A Solution to Open Standard of PKI

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

Zhongmin Su
Dept. of CS and Telecommunications
University of Missouri-Kansas City
Kansas, MO 64110, USA
zsu@cstp.umkc.edu

February 11, 1998

Abstract

PKI (Public Key Infrastructure) is fundamental for many security applications on the network. However, there are so many different kinds of PKI at current stage and they are not compatible. To solve the problem, we propose to implement the authority of authentication verification service systems as personal autonomous software agents, called security agents. In this paper, we introduce its concept and architecture, as well as its communication language, which is needed for public key management and secure communications among security agents and application agents.

Keywords: security, PKI (Public Key Infrastructure), authentication, agent architecture, KQML, interoperability.

1 Introduction

It is well known that the integrity of public key vitally determines the whole security of communication, especially electronic transactions on the network. So different kinds of Public Key infrastructures (PKI) [1] are designed and their implementations are currently evolving. The examples include IETF's PKIx(Public-Key Infrastructure,X.509) [2, 3, 4, 5], PKCS(Public Key Crypto System)[6], PGP(Pretty Good Privacy)[7], SPKI(Simple Public Key Infrastructure) [8], SDSI(Simple Distributed Security Infrastructure)[9], etc. Most of the systems are organized in a hierachical manner to issue and verify the certificates. and there is no single agreed-upon standard for setting up a PKI. Even those implementations are based on the same scheme (say X.509 recommendation), they are still not fully compatible with each other due to the independent interpretations in their actual implementation. So it is a crucial issue to overcome the imcompatibility and enable wide spread authentication offered by PKI.

The simplest solution is to establish a uniform system with only one kind of certificate format, name space and management protocol. However, it is not only infeasible to enforce in practice, but also undesirable in many situations. For example, in a given situation, the information of organizational relationships is needed as an element in a certificate, but in other situations, this information is not needed and it shouldn't be included in the certificate for the sake of security and privacy. This flexibility in PKI implementation requires that multiple types of certificates, definition of name space, and management protocols tailored for various applications must be developed[9].

A software agent is a process which can travel from one place to another within the telesphere. It can be unattended for a long time. Once an agent is in a place, it can interact with other agents to learn new knowledge and fulfil a goal. Nowadays, agents are widely used in many different kinds of applications. In this context, our research makes an effort at using the concept of agent to flexibly implement decentralized PKI[10].

One the other hand, the development of the Internet is changing the traditional paradigm of software, which is monolithic and passively operated by humans, to the new agent-based technology which works cooperatively and autonomously. Agents, as the new generation of software, will be delegated by humans to automatically perform tasks, including digitally conducting transactions across the Internet. Security issues are identified as critical for the success of agent-based Internet programming[11]. Agent-oriented authentication verification services must be supplied for most agent-based applications. In fact, as primarily human-delegated software, agents will be an ideal application domain of modern cryptography in the very near future.

Though agents have been widely used in many applications. It is still a new idea to introduce the concept of agent to solve security problems. The treatment on the security issues of software agent is also very scant. [12] discussed some basic principles for agent developers. In [11], language for agents to support the secret communication was discussed based

on cryptography techniques. However, like the applications of public key cryptosystem in human society, all of security schemes and protocols designed for open agent society can not make any sense without a scalable authentication service, and PKI aim at providing such authentication service.

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

We propose to implement the authorities of authentication verification service systems as autonomous software agents, called *security agents*. This open implementation of agent-based PKI facilitates interoperable, flexible, and agent-oriented authentication verification service for various applications.

In this paper, we discuss two aspects of our flexible PKI development: (1) The security agent concept and its functional modules — we describe the fundamental idea of implementing PKI by means of a security agent. (2) An extension of Knowledge Query and Manipulation Language (KQML)[13] — KQML is a language and protocol for exchanging information and knoweledge between agents. We propose a set of new elements to support key management and secure communication among agents.

