A Framework for Distributed Trust Management

Lalana Kagal, Scott Cost, Timothy Finin, Yun Peng
Computer Science and Electrical Engineering Department
University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21250
email : {lkagal1,rcost1,finin,ypeng}@cs.umbc.edu
phone : 410-455-3971
fax : 410-455-3969

Abstract

This paper discusses our infrastructure for handling distributed security and trust. It outlines a method for access control across domains that handles complex inter domain trust relationships. We have developed a flexible representation of trust information in Prolog, that can model permissions and delegations. We are currently working on modeling obligations, entitlements, and prohibitions as well. This paper describes a scheme for restricting re-delegation without using a specific delegation depth. Using examples, this paper explains the internal working of our system and the trust information that flows within it.

Keywords
Authorization, security, distributed trust, agents, X.509 certificates, knowledge representation, role based

1 Introduction

Authorization in a distributed systems is quite different from that in centralized systems. Various schemes for decentralized security have been suggested like Access Control Lists, Role based Access Control, PolicyMaker [1] etc. Though the above mentioned mechanisms are powerful, individually they are unable to meet the all requirements of trust management. Generally security systems should not only authenticate users, but also allow users to delegate their rights and beliefs to other users securely and have a flexible mechanism for this delegation. Most schemes either only support authentication, ignoring delegation altogether, or support delegation to some extent without providing the flexibility needed, or do not provide sufficient restrictions on delegation of rights.

We have tried to solve this problem through the application of a chain of trust using rights and delegations [5]. In our system, we model permissions as the rights of an agent. We associate rights with actions, so possession of a right permits the corresponding agent to perform a certain action. These permissions can be extended by delegation from an authorized agent.

This paper is organized as follows : 2 discusses other similar approaches, while 3 discusses the problem and the two scenarios we target. We then explain our infrastructure in brief in 4. We subsequently discuss the syntax used in the system including rights and delegations in 5. 6 expands on the architecture in greater detail giving a brief example, and section 7 discusses our ontology. 8 talks a little bit about the policy and how it is used. We describe the working of our system with examples in 9. 10 is a discussion of future research directions, and 11 contains the summary.

2 Related Work

Blaze, who coined the term Distributed Trust Management, tries to solve the problem by simple access control, but without any authentication or delegation. [1, 2]. The Simple Public Key Infrastructure (SPKI) was the first proposed standard for distributed trust management [4]. This solution, though simple and elegant, does not help in delegations. W.Johnston's Use-Condition Centered Approach [10] uses certificates for use-conditions that are created by those responsible for the resources. This approach can only be used when the resource is simple enough to be described by use-conditions, but in large systems there could be many types of access like read, write, execute etc. Another system we looked at was TE [11] from IBM. But we found that it was inadequate because it only considers role authorization. Delegation logics [12, 18] from IBM is very similar to our approach, but is not able to capture adequately the constraints associated with rights and delegations.

Although the above mentioned models are very powerful, individually they were unable to meet the all requirements of trust management. Generally security systems should not only authenticate users, but also allow them to delegate their rights and beliefs to other users securely. The systems mentioned above either support only authentication ignoring delegation altogether, or support delegation to some extent without providing the flexibility needed, or do not provide sufficient restrictions on the delegation of rights.

We drew on the key points of most of the above-mentioned schemes and put together an infrastructure that uses X.509 certificates and policies to enforce security. A policy contains basic or axiomatic rights, rights associated with roles, rules for delegation, and rules for checking validity of requests.

One of the most important features of our system is that it allows agents to delegate any right that they may have. Whether these delegations are honored depends on the policy. Constraints can be added to both the actual delegation and to the delegatee, tightening control on the rights and permissions, and we use a *redelegatable* flag that controls whether the right can be further delegated.

3 The Problem

We have tried to solve the problem of trust management in a system which has widely distributed resources and agents. There are two scenarios that we have tackled, a *Home/Office Automation* model, and an electronic *Supply Chain Management system* like EECOMS [35]. We have successfully implemented the EECOMS scenario and are currently working on the *SmartOffice* scenario with portable devices.

