Towards A Secure Agent Society

Qi He
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A.
qihe@cs.cmu.edu

Katia Sycara
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A
katia@cs.cmu.edu

March 23, 1998

Abstract

We present a general view of what a "secure agent society" should be and how to develop it rather than focus on any specific details or particular agent-based application. We believe that the main effort to achieve security in agent societies consists of the following three aspects:1) agent authentication mechanisms that form the secure society's foundation, 2) a security architecture design within an agent that enables security policy making, security protocol generation and security operation execution, and 3) the extension of agent communication languages for agent secure communication and trust management. In this paper, all of the three main aspects are systematically discussed for agent security based on an overall understanding of modern cryptographic technology. One purpose of the paper is to give some answers to those questions resulting from absence of a complete picture.

Area: Software Agents

Keywords: security, agent architecture, agent-based public key infrastructure (PKI), public key cryptosystem (PKCS), confidentiality, authentication, integrity, nonrepudiation.

1 Introduction

If you are going to design and develop a software agent-based real application system for electronic commerce, you would immediately learn that there exists no such secure communication between agents, which is assumed by most agent model designers. In fact, software agents, as primarily human-delegated software entities, would face almost all the risk and security threats, which human being have to face[1],especially in commercial activities.

1.1 Security mechanisms

Security risks are various. In addition to providing confidentiality for data communication between agents, it is also very important for agent designers and developers to take the following three basic security mechanisms into account:

1. *Authentication*: the mechanism that makes it possible for an agent to ascertain an origin of a received message; with the mechanism, an intruder should not be able to masquerade as someone else. For example, for a site or a node on the Internet, this mechanism makes it possible to verify the identification of a visiting mobile agent.

2. *Integrity*: the mechanism that makes it possible for agents to verify that a received message has not been modified in transit; with the mechanism, an intruder should not be able to substitute a false message for a legitimate one. The mechanism also makes it possible for a site or node on the Internet to verify that for instance, a legitimate mobile agent has not be modified in transit.

3. *Nonrepudiation*: the mechanism that makes an agent who sent a message not be able to falsely deny later that it sent the message. The mechanism may also make it impossible for anyone else to replay the message later.

To construct and use these security mechanisms itself is very crucial part in designing/planning procedure of commercial activities for agents.

1.2 Agent and Modern Cryptography

The ground breaking development of modern cryptography has been meeting the demand of security of various applications over open network: Conceptually, the openness of modern cryptosystems, both symmetrical cryptosystem such as DES[2], IDEA[3], and asymmetric cryptosystem, such as RSA[4], lays a shared foundation of security operations of agents across the Internet. Technically, some security standard software packages and APIs are under development, and they could be available in very near future[6][7]. Those achievements, like the achievements in AI, are ripe to be adopted in development of agent applications. Agents, as primarily human-delegated software, could become the primary area for us to practice modern cryptographic technology because:

1. Software agents are playing increasingly active and bigger role in electronic commerce. To be adopted into soft agent development would be the direction for modern cryptography to fully demonstrate its potential significant function in electronic commerce.

2. Execution of most security protocols would be big burden for end-user, however, it could be done fairly easily by autonomous software, agents.

RETSINA[8], an intelligent multi-agent project at Carnegie Mellon University, is developing a general architecture of agents which can communicate with agent communication language, such as KQML[9] and work together cooperatively by applying DAI theory and technology. The multi-agent system is expected to be employed in various agent-based applications over the Internet, particularly in electronic commerce. Because of the importance and feasibility of security mentioned above, security mechanisms and functionality are considered as one of the basic features in our design[10]. At same time, we recognize that to achieve agent security, it is necessary to establish trust relationship among agent societies and an *agent-based public key infrastructure* is under construction[11].

In this paper, we systematically and comprehensively introduce our methodology and design of trust establishment, internal security functional module structure of agent, as well as an extension of agent communication language to achieve security of agent and agent society for general agent-based applications.

