roles* are defined at the top level and under each role there are *rules* for *role* membership. See Figure 6 for illustration of the rule to add a new hospital to the Hospitals group (role), by providing two recommendations from existing members of the Hospitals group, as in the policy illustrated in Figure 5. In this section we describe the main components of this syntax; see our site [18] for the complete definition.

```
 <GROUP NAME="Hospitals">
     <!---- hospital recommended by  at least 2 hospitals  --->

     <RULE>
         <INCLUSION ID="reco" TYPE="Recommendation"
FROM="hospitals"  REPEAT="2"></INCLUSION>
         <FUNCTION>
            <GT>
               <FIELD ID="reco" NAME="Level"></FIELD>
                <CONST>1</CONST>
             </GT>
         </FUNCTION>
       </RULE>
  </GROUP>
```

Figure 6: Rule for hospital membership

There is a separation between the issuers of the certificates and the owner of the resource (except for 'group membership' certificate – see below). Only the owner defines the trust and access control polices. The certificates are general statements about the subject (e.g. a user can have a certificate from some institute with her degree and average marks) and a company policy can state the conditions on certificate fields (e.g. an employee should present a certificate from a recognized institute and the average marks should be higher than 80).

We describe two flavors of the TE policy language. The first one is called ***DTPL*** (Definite Trust Policy Language) which is monotonic and does not include negative rules. We show that DTPL specifications may be mapped to Prolog. The stronger ***TPL*** (Trust Policy Language) is non-monotonic since it includes negative rules. In future work we hope to show completeness and soundness of these languages.

3.1 Definite Trust Policy Language (DTPL)

3.1.1 The Group tag

A policy in DTPL (and TPL) consists of a sequence of definitions of groups (roles) using the <GROUP> tag. The only attribute of <GROUP> is NAME – the name of the group. Within the scope of each <GROUP> tag there is one or more <RULE> tags, each defining one rule for membership in the group; it is sufficient for one of the rules to hold for the entity (public key) to be added to the group.

3.1.2 The Rule tag

A rule defines a set of certificates necessary to join a group. Two types of requirements are possible on certificates: the issuer needs to belong to a specific group, and the attributes in the certificates may need to match some conditions. For example, in the rule presented in Figure 6, two certificates are required (REPEAT=2), both from already recognized hospitals, and of type Recommendation. Furthermore, the Level field in both should be more than 1.

We include two tags for defining the necessary certificates: the <INCLUSION> tag defines each of the necessary certificates, and the <FUNCTION> tag defines necessary conditions on the attributes. A rule may contain multiple <INCLUSION> tags, but only one <FUNCTION> tag.

3.1.3 The Inclusion tag

The inclusion tag defines a certificate that must exist for the rule to hold. For example, the tag <INCLUSION ID="C1" TYPE="T1" FROM="G1"></INCLUSION> stands for "exists a certificate of type *T1* whose issuer belongs to group *G1*". The basic attributes here are:
Type parameter - specifies the necessary type of the certificate, as explained in section 2.2.
From parameter - defines the name of one or more groups to which the issuer should belong.
ID parameter – an identifier for the certificate. It refers to this certificate within the <FUNCTION> tag to define additional conditions on the certificate.

An important attribute that can appear in an inclusion statement is "**REPEAT=k**". This defines that at least *k* certificates of that type should exist, from different issuers, for the rule to hold. A "REPEAT=2" parameter is used in the medical data access example from figures 5 and 6. Namely, to become a recognized hospital, we require two certificates from already recognized hospitals.

Another important attribute that can appear in an inclusion statement is the "**DEPTH=k**" which is used to limit the length of certificate chains. Consider a rule that requires a certificate with an issuer from some group. The issuer was put into this group by virtue of some other certificates from other issuers, and so on, until direct assignments (which we consider certifications by self). The owner may want to restrict the amount of indirection, namely the depth (length) of the chain of certificates. We facilitate this by the **DEPTH** parameter of the INCLUSION tag.