3.1 EECOMS : IBM project

This work was sponsored by the CIIMPLEX consortium [35] for the Extended Enterprise COalition for Integrated Collaborative Manufacturing Systems (EECOMS) project which is aimed at providing a set of technologies for integrated supply chain and business to business electronic commerce.

We implemented a distributed trust mechanism for a Supply Chain Management system for the CIIMPLEX EECOMS project [6]. A supply chain management system consists of groups of agents that are either vendors or clients. These agents need to access resources in each others' domains. For example, a software consultant may need to access the database of its client. Each group of agents that are part of the same company form a policy domain and follow the same security policy. The policy in each domain is enforced by special agents called *security agents*. Agents are identified by an ID certificate, which is an X.509 certificate. All communication in the system is via signed messages. *Security agents* are able to reason about these signed messages and policies to provide authorization.

Agents can make requests, either for certain actions or to ask for permission to perform some action, and they attach all their credentials, i.e. ID certificate, authorization certificates etc. to these requests. The *security agents* generate authorization certificates, which can be used as 'tickets' to access a certain resource. The authorization certificates are generated as the result of a request for permission, if the request is valid. Policies consist of rules about authorization, delegation and some basic knowledge about the agents. This knowledge could be about the role of the agents, and permissions associated with the agents or the roles. An agent is allowed to execute any action that it has the permission to execute, or if the ability has been delegated to it by an agent with the right to delegate. In our system we view 'delegation' as a permission itself. Only an agent with the right to delegate a certain action can actually delegate the action, and the ability to delegate itself can be delegated.

We have developed a representation of trust information in Prolog, that allows flexibility in describing requests and delegations. Delegations can be constrained by specifying whether the delegatee has the permission to delegate and to whom it can redelegate.

This system is currently being extended by the addition of entitlements, prohibitions and obligations and the ability to delegate them.

3.2 Home/Office Automation

Our architecture could apply to the wireless world in the following scenario. If a visiting lecturer at a University needs to use a projector in a lecture hall, she/he needs to be delegated the right by some authorized personnel. If the policy states that all professors can use the projector and that professors can delegate this right to the lecturer, the lecturer can obtain the 'token' from a professor. Using a hand-held device such as a PDA, mobile phone etc. the visitor beams her/his identifying token to the projector along with the delegation token. The projector may or may not have the processing power to reason about these certificates and rights. If it does not have the capability, the agent in the projector sends the token (using wireless or wire line communication) to a 'smart' agent that evaluates the request and returns the result. The agent that does the reasoning needs to check the identity of the requester and then make sure that the requester has the right to access the projector. In this case, the requester has been delegated the right by a professor, so the agent should verify that the professor has the right to delegate. Once the request is validated, the visitor can beam her/his slides to the projector agent that starts up the presentation.

We have started experimenting with Bluetooth [34] and believe that the above scenario is not too far in the future [25, 20].

4 Infrastructure

Our architecture assumes that each group of agents is protected by special *security agents* that trust each other. These agents are responsible for authorizing access to services/resources within that group. All delegations are stored by the security agent, which has the ability to reason about them. An agent (requester) can execute a right or access a resource by providing its identity and/or authorization information to the security agent. The security agent checks this information for validity, and reads its policies to verify that the requester has the right. If the requesting agent does not have the right, the security agent returns an error message, otherwise it forwards the request to the agent in charge of the resource, accessor agent, along with a message saying that the request is authorized by the security agent. As the security agent is trusted by every other agent in the system, the requesting agent is granted access. If the accessor agent has the computing power to reason about certificates, rights and delegations the request can be sent directly to it, instead of via the security agent.

The requester can also obtain access to a certain resource that it previously could not access, through a delegation from an authorized agent (delegator). An authorized delegator (an agent with the right to delegate a certain right) delegates the right by sending a message to the security agent. The delegation has to be approved by the security agent and should conform to its policies. The requester approaches the security agent with its identity information and a request for permission to access the right. The security agent verifies the identity of the requester and checks with its policies to make sure that the requester can be given access to the resource. The new delegation makes the request valid. The security agent generates a *authorization certificate* which contains a Prolog [36] like statement giving the requester permission to access the resource. This message is sent to the requesting agent. These statements are dated and are valid only for a certain period. While the statements are valid, the requesting agent can use them as *tickets* to access the resource. This causes the entire process of verification and reasoning to be skipped, and the requester gets access to the resource as soon as the authorizing statement is recognized and verified by the accessor agent.