2 Security Agent

2.1 Software Agent and KQML

Software agent is an emerging system-building paradigm, it is now widely used in information retrieval, distributed systems, database and knowledge base, and many other aspects of AI. Some researchers believe, like expert systems in the middle of 80's and object oriented techniques at he beginning in 90's, agent will also become a revolutionary technology in computer science.

A software agent is a process characterized in the following ways:

1. *Adaption*: Agents adapt to their environment and their users, and learn from experience.
2. *Cooperation*: Agents use standard languages and protocols to achieve common goals.
3. *Autonomity*: Agents act autonomously to pursue their agendas.
4. *Mobility*: Mobile agents migrate from machine to machine in a heterogeneous network under their own control.

The properties of agents make them useful in many diffenrent kinds of applications, as well as in security area. As we mentioned above, agents can communicate with each other by

their own language, so they can cooperate to fulfil a common goal. There are several kinds of common agent communication language, such as: KQML, KIF (Knowledge Interchange Format), Ontolingua (a language for defining sharable ontologies), Protolingua (a language for defining protocols based on communicative primitives). The most common language is KQML.

KQML is a high level language intended for the run time exchange of information between agents. There are three layers involved in KQML: the content layer, the message layer and the communication layer.

The content layer contains the actual content of the message. The communication layer describes the lower level communication parameters, such as the identity of the sender and the receiver. The message layer forms the core of KQML, and determines the kinds of interactions one agent may have with another.

The syntax of KQML is based on a balanced paranthesis list. The initial element is called the performative, the remaining elements are the performatives arguments (as keyword and value pairs). The following is an example:

```
tell:
   :sender customer
   :content (123 456 7890)
   :receiver retailer
   :in-reply-to credit-card-number
   :language LPROLOG
   :ontology NYSE-TICKS
```

In this message, the KQML performative is "tell". The content is "123 456 7890", this is the content level. The values of the :sender, :receiver, :in-reply-to keywords form the communication layer. The performative, with the contents of :language and :ontology form the message layer. From this example, we can also realize the necessity of a security machanism in agent communication.

2.2 The Idea of Security Agent

Existing PKI implementations began with specifying their certificate formats and the name spaces through a pre-defined hierarchies, such as the DNS(Domain Name System) name hierarchy. This method entails inflexible implementation. The properties of an agent make it ideal for the application of PKI. Since agents can adapt to the environment, instead of specifying the format of certificates, name space or hierarchy structure, we allow user to specify the details for implementation. Typically there are multi-parties involved in a security protocol, they have to cooperate with each other to make the protocol work. If the parties involved in the protocol are delegated by agents, then they can naturally cooperate with each other according the the security policy. And since agents are highly autonomous, human beings don't need to get involved into the details of the protocol. Thus, we can

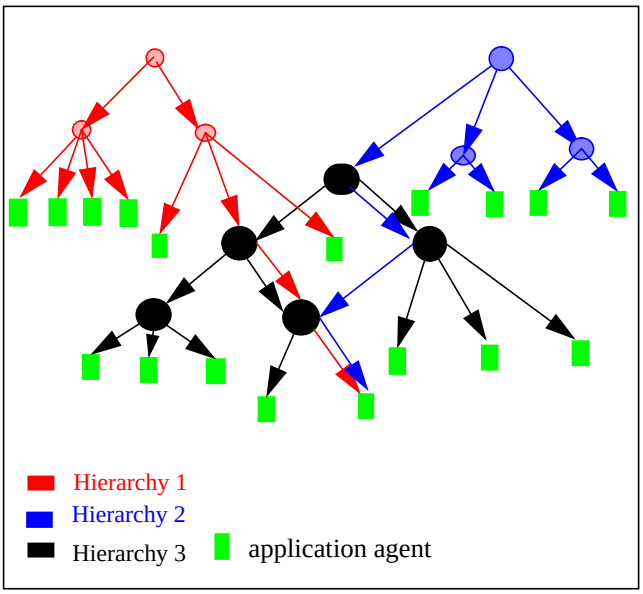

Figure 2.1 Multiple Hierarchies across a agent.

develop a *security agent* to deal with the authentication service. This provides a flexible framework where different applications can specify their own certificate formats.