The remainder of the paper is organized as follows: Section 2, will introduce our work on establishment of trust for security agent societies, which will lay, in a bottom up fashion, a authentication foundation for agent security. This is security infrastructure for secure environment of agent. In section 3, we will introduce the basic methodology, a three-level partition for agent security involving security policy making, security protocol generation and security operation execution, as well as the security architecture within an agent. In section 4, we will introduce an extension of agent communication language for agent secure communication and trust management. In section 5, we will discuss some limitations and open problems.

Since our work, unlike other papers[12][13], focuses on general reusable software agent architecture,we are not going to focus on any specific application project or on a particular aspect.

2 Agent Trust Infrastructure

Authentication, integrity, nonrepudiation are fundamental parts of modern cryptography. We use those mechanisms through out our daily life, when we sign our name to some document for instance, and as we move to a world where our decisions and agreements are communicated electronically, we need to replicate these procedures [1]. A great advantage of public key cryptography is that it provides mechanisms for such procedures in very efficient way[14].

2.1 PKI: Public Key Infrastructure

Whenever we, however, use a public key to encrypt a message or to verify the authenticity (digital signature) of a message, we must ensure that the public key we are using is valid and it belongs to the claimant rather than anyone else. This issue known as the *public key integrity problem* vitally determines the whole security of communication, including conducting transactions over the Internet.

The current state-of-the art solution is to establish in a hierarchical manner a system to issue public key certificates, in which the principal's public key (probably as well as some other information) is included and signed by an authority, and the authority may hold a certificate issued by a super authority, and so on up the hierarchy. This system is the so called public key

[1] Obviously, if we expect to delegate agents to do job for us, we must incorporate the corresponding security mechanisms into agent design and development.

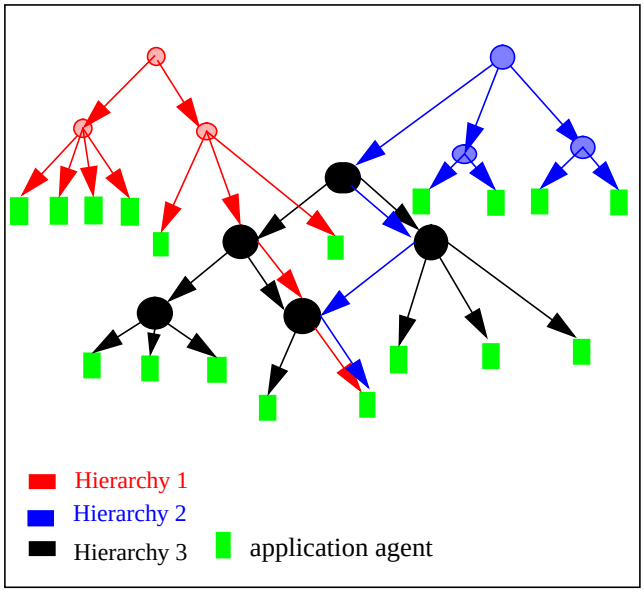

Figure 2.1 Multiple Hierarchies across a agent.

certificate management infrastructure, or PKI (Public Key Infrastructure)[15].

It is worth pointing out that although several PKI implementations are currently evolving (such as IETF's PKIx(Public-Key Infrastructure,X.509)[16], PKCS(Public Key Crypto System)[17], PGP(Pretty Good Privacy)[18], SPKI(Simple Public Key Infrastructure)[19], SDSI(Simple Distributed Security Infrastructure)[20],etc.), there is no single PKI implementation nor even a single agreed-upon standard for setting up a PKI. Even those implementations that are based on the same standard X.509 recommendation[16] are still incompatible with each other because of independent interpretations in their actual implementations[21] [22].

Further more, security protocols, operations and inter-operations between principals (agents), as well as public key management are really difficult for the ordinary end-users to handle. Those routines themselves should be autonomously and cooperatively performed by programs running on the Internet so that the workload of the users can be lessened.

2.2 Security Agent: Agent-Based/Oriented PKI

To exploit public key cryptography for agent security, it would be unavoidable for us to consider construction of agent-oriented PKI, which is expected not only to be suitable for agent-based applications[11], but also can take advantages of autonomous agent to facilitates inter-operability and flexibility.