For example, if we look again at the hospitals example in figures 5 and 6 above, there is a rule that a hospital can be mapped to the "Hospitals" group if it brings at least 2 certificates from already trusted hospitals. We can limit the depth of that rule by setting DEPTH, as in Figure 7.

```
<GROUP NAME="Hospitals">
        <!---- hospital recommended by  at least 2 hospitals  --->
<RULE>
     <INCLUSION ID="reco" TYPE="Recommendation"
FROM="hospitals"  REPEAT="2" DEPTH="3">

     </INCLUSION>
</RULE>
</GROUP>
```

Figure 7: DEPTH attribute of INCLUSION tag

The result is any hospital was selected by the owner (self) directly (depth=1), by hospitals selected by self (depth=2), or by hospitals selected by hospitals selected by self (depth =3) – but no more indirection.

3.1.4 The Function tag

This tag allows definition of additional conditions over the certificates, as a function of the certificate fields. DTPL supports a syntax for simple operators (e.g. comparing two values, AND between two Boolean expressions) which is expressed as a computation tree with conditions on the certificates fields. It also enables invocation of an external code for more complex computations. For complete details on the expressions supported by the <FUNCTION> tag, see the complete definition in our site [18].

For example, in the hospitals policy of Figure 6, the function states that the value of the field "Level" (NAME="Level") in both certificates should be greater than 1. Note that in this example we assume that certificate of type "Recommendation" has a field called "Level".

3.1.5 The group membership certificate

As mentioned above, we advocate separation between certificate issuers and the policy owner. We assume most certificates carry some general statement about the certificate's subject (e.g. X is an IBM employee) and not some policy specific enforcement as exists in other systems (e.g. SPKI [7]).

However, in this section we illustrate that DTPL allows the owners to delegate the membership decision (is X in G). For that end, we suggest that owners use a special certificate type, e.g. *certType=membership,* which has an attribute named *groups,* holding a list of policy groups that the subject can be mapped to. This rule can be expressed as in Figure 8 below.

```
<GROUP NAME="Hospitals">
        <RULE>
             <INCLUSION ID="c1" TYPE="membership"
FROM="delegators">

             </INCLUSION>
             <FUNCTION>
                  <ITEM>
                       <CONST>Hospitals</CONST>
                       <FIELD ID="c1"
NAME="groups"></FIELD>
                  </ITEM>
             <FUNCTION>
        </RULE>
    </GROUP>
```

Figure 8: Group membership certificate

The policy above assumes a group "delegators" which includes all the trusted delegators. The rule above states that if exists a certificate of type 'membership' where X is the subject, and the issuer is one of the trusted delegators, and the group 'Hospitals' appears as one of the groups in the certificate (the ITEM tag), then according to the policy, X can be mapped to the 'Hospitals' group.

3.2 DTPL as a logic programming language

DTPL (Definite TPL) is monotonic since it does not include negative rules. It can be easily shown that DTPL can be mapped to a standard logic programming e.g. Prolog [15]. We show now how to express DTPL in Prolog and analyze the advantages of using our own language implementation over Prolog.

1. Each certificate can be expressed as a predicate of the form *cert (_Issuer, _Subject, _Type, _Fields).*

2. The predicate *field(_Fields, _FieldName, _FieldValue)* is used to denote that a field *_FieldName* is one of the fields in the *_Fields* variable and its value is *_FieldValue.*

3. The predicate *group(X, Group)* is used to denote that *X* is a member in group *Group*.

4. The DTPL *function* tag can be programmed using standard prolog programming.

5. We can now define rules for group membership in group G as clauses with head *group(_Member, _Group)* and body as a prolog program.