5 Syntax

We use a number of predicates to represent the information flowing in the system.

5.1 Rights

Our system encodes rights into a logical form in Prolog as the following

```
rightToDo(agentName, Action, Constraint)
```

- agentName : uri for the agent
- Action : representation of the ability i.e. accessDB(db5)
- Constraint : restriction on the right, i.e. employee(agentName,XYZ)

Using this statement, all kinds of rights can be specified. An agent is given the right to perform a certain action based on a constraint. An agent can execute the action only if is satisfies all the constraints.

5.2 Delegation

An agent can execute any right, that is either an axiomatic right specified in the policy or that has been delegated to it. It can also delegate this right to other agents, if it has been given the right to subsequently delegate it. A delegation itself is a right which can be delegated. So, an agent could be given the right to perform some action but not to further delegate it or given the right to some action and the right to delegate it, or the right to delegate some action but not the right to execute it.

So, an agent can delegate any 'delegatable' right. This leads to a chain of delegation, and if any one link is no longer valid the access is denied. We also allow for constraints on rights, delegations and ability to re-delegate.

One of the main features in our system is that false delegations are not rejected as soon as they enter the system, but are stored for later evaluation of a possible security breach. An agent has the ability to make any delegation, but whether it is honored depends on various factors, including the security policy, the agent's rights, and the rights of the agents in the delegation chain.

The statement that is used to describe delegations and constraints on delegations is

```
delegate(IssueTime, StartTime, EndTime, From, To,
    canDo(X, Action, CDC), IDC, Redelegatable)
```

- IssueTime : when the statement was issued
- StartTime : when the delegation becomes valid
- EndTime : when the delegation becomes invalid
- From : delegator agent
- To : delegatee agent
- canDo(X,Action,Action, CDC) : delegated action, X has the right to the action, only if X satisfies the condition CDC
- IDC : condition on the delegation
- Redelegatable : true if the To can re-delegate the action

5.2.1 Types of delegations

Our work in the EECOMS scenario involved several different types of delegations which we describe here and give simple examples.

- Time Bound Delegation : This is a delegation that is valid only for a certain time period

  ```
  delegate(1105001120,1105001121, 1110001120, From, X,
      canDo(Y, Action, CDC), employee(X,abc), Flag)
  ```

  The delegation is only valid between 1105001121 and 1110001120.

- Group Delegation : This can be used to delegate to agents from a group who satisfy certain conditions

  ```
  delegate(IssueTime, StartTime, EndTime, From, X,
      canDo(Y, Action, CDC), (employee(X,abc),age(X,24)), Flag)
  ```

  This delegates to all employees of abc who are 24 years old, the ability to perform a certain action.

- Action Restricted Delegation : This is a delegation that requires the delegatee to satisfy certain conditions before the action can be carried out.

  ```
  delegate(IssueTime, StartTime, EndTime, From, X,
      canDo(Y, Action, name(Y,john) ), (employee(X,abc),age(X,24)), Flag)
  ```

Only employees of abc who are 24 and named John can execute this action, even if all employees aged 24 have been delegated the right.

- Redelegatable Delegation : In this delegation, a right can be delegated along with the permission to re-delegate the right.

  ```
  delegate(IssueTime, StartTime, EndTime, From, To,
      canDo(X, Action, CDC), IDC, true)
  ```

  This statement allows the recipient to further delegate the right.

- Strictly Redelegatable Delegation : This statement allows a right to be re-delegated without giving the delegatee the right to actually execute the action.

  ```
  delegate(IssueTime, StartTime, EndTime, From, john,
      canDo(Y, Action, notname(Y,john)), IDC, true)
  ```

  John is given the right to further delegate the Action, but John cannot execute the action himself.