Unlike the traditional way of PKI implementation, our PKI implements the authorities of authentication verification service systems as autonomous software agents, called *security agents*, instead of building a static monolithic hierarchy. Formats of certificates for various applications can be personalized by the users or specific applications. The authentication relationship can be

dynamically established even across multi-certificate hierarchies by use of the security agents.

From the viewpoint of a user, the security agent can be thought of as a kind of configurable facilitator that can be employed by any group of agents or the owner of the agents to construct their own authentication verification service system. What we mean by "configurable facilitators" is that we do not pre-specify any particular certification format and hierarchical relationship in the software (like in other traditional PKI projects), but allow the users to define the format(s) of the certification(s) and the name space(s) as they need (customizing). The hierarchical relationship is dynamically formed as the agents apply/issue their certificates according to the desires of the applications.

From the viewpoint of PKI structure, a security agent can be thought of as a node in a dynamically formed hierarchy. More than one authentication verification systems may cross a node, since a single security agent can hold multiple certificates with different certificate name (such as "PGP" "certificate", "RSA PKCS certificate", "X community certificate", etc.), formats and name spaces-hierarchical relationships. (refer to Figure 2.1).

In order for a security agent to manage public key certifications, it is capable of performing a basic set of tasks: issue/apply a certificate, update/revoke a certificate. We note that a security agent could potentially provide additional capabilities, such as retrieve, transfer, or exchange credentials among different hierarchy systems, or introduce one agent to another, or delegate one agent to act on another's behalf, etc. But we leave those for future work. Interested reader can refer to [11] for more details.

2.3 The Characteristics of Agent-Based PKI

The implementation of the agent-based PKI lays a authentication foundation for agent security. Additionally, it brings to the construction of PKI (not only for agent-based applications) with some advantages:

1. It makes the construction of scalable authentication system much more feasible by employing the security agents in a bottom up fashion.
2. It makes automatic inter-operation of multi-certificate authentication system possible.
3. It makes it much easier to customize certificate management while relieving the workload for certificate users.

Naturally, there remain some open problems and issues that we discuss in section 5.

3 Agent Security Methodology and Architecture

The establishment of agent-based PKI makes it technically feasible for agents (also for non-agent-based applications) to conduct security operations based on public key cryptosystem rather than radically change paradigm of the basic methodology of cryptology. In this section, however, we give a brief summary of the methodology[1].

3.1 Basic Methodology

Agent-based PKI lays a foundation so that public keys of agents can be distributed and managed among agent society in a trusted way. The confidentiality of communications can be provided

easily by encryption with public keys, which are included in public key certificates publicly distributed. And the authentication, integrity, and nonrepudiation can be provided by digital signature signed on the different objects:

1. Authentication verification can be obtained in such a way that the verifying agent generates a fresh nonce and the proving agent signing a digital signature on the nonce. With the public key included in the certificate of the proving agent, the verifying agent can verify the authenticity of the identity of the proving agent.

   If the prover is a mobile agent visiting a site of the Internet (or node on a network), it can prove its identity to the site by signing a signature on whole code of the mobile agent. The mobile agent travels with the signature and the certificate, so that the site can verify the authenticity of the mobile using the public key in the attached certificate. If the mobile agent is authorized, then it can be added in an access control list (ACL) of the site.

   Agent running as an applet may be signed up with a URL[23][24] and it will be able to do more work (create a socket, read/write file etc.) locally, if its identity has been authenticated and it is in the ACL.

   A systematic study on authentication can be found in [25].

2. To provide integrity of a message, the sender signs a signature on the message and attaches the signature and its public certificate with the message, then sends it to the receiver. The receiver can verify the integrity of the message by checking the signature with the public key in the certificate.

3. A message with a signature lets the receiver be able to verify authenticity of the message. On the other hand, this function of public key cryptosystem provides a way for nonrepudiation, in that the principle who signs the message can not deny that it sent the signed message because only the one who holds the corresponding private key can sign a correct signature on the message. To prevent a signed message from being replayed, a fresh nonce, such as timestamp, is generally included in the data to be signed.

It is suggested that if the identity of a principal (agent) is essential to the meaning of a message, it is prudent to include the principal's identity explicitly as a part of data to be signed[27].