Getting back to the hospitals policy (Figures 5 and 6), the first rule that states that an hospital *X* can be mapped to the *Hospitals* group if *X* has a *recommendation* certificate signed by someone from the self group, can be expressed in DTPL:

```
  <GROUP NAME="Hospitals">
      <!---- A hospital recommended by  self  --->

      <RULE>
          <INCLUSION ID="reco" TYPE="Recommendation"
FROM="self"></INCLUSION>
       </RULE>
   </GROUP>
```

And can be mapped to the following prolog program:

```
group(X, Hospitals) :-
        cert(Y, X, "Recommendation", _RecFields),
        group(Y, self),
```

The more complicated DTPL rule in Figure 6 states that an hospital *X* can be mapped to the *Hospitals* group if *X* can show at least two different *recommendation* certificates whose issuers are already known to be in the *Hospitals* group, and their *recommendation* level is higher than 1. The corresponding prolog program is:

```
group(X, Hospitals) :-
        cert(Y1, X, "Recommendation", RecFields1),
        cert(Y2, X, "Recommendation", RecFields2),
        Y1 != Y2,
        group(Y1, Hospitals),
        group(Y2, Hospitals),
        field(RecFields1, "Level", L1), L1 > 1,
        field(RecFields2, "Level", L2), L2 > 1.
```

Note that if the policy requires at least k certificates (Repeat=k where k is larger than 2), the Prolog program may become quite complex.

There are some advantages to using our language and our policy engine over using a prolog interpreter. First, DTPL has special constructs such as the *repeat* and *depth* tags, which would require some extra logic programming for each policy (see example above how the Repeat tag can be implemented in Prolog). In addition we support the *proof* for each mapping and use it to check validity while implementing this in a general Prolog policy would require non-trivial programming for each policy.

Another feature that our role assignment module supports is dynamic extension of the assertion database through the collector, while a Prolog interpreter can solve goals only from its existing assertion base. Moreover since TE is targeted at a specific task of mapping entities to roles, we can optimize the algorithm over a general-purpose Prolog engine.

3.3 TPL – extending DTPL with negative rules

We extend our policy language with *negative certificates,* namely certificates which are interpreted as suggestions not to trust a user or not to assign him with a given role or group. In this case, the rule states that a user cannot be assigned the role if there exists a negative certificate of given type about it.

The syntax we introduce is the "Exclusion" tag, which has the same attributes as the "Inclusion" tag. The semantics of the exclusion tag is "Not exist a certificate X such that the rule's function holds".

Notice that for positive certificates, we assumed that the user will provide the certificate repositories with the submitted certificate. However this assumption is not reasonable for negative certificates. Instead, we require the policy owner to define where the system should look for negative certificates. This mimics the real life, where a part of an organization or individual policy is which sources are queried for bad references (e.g., do you query the BBB or TRW, etc.).

We support the specification of repositories by adding a new tag to the rule, the <REPOSITORY> tag. Repositories may be specified as a URI, using the HREF parameter of the <REPOSITORY> tag. Alternatively, specific groups may be defined as repositories, so that the TE system will automatically query all of them for negative certificates. This is done using a GROUP parameter of the <REPOSITORY> tag.

4. Trust Establishment implementation

We have implemented a prototype of the Trust Establishment system, using Java and X509v3 for the certificate format. We describe now its four main components; the certificate library, the policy engine, the certificate collector and the database. We also describe the integration with a web server.

4.1 The Certificate Library

A certificate object in the Trust Establishment system (TEcertificate) is a statement signed by the Issuer, which contains the Subject's public key and a list of <attribute,value> pairs. Certificates may expire or be revoked.

The designs calls for a library of transcoders from different certificate formats to the abstract certificate object. We currently support only transcoder from the X509v3 [1] format. The X509v3 format is currently by far the most common and widely used format for digital certificates and its latest version (version 3) provides sufficient flexibility that could be used to implement all requirements stated above. We currently support only the following simple transcoding from X.509v3 certificates and the certificate object:

Attributes: X509v3 format allows to define as many additional extensions as needed to the basic X509v1 object, where an X509 extension is identified by Object Identifier (OID) [12] and have a Value. OIDs are strings of numbers (e.g. 1.20.3.5) allocated in hierarchical order so the owner of an OID can create new OIDs with the same prefix. Since the policy deals with names and not with OIDs, we have a mapping from OIDs to names. The mapping is kept in the certificate profile. We use an X509 extension to define attributes of certificates: the attribute name becomes the extension's Name and the attribute's value is the extension's Value. X.509 extensions may also be defined as "mandatory" and this mechanism is used to define the mandatory attributes in a TEcertificate such as *certType, version*, *profileUrl, subjectCertRepository* and *issuerCertRepository.*

The Issuer and Subject: The Issuer and Subject fields in an X509 certificate are based on a textual format called distinguished names. Therefore they can not be used as the Issuer and Subject identities in the TE system, which should be public keys or their cryptographic hash. Instead, the principal's unique name, which is derived from its public key, is defined by two special mandatory fields, *issuerAltName* and *subjectAltName*. The X509 name fields may optionally be used as attributes, if the principals are provided with a DN.

Validity: Every certificate has an expiration date as well as a serial number that is unique to the issuer. This serial number is used to indicate revocation - if it is included in the Issuer's revocation list (CRL). The expiration date and serial number are both provided by the X.509 format.

Certificates Management: Every TE module maintains a local library of certificates and CRLs which it had collected.

Revocation List: The TE System uses the existing X.509 mechanisms for certificate revocation. Specifically, every TE module has the capability of producing its own CRL, which is a certificate that contains a list of all revoked certificates it had issued in the past, along with the revocation Reason. We use the X509 CRL format to produce the CRL of a TE module. This CRL is valid for a given period of time.

Certificate Validity check: Validity of a certificates is determined by expiration date, which is embedded in the certificate itself, and by inspecting the CRL of its Issuer. The CRL of the certificate Issuer either resides locally (if still valid) or is requested on-line by approaching the Issuer's URL (recall that *issuerURL* is one of the mandatory fields of the certificate).

4.2 The Policy Language and the Policy Engine

The policy language is defined in XML and can be viewed/edited either through a text editor or graphically. We have developed a graphic editor that reads the XML policy and displays it as a graph were nodes are groups and edges are rules defining relations between groups.

The core of the TrustEstablishment module is the *Policy Engine* which is used to decide if an entity X (given by its public key) can be mapped to a given policy group. The current implementation uses a very simple, but inefficient, policy engine. An efficient algorithm for the policy engine will be presented in another work.

4.3 The Certificate Collector and Repository

The certificate collector is responsible for collecting missing certificates from certificate repositories. The collector holds a local database of collected certificates, and has the capability to crawl the network and retrieve certificates from remote certificate repositories. The certificate collector is operated particularly when the role assignment module tries to map X to role R and the checked rule prescribes that X can be mapped to role R if exist a certificate where the issuer is in group G. Assume that a certificate exist with issuer Y but Y is not known locally to be in group G. The Trust Establishment module requests the collector to find certificates about Y (where Y is the subject) that can prove that Y is in G. If such certificates exist in the collector's local DB then it can return them immediately to the caller. Otherwise it has to request it from a remote certificate repository. As described in section 2.2, one of the fields in each certificate is the *issuerCertRepository,* which holds the address of the issuer's server. The collector will contact this repository to ask for additional certificates. Similiarly, if looking for negative certificates, the collector will contact the repositories listed in the policy. In addition the collector can be configured to collect certificates from central repositories.

The architecture of the TE module and the collector is depicted in Fig 9. Every TE has a local collector to which it sends requests. It is only natural that a certificate repository may not be willing to give its certificates to everyone, instead it may contact an access control system (which may in turn contact a local Trust Establishment module) to decide. An example policy might be that a repository is willing to give all certificates to "IBM servers", and only certain certificates to all other servers. The repository can use a local access control and/or TE system to map remote collectors to roles according to its policy (Fig 9 - edge labeled "check remote collector"). Note that in order to implement that, the certificate repository must receive a certificate from the collector in the initiation of the request.