From the viewpoint of a user, the security agent can be thought as a kind of *configurable facilitator* that can be employed by any group of users, organization, community etc. to construct their own authentication verification service system. "configurable facilitator" means 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 (customized). The hierarchical relationship is dynamically formed as the agents apply/issue their certificates according to the goals of the applications. Of course, the certificate formats in existing PKI implementations can be adopted if they are suitable for an application.

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 hierarchical relationships for name space. (refer to Figure 2.1).

Security agents, like other application agents, communicate with each other with KQML. However, the current version of KQML does not support many security operations needed in public key management, although some changes were made for agent security in [11]. We propose a security extension of KQML in the following section, our extension enable

agents to identify multiple certificates and cooperatively conduct security interoperations.

2.3 3-level Module for Security Agent

Like human being, an agent needs to know the following for a given task:

1. security policy: what security rule can satisfy the security requirements. (e.g. which or what kinds of agents can access a certain kind of information?)

2. security protocol: how to put the policy into effect. (e.g. do the job step by step to reach the goal.)

3. security operation: in each step, what operation should be carried out on which object. (e.g. verify signature on query to check the integrity of query, etc.)

This top-down analysis gives us a hint for designing the architecture for security agent.

The security agent architecture is based on the agent architecture we have developed in the RETSINA multistage infrastructure[14]İn RETSINA, an agent consists of a set of functional modules, each module would deal with a specific job. For instance, "communicator" module deals with the communication with other agents. Three modules are directly involved into agent security: agent editor, planner, and security module, which are corresponding to the three level works, policy specification, protocol generation, and operation execution.

Defining a set of security policies for a given task is the first level job for agent security and it would be done during the period the owner of the agent customizes his agent through the agent editor.

A security protocol is generated by "planner" for the agent to complete the task according to the security policy. This is the second level job.

To execute the security protocol, some basic security functions, such as encryption, decryption, signing, verification, etc. would be called during the execution of task. This is the third level job done by security execution module. The detailed architecture will be discussed later.

The relationship among cryptographic functions, security operations, security mechanisms, security protocol, and security policy are showed in Figure 2.2.

2.4 Function Modules and Architecture

Though security agent could potentially provide many services, such as retrieve, transfer, exchange credentials among different hierarchy systems, introduce one agent to another, or delegate one agent to act on another's behalf, etc, the basic operations are more or less the same. Here, we sketch the structure of security agent based on these basic operations:

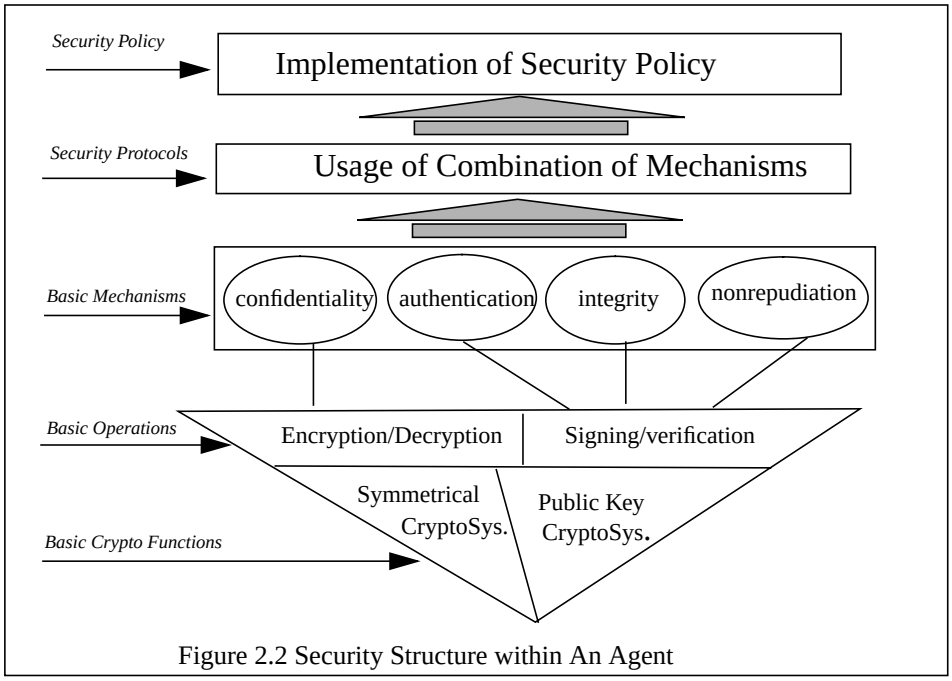

Figure 2.2 Security Structure within An Agent

issue/apply a certificate, update/revoke a certificate. We describe the components (modules) of security agents by their functionality.

The modules in the current implementation of the security agent are as follows:

1. Communicator: It deals with communications with other agents. In fact, what the communicator module does is to accept and parse messages (KQML packages) from outside agents, or to pack outgoing messages into KQML packages and send them out to intended agents.

   The parser must recognize if a message is encrypted, put it into a task object and send it to the planner. In some circumstances, this procedure may be repeated several times, if the original KQML message includes recursive KQML messages.

2. Task Planner: The message from outside, represented as a task object, is passed to the task planner. Upon receiving a task object, the planner initializes a process with the received data as the input according to a specific protocol extracted from PDB (Protocol Database, see below). The protocol steps are passed to the scheduler.

3. Task Scheduler: This module schedules the protocol steps to be executed. Since its services are used by many other agents, the security agent needs to arrange the priority and schedule the requests for security it receives from many different agents. After the protocol steps have been scheduled, they are passed to the execution module.

4. Execution Module: This module executes the process initiated by the task scheduler step by step. The basic security operations executed by the execution module are: encrypt/decrypt, sign/verify a message.

5. Human-Agent Interface: Human/agent interface is designed as an interface for user to set up and customize the system. More precisely, through the interface users can:

   (a) can define or choose a format of certificate they want, name space length of their public key and algorithms of cryptography, as well as a name of certificate.

   (b) apply/issue some kind of public key certificates - During the application procedure, the applicants need to interact with their agents. When applicants receive their certificates, they also need to confirm that the information included in the certificate is correct and the signature is signed correctly by the intended security agent.

   (c) Input the sets of security protocols for various certificate management strategies and policies of authentication service system.

6. PDB (Protocol Database): Every security agent should store all sets of security protocols needed in its PDB for various managements tasks (routines) required in all of the authentication service systems across it. The basic protocols are certificate update protocols, certificate revocation protocols, certificate application/issuing protocols, etc. Given a task object by the parser, the planner looks up the PDB, then starts a process according to the matched protocol from PDB. Subsequently, the execution module executes the protocol automatically.

7. CDB (Certificate Database): When the agent applies for a certificate from a security agent, it will be given not a single certificate but a *chain of certificates*. This chain of certificates consists of the certificates of all the security agents along the path from the root security agent through the parent security agent, from which it applies its certificate, in the authentication hierarchy. Each security agent stores its chain of certificates in its CDB. Later on, when the security agent wants to communicate with another security agent, it does not necessarily contact other higher level security agents to retrieve the participant's public key certificate(s). The agents can exchange their certificate chains (or part of their chains) to prove their authenticity according to their positions in the name space. By caching some most frequently used certificates, the communication costs will be cut down dramatically.

Figure 2.3 shows the relationships and data flow among the security agent's functional modules.

Suppose, a message from another agent comes to the communications module. After the message is received, it is parsed by the parser. In the simplest situation, the message is a kind of datum that represents a request from another agent. It is processed by the parser, which outputs it as a task object and passes it as an objective to the agent's planner.