6 Implementation

We have applied our approach to a electronic commerce multi-agent system, EECOMS, for evaluation. We use X.509 certificates created by a Certificate Authority (CA) as identity certificates and X.509 certificates containing signed Prolog statements as authorizing statements.

When an agent needs to access a resource, it contacts its security agent with a Signed Message Object, SMO, (refer to Data Structure) containing its ID certificate and the request. If the security agent is convinced that the requester has the right and the request is within the company, it is allowed to go through. If the resource belongs to another company, the security agent creates an *authorization certificate* and returns it to the requester. The requester sends a request to the other company's security agent, along with its ID certificate and its authorization certificate. The resource's security agent verifies all the certificates and as security agents are trusted across companies, allows the request to go through. This authorization certificate can be used for a certain time period after which it expires. Then the agent has to request a new authorization certificate. By having very short time periods, *revocations can be handled* .

If an agent wants to delegate a certain right, it sends a Signed Message Object containing the delegation and its certificate to the security agent. As delegations can be made to a group of agents, it is not possible to send the delegation to each agent. When one of these agents, for example agent1, needs to use the right, it approaches the security agent with a Signed Message Object. The security agent knows that agent1 has been delegated the right, so it is authorized. The security agent returns a *delegation certificate* containing an authorization statement in Prolog, giving the agent with the corresponding id certificate the right. If the resource belongs to another company, the requester sends the request with the delegation and ID certificates to the security agent of the other company. The security agent of the other company verifies the certificates and checks the delegation chain. If it is satisfied, the access is allowed to go through and the request is forwarded to the agent responsible for the resource. Similar to authorization certificates, delegation certificates are valid for a short period, after which the agent has to get a new certificate.

6.1 Data Structure

Each agent communication is a series of objects, based on a simple Agent Communication Language (ACL) [23, 38], known as Signed Message Objects (SMO). Each SMO consists of a list of certificates, the request or the authorization statement signed with the senders private key, and other required fields. The relevant certificates are included as part of the communication data structure to expedite the authorization process; message, signed message. Each SMO should contain the ID certificate used to sign the message. The message field contains Prolog statements, which can be either requests or delegations. The signed message field, as its name suggests, contains the signed version of the Prolog statement. The requesting agent sends long with his request, other credentials that will strengthen his request.

To verify the validity of a Signed Message Object the following items are checked : The certificate that has been used to sign the message should to be issued by a trusted Certificate Authority and should be still active. All other certificates are verified as well. After this, the signed message is matched with the clear text message to make sure that the message was indeed signed by the attached certificate. The SMO is also time bound, so that has to be checked as well.

7 Ontology

This approach uses a simple ontology of agents, propositions and actions which we describe briefly.

- Agents : An agent is an entity is the system. This could be a program or a human.
- Propositions : We use 2 propositions, *ability* and *delegate*.
  - Ability is a property that an agent possesses. An agent can perform an action, if it has the ability/right to do so.

    ```
    canDo(<agent>,<action>,<constraintsOnAction>)
    ```
  - Delegate is a proposition asserted into a database which indicates that one agent has delegated to another agent, the right to perform some action.

    ```
    delegate(<issueTime>,<startTime>,<endTime>,<fromAgent>,
    <toAgent>,<ability>,<constraintOnDelegation>,<redelegateFlag>)
    ```
- Action is what an agent can perform and is closely linked to abilities.
  - Idelegate is the action of delegating the ability to perform the action from one agent to another.

    ```
    idelegate(<startTime>,<endTime>,<fromAgent>,<toAgent>,
    <ability>,<constraintOnDelegation>,<redelegateFlag>)
    ```
  - An agent can tell another agent a proposition that it believes is true

    ```
    tell(<fromAgent>,<toAgent>,<proposition>)
    ```
  - An agent can ask a security agent if he has the right to perform the action

    ```
    ask(<fromAgent>,<toAgent>,<action>)
    ```
  - An agent requests a security agent to perform some action on his behalf. The security agent will perform this action only if the agent has the ability.