An application in electronic commerce may involve various security problems for which more than one security mechanisms must be comprehensively used. For instance, if we want that an information agent just serves authorized query agents, then, all of authentication, integrity and confidentiality mechanism would be needed:

1. Information agent needs to authenticate the identity of the querying agent.
2. Information agent needs to check the integrity of the query.
3. Information agent needs to encrypt the reply in order for only the intended agent who sent the query can get the information.

From the simple example, we see that given a task, an agent needs to know:

1. security policy: what security rules can satisfy the security requirements. (e.g. do we want only authorized agents access to the information?)
2. security protocol: how to put the policy into effect.
3. security operation: in each step, what operation should be carried out on what object. (e.g. verify signature on query for checking the integrity of query, etc.)

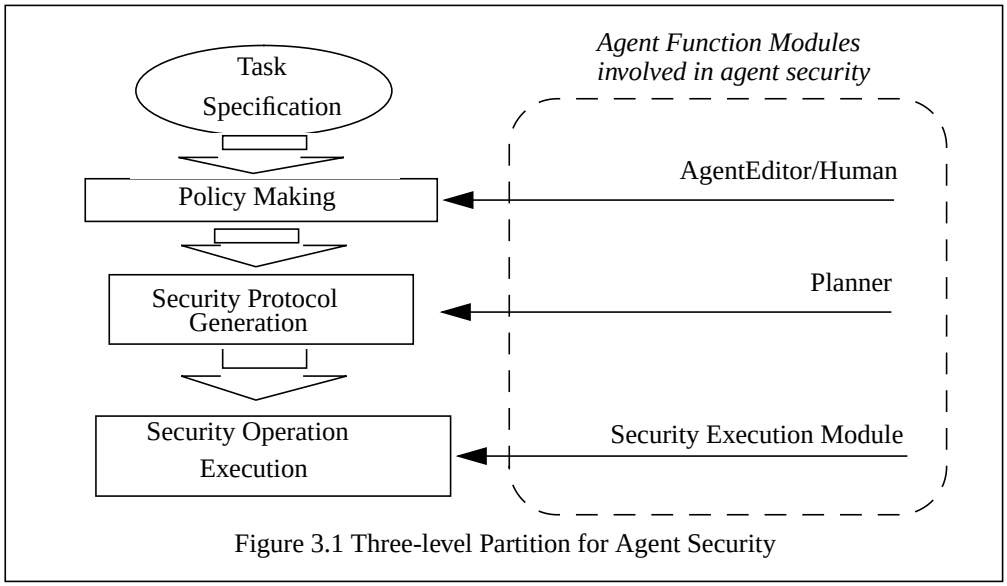

Figure 3.1 Three-level Partition for Agent Security

3.2 Agent Security Architecture

In the RETSINA project[8], an agent consists of a set of functional modules, where each module deals with a specific aspect of agent functionality. For instance, "communicator" module is a RETSINA agent dealing with the communication with other agents. There are three modules involved in agent security: AgentEditor, Planner, and Security Module, which correspond to the three levers, policy specification, protocol generation, and operation execution. See also Figure 3.1.

Defining a set of security policies for a given task is the first level job for agent security and it would be done by the owner of the agent. According to the security policy, a security protocol must be generated by a functional module (planner) as a part of procedure for the agent to complete the task. To execute the security protocol, some basic security functions, such as encryption, decryption, signing, verification, etc. would be called during the execution of task.

The relationship among cryptographic functions, security operations, security mechanisms, security protocol, and security policy are shown in Figure 3.2.

3.3 A Work Diagram

Let's take a further look at the work diagram in the design of the RETSINA project:

Policy making with AgentEditor: AgentEditor is the interface for the human to build their agent based on RETSINA. Defining security policy is part of specification of the task for which the agent is being built, or say, customized by the owner of the agent. Conceptually, the policy is specified with the interface as follow:

1. Specify the expected security requirements for both incoming and outgoing message, such as whether there is a need to verify the origin of a query message, or whether a reply message must be signed so that the receiving agent can verify the authenticity of it.