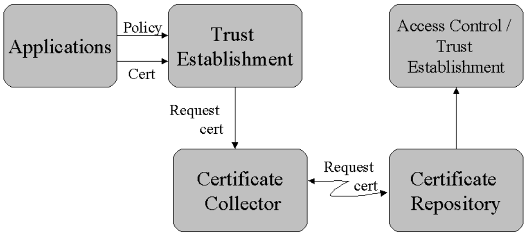

Figure 9: Certificate Collector and Repository

The collector has a local DB of collected certificates. It handles queries like:

- Get from local DB all certificates with *certType* X about a given *subject* Y.
- Request from repository (address given) all certificates with *certType* X about a given *subject* Y.
- Get an updated CRL of some issuer. If the CRL exist and is valid in local DB it is returned, otherwise the issuer (or its certificate repository) is contacted to supply an updated CRL.

4.4 Implementation of the Database

The policy related data is kept in a database where collected client certificates, their issuers, subjects, and other data is stored. The results from the reasoning process over the policy rules are also stored in the database. These results include the assignment of the certificate holders to particular groups. Typically, during the database design process we build conceptual model that represent as fully as possible the semantics of the particular application. The resultant DB structure is rather static and does not change considerably over time. In our case, however, we expect wide variety of different application areas and flexible policies for each one of them. This will lead to very different database structures depending on the particular case that have to be recreated in each particular case. In order to avoid this we decided to take into consideration during the DB design process only the application independent aspects of the policy related issues. This led us to rather static conceptual model, consisting of the following classes: issuers, subjects, entities, certificates, groups, memberships, and proofs. The collector manages the certificates in the DB while the TE component manages the groups, memberships and proofs. Here the entity class is generalization of the issuer and subject classes. The conceptual model is mapped into relational and LDAP data model. We have implemented the policy database using relational DBMS (DB2), and LDAP directory server.

4.5 Adding TE to a Web server

The TE component exposes APIs to applications that need to map entities to roles based on their certificates and a given policy. In this section we describe how we extended a web server to use the TE for mapping web users to roles.

The configuration (depicted in Fig 10 below) demonstrates how a web server can serve different pages to unknown users accessing it through a regular browser. The demo was run with apache server that supports SSL, and with Netscape browser. The servlet case was run under IBM WebSphere Application Server [21]/

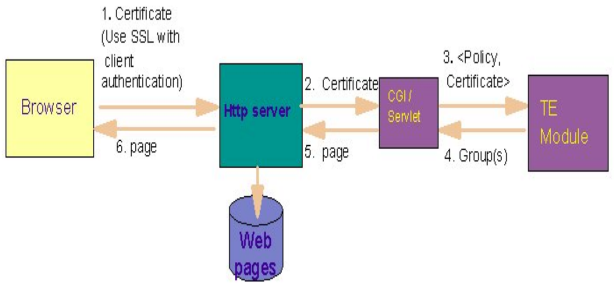

Fig 10 – Extending web server with the TE module

The sequence of requesting a page is described below:

1. The browser requests a page from the http server. The page is requested through SSL session [2] and the server is configured to use SSL client authentication. The http server asks the browser for a certificate and the browser displays to the user a screen to select which certificate to send. The user selects a certificate and it is sent to the http server for client authentication.

2. The http server runs a CGI/Servlet program and passes the client certificate to it.

3. The CGI/Servlet sends the certificate and a policy to the TE module for deciding on the role of this user.

4. The TE checks the policy and returns to the CGI/Servlet program the set of role(s) of that user.

5. The CGI/Servlet decides based on the user's roles, which page to display to the user and sends that page to the http server.

6. Page is sent to the user

Note that authentication is done by the application that uses the TE. The TE's job is to map the subject of a certificate to a role based on the certificate and policy, not to authenticate. It's the web server's who makes the authentication.