    ```
    request(<fromAgent>,<action>)
    ```

8 Policy

Each domain has a policy associated with it. This policy consists of authorization and delegation policies. Authorization policies deal with the rules for checking the validity of requests for actions. An example of a rule for authorization would be checking the identity certificate of an agent and verifying that the agent has an axiomatic right. Delegation policies describe rules for delegation of rights. A rule for delegation would be checking that an agent has the ability to delegate before allowing the delegation to be approved. A policy also contains basic or axiomatic rights, and rights associated with roles. We introduce the concept of primitive or axiomatic rights, which are rights that all individuals possess and that are stored in the global policy. For example, freedom of speech is a constituitional right, and anyone who owns a database has the right to delegate the right to read from/write to that database. These are basic rights that are not often expressed, but used implicitly. All policies are described in Prolog. A policy can be viewed as a set of rules for a particular domain that defines what permissions a user has and what permissions she/he can obtain.

Users of the system are generally assigned roles. A role is defined as a collection of rights and duties. Roles are arranged in a hierarchy, so that rights can be inherited. An agent has a right if it is mentioned in the policy or if the right has been delegated to it by another agent that has the ability to delegate. Delegations generally flow downwards in the role hierarchy, and are from a higher role to a lower role. However our framework does not strictly adhere to role based access, and allows rights and delegations to be assigned to individuals and groups.

9 Operation of the System

We now describe the internal working of the system along with the trust information that flows within the system. *Security agents* reason about the policy and the signed messages from other agent to provide trust management. The security agent maintains the company policy in a Prolog [36] knowledge base. When it receives an SMO, it inserts the Prolog statement into the knowledge base for verification. If the message is a 'tell' it is only asserted and no reasoning is carried out. This allows a log to be kept, and includes false delegations as well. When a message contains a 'request' either for authorization or action, it triggers the associated Prolog rule. This rule makes sure that the request is valid by checking the delegation chain and the company policy.

All the reasoning about rights and delegations is handled by a set of Prolog rules causing incorrect delegations and statements to be trapped by Prolog's backward chaining mechanism and prevented from going through. We have rules that cause constraints on rights to be propagated when a delegation occurs.

We now describe a particular situation and then explain how this is resolved in our implementation.

9.1 Example Situation

Let us assume that there are two organizations, ABC and XYZ, that are collaborating on a certain project. If Marty, a design engineer at ABC, wants to access the database of the client, XYZ, and if Marty has the permission to do so, he sends a *Request for Action* (refer to next subsection) to his own security agent. The security agent returns an authorization certificate, that Marty uses to access the database.

We also assume that Marty has the permission to access the database and that this permission can be delegated. Marty wants all programmers to access the database as well, and so he sends a certificate containing a delegate statement to security agent.

Now if Harry, a programmer working with ABC, needs to access a database of his client, XYZ, it constitutes a *Request for Permission*. When Harry makes a request for permission, the security agent, checks Marty's delegation and then creates a delegation certificate and returns it. Using this delegation certificate, Harry can gain access to XYZ's database.

9.2 Request for Action

We illustrate the working of Request for Action by using the above example. ABC and XYZ are two companies represented by their security agents, SA-ABC and SA-XYZ. XYZ is the client of ABC. Marty is a design engineer in ABC, where design engineers can access their client's database, db5. The following steps show how the authorization actually takes place.

1. SA-XYZ loads the company policy for XYZ and loads a global shared policy
2. SA-ABC loads the company policy for ABC and loads a global shared policy
3. SA-XYZ sends message to SA-ABC saying that SA-ABC has the right to delegate access to db5, which is a database in XYZ, to all employees.

   ```
   tell(sa-xyz, sa-abc,
       idelegate(StartTime, EndTime, sa-xyz, sa-abc,
           canDo(X,accessDB(db5), employee(X,abc)),
           true,true).
   ```

   SA-ABC asserts the proposition

   ```
   delegate(IssueTime, StartTime, EndTime, sa-xyz, sa-abc,
       canDo(X,accessDB(db5),employee(X,abc)),true,true).
   ```

   SA-ABC gives all Design Engineers the right to access db5, but not the ability to delegate

   ```
   tell(sa-abc,sa-abc,
       idelegate(StartTime, EndTime, sa-abc, X,
       canDo(X, accessDB(db5),true), role(X,designEngineer),false).
   ```