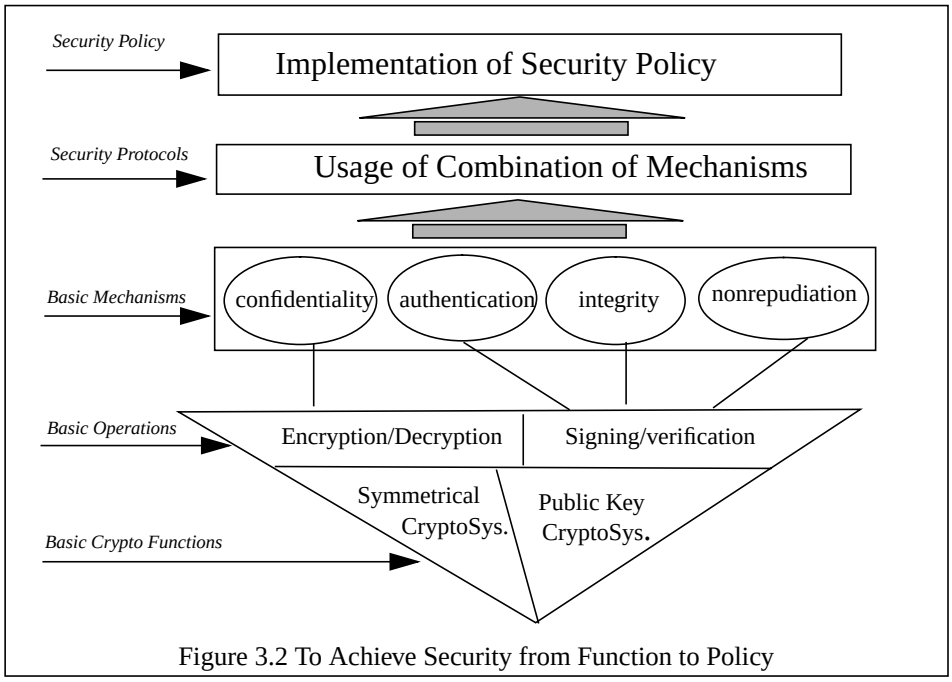

Figure 3.2 To Achieve Security from Function to Policy

2. Define the credential requirements for acceptable queries or other messages. For example, which certificate could be valid for a querying agent to query which kind of information, or how many authorities are required for an agent to validate a query.

3. Check the correctness and consistency of policy specification. For example, if it is specified that an incoming query must be authenticated, then the corresponding outgoing message must be specified as confidential message, which must be encrypted with the intended receiving agent's public key – this would be embodied when the protocol is generated by the planner (see below).

Policy making is an active research and development field as application of cryptography [26] and security usability.

Protocol Generation: In RETSINA, a planner takes charge of planning to fulfill the agent's goals and tasks. To meet the security requirements defined by human, the planner will generates a set of security protocols according to the methodology discussed in the last subsection, which described what security operations are needed comprehensively on what objects to satisfy the required security mechanisms. The protocol specifies all the concrete operations and the objects of the operation in each step. The generated protocol will be checked by means of some automatic checking procedure. Automatic protocol checking is another active research and development field in application of cryptography[30], and its application on agent security could be an exclusive research topic. The approved security protocols will be saved in protocol database (PDB).

Operation Execution: When the customization of agent is finished, the security policy would have been specified and the protocols would have been generated and saved in PDB. Once a task is given, a protocol matching the task is chosen from protocol database (PDB), and will be conducted as a part of procedure of execution of the task. An execution monitor will monitor the

execution of the protocol. The security execution module is the functional module that executes the concrete security operations.

4 Extension of KQML for Agent Security

KQML (Knowledge Query and Manipulation Language), is a communication language and protocol which enables autonomous and asynchronous agents to share their knowledge and work towards cooperative problem solving[9]. However, agent security issues were not taken into consideration in the original version of KQML specification. In this section, we introduce an extension of KQML for agent security, including: a set of new parameters, a set of new performatives.

4.1 New Parameters

1. **:signature**
   The value of the signature in a performative is a digital signature signed on the content of the performative. This signature is signed by the agent that sends the KQML message.

2. **:senderCert**
   To verify the signature in a performative, the receiver needs the public key of the sender. The included senderCert of a performative enables the receiver to get and verify the authenticity of the public key, and then to verify the signature with the authenticated public key. Generally, signature and senderCert appear at the same time in a performative.