Implementation note: the TE can decide on user's role even if the client certificate's issuer is not known, however some Web servers would abort any SSL session where the issuer is not known. We overcame this problem by adding a patch to the Apache server, such that it will be willing to continue the SSL session and pass the client certificate to our CGI even if the issuer is not known.

Another solution can be to define a public key whose correspondent private key is known to everyone. This key will be added as a known CA to the web server and TE can be used to create a dummy certificate for that user with an extension *subjectCertRepository* that will direct the collector to collect more certificates about that user. The dummy certificate will pass SSL client authentication since the issuer is known and thus the http server will support the SSL session and will pass the certificate to the CGI.

5. Summary, conclusions and future work

We presented a mechanism that allows a business to define a policy to map accessed users to roles, based on certificates received from the user and collected automatically by the system. The policy language is expressed in XML and allows the system administrator to define flexible rules based on attributes in X509 certificates. The TE supports privacy (no need to know the user identity in order to map it to a role) and it is able to collect missing certificates from the web to reach a decision.

A possible work could be to develop a tool that checks the validity of the policy, checks that it has no loops and suggests alternatives to make the search on the policy tree more efficient. Another research direction could be to add inheritance to the policy language on groups (i.e. a group G' extends a group G by inheriting all the rules of group G and adding more rules to group G'). Another direction could be to define parts of the policy to be computed by another policy (e.g. a group G is computed by some other policy) which can be done for example in different departments in the same organization.

Other work could be done to improve the collector work by cooperation between collectors. For example, when a collector checks a path and needs to check that along the path some Y is in G', then it could ask another collector to check this and return a Y/N answer. This cooperation requires some synchronization in the groups in both collectors' policies and it could be done, for example, in different departments in the same organization. This approach could simplify the definition of the policy, and moreover, it could be used to keep the privacy of the proof such that one collector would know only that the proof exists, without knowing how the proof was made. Another extension could be the design of a new collector algorithm to decide on the order to look in different repositories and to improve the check if a certificate is revoked.

References

[1] Internet X509 Public Key Infrastructure documents, http://www.ietf.org/ids.by.wg/pkix.html

[2] SSL 3.0 Specification – The SSL Protocol Drafts http://home.netscape.com/eng/ssl3/index.html

[3] Extensible Markup Language (XML), W3C Specifications, http://www.w3.org/TR/WD-xml-lang.html

[4] M. Blaze, J. Feigenbaum, and J. Lacy, Decentralized Trust Management, In Proc. of the 17th Symposium on Security and Privacy, pp 164-173, 1996.

[5] Resource Access Control Facility (RACF) , V.2 http://www.s390.ibm.com/products/racf/

[6] Common Data Security Architecture (CDSA), http://developer.intel.com/ia1/security/documentation.htm

[7] Simple Public Key Infrastructure (SPKI), http://www.ietf.org/html.chapters/spki-chapter.html

[8] M. Winslett, N. Ching, V. Jones, and I. Slepchin, "Using Digital Credentials on the World-Wide Web," Journal of Computer Security, 1997

[9] M. Blaze, J. Feigenbaum, J. Ioannidis and A. Keromytis, The KeyNote Trust-Management System, http://www.cis.upenn.edu/~angelos/keynote.html

[10] Y.-H. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick and M. Strauss, REFEREE: Trust Management for Web Applications, in World Wide Web Journal, 2, pp. 127-139, 1997.

[11] D. F. Ferraiolo et el, Role based Access control (RBAC) Features and Motivations, NIST, U.S. Department of Commerce.

[12] Object Identifier, OID, ITU-T recommendation X.208 (ASN.1).

[13] K. E. Seamons, W. Winsborough, and M. Winslett, "Internet Credential Acceptance Policies", Proceedings of the Workshop on Logic Programming for Internet Applications, Leuven, Belgium, July 1997.

[14] H. Gladney, "Safe deals between strangers", IBM Research technical report (draft), August 1999.

[15] N. Li, B. N. Grosof, J. Feigenbaum, A Logic-based knowledge Representation for Authorization with Delegation, IBM Research Report, RC 21492, May 99.