This causes a delegate statement to be inserted into the knowledge base.

```
delegate(IssueTime, StartTime, EndTime, sa-abc, X,
    canDo(Z, accessDB(db5),true), role(X,designEngineer),false)
```

4. Marty requires some information from database, db5, at XYZ. He sends a request to SA-ABC along with his certificate

   ```
   request(marty,accessDB(db5)).
   ```

5. SA-ABC knows that the request is from Marty because of his certificate. It then checks the rules to see if Marty as a Design Engineer has access to db5. As this is true, SA-ABC creates an *authorization certificate* and sends it back to Marty.

6. Marty sends a request to SA-XYZ with his ID certificate and the authorization certificate.

   ```
   request(marty,accessDB(db5)).
   ```

7. SA-XYZ verifies both the certificates and checks its policy. SA-XYZ approves the access and the request is sent to the agent controlling access to the database.

8. If Harry, a programmer tries to access the database, db5, his request will fail because the SA-ABC has only given design engineers the right.

9.2.1 Request for Permission

The sequence of actions below describes the delegation process in detail, where Harry, a programmer in ABC, requires some information from the db5 database at XYZ.

1. SA-XYZ loads the company policy for XYZ and loads the global shared policy

2. SA-ABC loads the company policy for ABC and loads the global shared policy

3. SA-XYZ sends message to SA-ABC saying that SA-ABC has the right to delegate access to db5, which is a database in XYZ, to all employees.

   ```
   tell(sa-xyz,sa-abc,
        idelegate(StartTime, EndTime, sa-xyz, sa-abc,
        canDo(X,accessDB(db5), employee(X,abc)), true,true).
   ```

   SA-ABC asserts the proposition into its own knowledge base

   ```
   delegate(IssueTime, StartTime, EndTime, sa-xyz, sa-abc,
       canDo(X,accessDB(db5),employee(X,abc)),true,true).
   ```

   Then, SA-ABC decides to give all Design Engineers the right to access the database and the right to delegate this right further

   ```
   tell(sa-abc,sa-abc,
       idelegate(StartTime, EndTime, sa-abc, X,
       canDo(Z, accessDB(db5),true), role(X,designEngineer),true).
   ```

   This causes a delegate statement to be inserted

   ```
   delegate(IssueTime, StartTime, EndTime, sa-abc, X,
        canDo(Z, accessDB(db5),true), role(X,designEngineer),true)
   ```

4. Marty is a design engineer and he gives all programmers the right to access db5.

   ```
   tell(marty,sa-abc,
       idelegate(StartTime, EndTime, marty,X,
       canDo(X,accessDB(db5),true), role(X,programmer),false).
   ```

   SA-ABC asserts the following clause

   ```
   delegate(IssueTime,StartTime,EndTime, marty,X,
       canDo(X,accessDB(db5),true), role(X,programmer),false).
   ```

5. Harry, a programmer, requires some information from database, db5, at XYZ. He sends a request to SA-ABC along with his certificate

   ```
   request(harry,accessDB(db5)).
   ```

6. SA-ABC knows that the request is from Harry because of his certificate. It then checks to see if Harry has access to db5. Marty has given all programmers access to db5, and Harry is a programmer, so Harry has the right. SA-ABC creates a *delegation certificate* and returns it to Harry.

7. Harry sends a request to SA-XYZ along with his ID certificate and his delegation certificate

   ```
   request(harry,accessDB(db5)).
   ```

8. SA-XYZ verifies Harry's id. As SA-XYZ trusts SA-ABC and the delegation certificate is still valid, the request is approved and sent to the access agent controlling the database. If required, SA-XYZ, can double check the delegation chain and ask SA-ABC for more information.

9.3 Implementation Details

All the agents are Java Servlets that communicate using HTTP. The knowledge base is Prolog [36] and the reasoning is carried by rules written in Prolog. We also have a Java applet that can be used to generate SMOs. The security agents have a Jasper [37] interface to Prolog.