3. **:senderCertChain**
   For the dynamic management of certificates, the senderCertChain, in which the certificates of the agents along the path from the root security agent through the agent that holds the senderCertChain, will be needed as parameter in the performative. See also [11].

4. **:senderCertName**
   This parameter indicates which kind of certificate is used by the sender of the massage, so that the receiver will be able to parse the information included in the senderCert with certain format under the name of "senderCertName".

5. **:receiverCert**
   The certificate of receiver's public key.

6. **:receiverCertName**
   The name of the receiver's public key certificate. This parameter indicates which public key of the receiver is used to encrypt the content of message, because with multi-certificate authentication system, a receiver can hold more than one public key certificates. Being informed of the certificate, the receiver can easily choose the corresponding private key to decrypt the encrypted content of the message.

For example, "tell" is one of the performatives defined in original KQML[9]. But new parameters in the performative enable agent to "tell" verifiable secret:

```
tell:
        :language CIPHER
        :content {the encrypted M}
```

```
        :receiverCertName CMUCertificate
```

and M is another KQML message embodied in the first KQML package:

```
tell:
        :language PLAINTEXT
        :content {the content}
        :senderCert {a public certificate of sender}
        :senderCertName RetsinaCertificate
        :signature {signature signed by sender}
```

A detailed processing of the KQML message is as following:

1. The KQML parser of receiver extracts the content of first KQML package, encrypted M and passes it with RetsinaCertificate to security execution module.
2. The security execution module picks up the corresponding private key, decrypts it and gets plain M.
3. Since M is a KQML message, it will be returned to KQML parser. The parser parses M and passes the content, signature, and senderCert to security execution module.
4. The security execution module verifies the authenticity and integrity of the content.

4.2 New Performatives

In [11], we presented new KQML performatives for public management of agent-based PKI. These are the following:

1. **apply-certificate**
2. **issue-certificate**
3. **renew-certificate**
4. **update-certificate**
5. **revoke-certificate**

Please refer to [11] for more details, such as how to use the new performatives for management of trust relationship.

5 Conclusion and Further Work

Three basic efforts to achieve agent security, which are currently being implemented within our multiagent infrastructure, RETSINA, are discussed in the paper: 1) establishment of external environment – agent-basic PKI; 2) internal design of security architecture; 3) extension of agent language for security inter-operations between agents. An overview of these areas is given. However, since the treatment of agent security literature has very scant, there are lots of work to be done. Some of the open problems are:

1. How to specify more completely security strategy and policy and how to implement them in agent development. Since many agents are customizable, we need to find a method for the user to configure the security characteristics for their application. Automatic checking mechanisms are also needed to verify whether the security configuration could meet their requirements. This needs a further study on formal method on verification of security protocols[29][31].

2. Is it secure for a mobile agent to carry private key for signing signature when it is visiting a remote host?

3. How to incorporate existing sophisticated cryptology technology with agent development. such as: blind signature for electronic commerce[32], zero knowledge proof for authentication[33], etc.

4. How to allow a mobile agent to do some computation on sensitive data without revealing the data?

References

[1] Bruce Schneier, Applied Cryptography, Second Edition, John Wiley & Sons, Inc. 1996.

[2] National Bureau of Standards, NBS FIPS PUB 46-1, Data Encryption Standard, U.S. Department of Commerce, Jan 1988.

[3] Xuejia Lai: On the Design and Security of Block Ciphers, ETH Series in Information Processing, vol. 1, Hartung-Gorre Verlag, Konstanz, Switzerland, 1992.

[4] R.L. Rivest, A. Shamir, and L.M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2): 120-126, February 1978.

[5] L. Gong and R. Schemers, Implementing Protection Domains in the Java Development Kit 1.2, In Proceedings of the Internet Society Symposium on Network and Distributed System Security, San Diego, California, March 1998.

[6] L. Gong, Java Security: Present and Near Future, IEEE Micro, 17(3):14–19, May/June 1997.