[16] P. Zimmerman, “The Official PGP User’s Guide”, MIT Press, Cambridge, 1995.

[17] J. W. Lloyd, Foundations of Logic Programming, second edition, Springer, Berlin, 1987.

[18] IBM Haifa Research Lab, E-Business and Security Technologies group, http://www.hrl.il.ibm.com

[19] IBM AlphaWorks, http://www.alphaworks.ibm.com

[20] Policy Director, http://www.ibm.com/software/security/policy/

[21] IBM SecureWay Software, http://www.ibm.com/software/secureway/

Appendix A - the medical data policy file

We describe now the policy file for the medical data example in section 5.1 above. The tricky part are the two rules for becoming a recognized hospital which appear under the <GROUP NAME=”Hospitals”> line. The first rule states that a recognized hospital is every hospital that has a certificate issued by ‘self’ which means that the hospital is certified directly by the policy owner (self). The second rule is more complex and it states that a recognized hospital should have at least two certificates from already known hospitals and that there not exist a ‘warning’ certificate from any recognized hospital.

```
<?xml version="1.0"?>
<POLICY>

  <GROUP NAME="self">
  </GROUP>

  <!---->
  <!-- known hospitals -->
  <!---->

  <GROUP NAME="Hospitals">


     <!---- First rule : a hospital recommended by  ‘self’ with recommendation value greater then  1  --->

    <RULE>
      <INCLUSION ID="from_self" TYPE="Recommendation" FROM="self" ></INCLUSION>
      <FUNCTION>
        <GT>
          <FIELD ID="from_self" NAME="Recommendation"></FIELD>
          <CONST>1</CONST>
        </GT>
      </FUNCTION>
    </RULE>

     <!---- Second rule : a hospital recommended by  at least 2 hospitals, and there is no warning about it from any hospital  - -->

    <RULE>
       <INCLUSION ID="reco" TYPE="Recommendation" FROM="hospitals"  REPEAT=2></INCLUSION>
       <EXCLUSION ID="warn" TYPE="Warning" FROM="hospitals"></EXCLUSION>
       <FUNCTION>
         <AND>
          <GT>
            <FIELD ID="reco" NAME="Recommendation"></FIELD>
            <CONST>1</CONST>
          </GT>
          <GT>
            <FIELD ID="warn" NAME="Recommendation"></FIELD>
            <CONST>4</CONST>
          </GT>
         </AND>
       </FUNCTION>
     </RULE>
  </GROUP>

  <!---->
  <!-- A regular doctor - should have a certificate from some hospital -->
  <!---->

  <GROUP NAME="Doctors">
    <RULE>
      <INCLUSION ID="from_hospital" TYPE="doctor" FROM="Hospitals"></INCLUSION>
      <FUNCTION>
      </FUNCTION>
    </RULE>
  </GROUP>

  <!---->
  <!-- A cardiologist - should have a certificate from some hospital -->
  <!---->

  <GROUP NAME="Cardiologists">
    <RULE>
      <INCLUSION ID="from_hospital" TYPE="doctor" FROM="Hospitals"></INCLUSION>
      <FUNCTION>
        <EQ>
          <FIELD ID="from_hospital" NAME="Rank"></FIELD>
          <CONST>Cardiologist</CONST>
        </EQ>
      </FUNCTION>
    </RULE>
  </GROUP>
```

```
    <!---->
    <!-- An oncologist - should have a certificate from some
hospital -->
    <!---->

    <GROUP NAME="Oncologists">
        <RULE>
            <INCLUSION ID="from_hospital" TYPE="doctor"
FROM="Hospitals"></INCLUSION>
            <FUNCTION>
                <EQ>
                    <FIELD ID="from_hospital"
NAME="Rank"></FIELD>
                    <CONST>Oncologist</CONST>
                </EQ>
            </FUNCTION>
        </RULE>
    </GROUP>

</POLICY>
```