10 Future Work

For our implementation, we had concentrated on Prolog. We are now experimenting with XML [28] based languages like RDF [26, 27] and DAML [22, 24] for describing rights, delegations and authorizations, and XML signatures [29] instead of X.509 certificates. We are also working on other issues related to Distributed Trust Management. If an agent is able to access certain public policies of the agent which is in charge of authorization, then it will be in a better position to fulfill those requirements. This leads to the problem of dividing the policy into private and public and the making the public policy available. We are still deciding whether the policy should be made downloadable through HTTP or sent to any agents that request it. Another possible improvement would be for a security agent to return a list of rules that it used to come to the decision, in case the authorization process fails. This allows the requester to figure out where its credentials failed and correct the faults.

11 Conclusion

The central idea of the paper is to use a system of rights and delegations along with certificates to facilitate trust management. The requester can access a foreign resource by providing it's identity information along with any delegations it may have to the *security agent* of the domain. The security agent uses its policies to verify the identity and delegations of the requester, granting him permission only if everything is valid. We were able to evaluate our infrastructure and rule based language by implementing a multi-agent scenario, EECOMS. We have described our implementation for distributed access control and trust management which we believe will be very useful in the pervasive computing environment.

References

[1] M.Blaze, J.Feigenbaum, J.Lacy. ”Decentralized Trust Management”, IEEE Proceedings of the 17th Symposium on Security and Privacy, 1996

[2] M.Blaze, J.Feigenbaum, M.Stauss. ”Compliance Checking in the Policy Maker Trust Management System”, Proceedings of Financial Crypto’98, Lecture Notes in Computer Sciences vol.1465, Springer Berlin, 1998

[3] M.Blaze, J.Feigenbaum, J.Joannidis, A.Keromytis. ”The KeyNote Trust Management System”
http://www.cis.upenn.edu/ãngelos/Papers/draft-keynote.txt

[4] Ellison, M.Carl, et al, ”SPKI Certificate Theory”, RFC 2693, Internet Society, 1999

[5] J.J.Huang, B.M.Shao, P.C.Wang, ”A new access control method using prime factorization”, The Computer Journal, vol 35, No.1, 1992

[6] J.K.Jan, C.C.Chang, S.J.Wang, ”A dynamic key-lock-pair access control scheme”, Computer & Security, Vol.10, 1991

[7] M.S.Hwang, W.G.Tzeng, W.P.Yang, ”A two-key-lock-pair access control method using prime factorization and time stamp”, IEICE Transactions Inf. & Syst, Vol.E77-D, No.9, 1994

[8] M.Blaze, J. Feigenbaum, J.Ioannidis, K. Keromytis, ”The Role of Trust Management in Distributed System Security”, Secure Internet Programming, J.Vitec, and C.Jensen (Eds)., 1999

[9] R.Sandhu, E.J.Coyne, H.L.Feinstein, C.E.Youman, ”Role-based access control methods”, IEEE Comput., Vol 29, No.2(Feb), 1996

[10] W.Johnston, C.Larsen. ”A use-condition centered approach to authenticated global capabilities: security architectures for large-scale distributed collaboratory environments”
http://www-itg.1bl.gov/Security/Arch/publications.html

[11] A.Herzberg, Y.Mass, J.Michaeli, D.Naor, Y.Ravid, ”Access control meets public key infrastructure, or: assigning roles to strangers”
http://www.hrl.il.ibm.com/TrustEstablishment/paper.htm

[12] Ninghui Li, Benjamin N. Grosof, Joan Feigenbaum, ”A Logic-based Knowledge Representation for Authorization with Delegation”, IBM Research Report, May 1999
http://research.ibm.com/

[13] M.Abadi, M.Burrows, B.Lampson, ”A Calculus for Access Control in Distributed Systems” ACM Transactions on Programming Languages and Systems, Sep 1993

[14] Tomas Aura ”On the Structure of Delegation Networks”, 11th IEEE Computer Security Foundations Workshop, IEEE Computer Society Press, June 1998

[15] FIPA 97 Specification Part 2, Version 2.0, Agent Communication Language, 1998

[16] FIPA ACL Message Representation in XML Specification, 2000

[17] Ford, Warwick and Baum, ”Secure Electronic Commerce : Building the Infrastructure for Digital Signatures and Encryption”, Prentice hall, 1997