[7] http://java.sun.com/products/jdk/1.2/docs/guide/security/index.html

[8] Syscara, K., Decker, K., Pannu, A., Williamson, M. and Zeng, D., Distributed Intelligent Agents. IEEE Expert, pp.36-45, December 1996.

[9] Tim Finin, Yannis Labrou, and James Mayfield, KQML as an agent communication language, in Jeff Bradshaw (Ed.), Software Agents, MIT Press, Cambridge (1997).

[10] Qi He, Security Module v1.0: Design Document of Project Retsina, Robotics Institute, Carnegie Mellon University, Jaun 12, 1997.

[11] Qi He, Katia P. Sycara, and Timothy W. Finin, Personal Security Agent: KQML-Based PKI, to appear in Autonomous Agents'98, Minneapolis/St. Paul, May 10-13, 1998.

[12] Leonard N. Foner, A Security Architecture for Multi-Agent Matchmaking, Proceeding of Second International Conference on Multi-Agent System, Mario Tokoro, 1996

[13] Tim Finin, James Mayfield, Chelliah Thirunavukkarasu, Secret Agents - A Security Architecture for the KAML Agent Communication Language, CIKM'95 Intelligent Information Agents Workshop, Baltimore, December 1995.

[14] URL, http://www.rsa.com/rsalabs/newfaq/q1.html

[15] W. Timothy Polk, Donna F. Dodson, etc, Public Key Infrastructure: From Theory to Implementation, http://csrc.ncsl.nist.gov/pki/panel/overview.html, NIST

[16] URL, Public-Key Infrastructure (X.509) (pkix), http://www.ietf.org/html.charters/pkix-charter.html

[17] URL, RSA Laboratories, PKCS (Public Key Crypto System) http://www.rsa.com/rsalabs/pubs/PKCS/

[18] Philip R. Zimmermann, The Official PGP User's Guide, MIT Press 1995.

[19] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian M. Thomas, Tatu Ylonen, Simple Public Key Certificate, http://www.clark.net/pub/cme/spki.txt

[20] Ronald L. Rivest, Butler Lampson, SDSI - A Simple Distributed Security Infrastructure, http://theory.lcs.mit.edu/ cis/sdsi.html

[21] Peter Gutmann, X.509 Style Guide, http://www.cs.auckland.ac.nz/pgut001/x509guide.txt

[22] E. Gerck, Overview of Certification Systems: X.509, CA, PGP and SKIP, http://novaware.cps.softex.br/mcg/cert.html.

[23] URL, http://java.sun.com/products/jdk/1.2/docs/tooldocs/solaris/jar.html

[24] URL, http://java.sun.com/security/

[25] Butler Lampson, Martin Abadi, Michael Burrows, Authentication in Distributed Systems: Theory and Practice ACM, 1992.

[26] Matt Blaze, Joan Feigenbaum, Jack Lacy, Decentralized Trust Management, In Proceedings 1996 IEEE Symposium on Security and Privacy, May, 1996.

[27] Martin Abadi and Roger Needham, Prudent Engineering Practice for Cryptographic Protocols, IEEE Transactions on Software Engineering, Vol. 22. No.1, January 1996.

[28] R. Anderson and R. Needham Robustness Principles for Public Key Protocols Lecture Notes on Computer Science, 963:236-247, 1995.

[29] M.Burrows, M.Abadi, and R.Needham, A logic of authentication. Technical Report39, DEC Systems Research Center, February 1989.

[30] Darrell Kindred, Jeannette M. Wing, Fast, Automatic Checking of Security Protocols, Proc. of the USENIX 1996 Workshop on Electronic Commerce, November 1996.

[31] L.Gong, R.Needham, and R.Yahalom, Reasoning about belief in cryptographic protocols, In Proceedings of the 1990, IEEE Computer Society Symposium on Research in Security and Privacy, pages 234-248, May 1990.

[32] D. Chaum, Security without Identification: Transaction systems to make big brother Oslete, Communications of the ACM Oct, 1985, Vol. 28, No. 10.

[33] Amos Fiat and Adi Shamir, How to Prove Yourseldf: Pratical Solution to Identification and Signature Problems, CRYPTO'86, LNCS 263, Springer-Verlag, 1987.