[18] Benjamin N. Grosof, Yannis Labrou, ”An Approach to using XML and Rule-based Content Language with an Agent Communication Language”, IBM Research Report, RC 21491(96965) May 1999

[19] Ninghui Li, Benjamin N. Grosof, Joan Feigenbaum, ”A Logic-based Knowledge Representation for Authorization with Delegation”, IBM Research Report, May 1999
http://research.ibm.com/

[20] Chen, H., Joshi, Anupam, Finin, T., and Chakraborty, D., ”Dynamic Service Discovery for Mobile Computing: Intelligent Agents meet Jini in the Aether”, to appear, Baltzer Science Journal on Cluster Computing, Special Issue on Advances in Distributed and Mobile Systems and Communications, 2001

[21] Chen, H., Chakraborty, D., Xu, L., Joshi, Anupam, and Finin, T., Service Discovery in the Future Electronic Market, Proc. Workshop on Knowledge Based Electronic Markets, AAAI2000, Austin, 2000

[22] DAML specification, **http://www.daml.org/, October 2000**

[23] Yannis Labrou, Tim Finin, Benjamin Grosof and Yun Peng, ”Agent Communication Languages, in *Handbook of Agent Technology*, Jeff Bradshaw, ed., MIT/AAAI Press, 2000

[24] Jim Rapoza, ”DAML could take search to a new level”, PC Week Labs, February 7, 2000 **http://www.zdnet.com/eweek/stories/general/0,11011,2432538,00.html**

[25] Lalana Kagal, Vlad Korolev, Harry Chen, Anupam Joshi, Timothy Finin, ”Centaurus : A Framework for Indoor Mobile Services”, September 2000

[26] Ora Lassila (Nokia Research Center) and Ralph R. Swick (World Wide Web Consortium), Resource Description Framework (RDF) Model and Syntax Specification, February 1999, **http://www.w3.org/TR/REC-rdf-syntax/**

[27] Resource Description Framework (RDF) Schema Specification, W3C Proposed Recommendation, March 1999, **http://www.w3.org/TR/1998/WD-rdf-schema/**

[28] Extensible Markup Language (XML) 1.0 (Second Edition) W3C Recommendation 6 October 2000 **http://www.w3.org/TR/2000/REC-xml-20001006**

[29] XML-Signature Syntax and Processing, W3C Candidate Recommendation 31-October-2000, **http://www.w3.org/TR/xmldsig-core/**

[30] Ninghui Li, Benjamin N. Grosof, and Joan Feigenbaum ”A Practically Implementable and Tractable Delegation Logic”, In: Proc. of IEEE Symp. on Security and Privacy, held Oakland, CA, USA, May 2000.

[31] Ninghui Li, Delegation Logic: A Logic-based Approach to Distributed Authorization, PhD Dissertation, New York University, NYU. **http://cs1.cs.nyu.edu/ninghui/thesis.htmli**

[32] Rivest, Ron, Lampson, ”SDSI - A Simple Distributed Security Architecture” **http://theory.lcs.mit.edu/cis/sdsi.html**

[33] H.C. Wong, K. Sycara ”Adding Security and Trust to Multi-Agent Systems”, Proceedings of Autonomous Agents’99 (Workshop on Deception, Fraud and Trust in Agent Societies), 1999

[34] The Official BlueTooth website **http://www.bluetooth.com/**

[35] CIIMPLEX Consortium, Consortium for Integrated Intelligent Manufacturing PLanning and EXecution **http://www.ciimplex.org**

[36] Swedish Institute of Computer Science, SICStus Prolog **http://www.sics.se/sicstus/**

[37] Jasper **http://www.sics.se/isl/sicstus/jasper/**

[38] M.Woolridge ”Semantic Issues in the Verification of Agent Communication Languages”, Autonomous Agents and Multi-Agent Systems, 2000

[39] H.C. Wong, K. Sycara ”Adding Security and Trust to Multi-Agent Systems”, Proceedings of Autonomous Agents’99 (Workshop on Deception, Fraud and Trust in Agent Societies), 1999