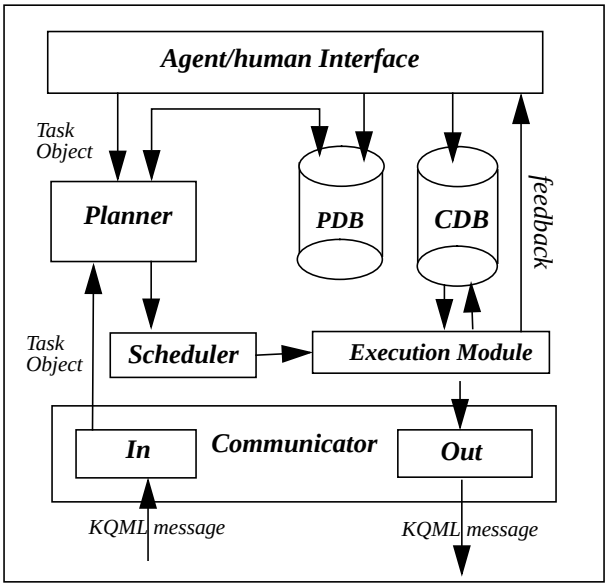

Figure 2.3 Structure of Security Agent.

After the planner has planned for this objective, the plan actions are passed to the task scheduler module to be scheduled. Subsequently, the scheduled actions are executed by the execution module. Results are sent back to the agent who originated the message through the communicator.

3 Extensions to KQML

KQML (Knowledge Query and Manipulation Language) is a widely used communication language and protocol which enables autonomous and asynchronous agents to share their knowledge and work towards cooperative problem solving[13]. However, agent security issues were not taken into consideration in the original version of KQML specification. Though some changes were made for secure communications based on KQML[11], it is still can not satisfy the requirements of public key certification management. In order to implement KQML-based PKI, we propose a KQML ontology, several new parameters and performatives as follows. The new ontology is:

PKCertificate

It enable the agents (including application agents) to know that the performative they received concerns interactions about public key certificate management. Upon receiving the performative, the receiver will check the authenticity of the updated certificate by verifying

signature with the public key included in the original certificate.

3.1 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 is the holder of the senderCertChain, will be needed as parameter in the performative. See also [15].

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 on public key certificate. Being informed of the certificate, the receiver can easily choose the corresponding private key to decrypt the encrypted content of the message.

Following is an example of KQML message with some new parameters:

```
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}
```

"tell" is one of the performatives defined in original KQML[16], the new parameters in the performative enable agent to "tell" verifiable secrets:

1. The value of parameter language, CIPHER, indicates that the content is encrypted.
2. Knowing CMUCertificate, the receiver is able to choose the corresponding private key to decrypt the cipher.
3. With the signature signed by sender and the senderCert, the receiver can verify the authenticity of the content (cipher).

Generally speaking, signing before encrypting prevents the attack with "trapdoor" moduli for which the signed document can be forged by computing discrete logarithms and changing the public key (key spoofing)[17]. How to make such a decision is the job of planner that schedules how to complete a task step by step.

A detailed processing the KQML message would be as following:

1. The KQML parser of receiver extracts the content of first KQML package, encrypts 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 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.

3.2 New Performatives

The extension of new performatives is mainly for public key management of agent-based PKI[15].

1. **apply-certificate**
   In order to securely communicate with others, when an agent is created, it will apply for a certificate in which an automatically generated public key will be included. To apply for the certificate from an authentication authority, a security agent, the agent will send the following performative in the KQML message, as its certificate

application.

```
apply-certificate:
   :language {name of certificate}
   :content {all the elements of certificate except signature
        of the authority}
   :ontology PKCertificate
```

where the content of "content" is all the elements needed to be included in the certificate which is applied. The content of "language" identifies the name of certificate, which will enable receiver's KQML parser to know what elements are included as the "content" of this performative and then extract them out.

2. **issue-certificate**

   If an application for a certificate is approved, the security agent in charge of issuing certificates will send back a performative as follow:

```
issue-certificate:
   :language {name of certificate}
   :content {issued certificate}
   :senderCert {authority's certificate}
   [:senderCertChain {the certificate chain of authority}]
   [:signature {signature signed by the security agent}]
   :ontology PKCertificate
```

   Where the content of "language" also identifies the type of certificate which should be the type intended by the applicant agent. The issued certificate is included as the content of "content".

   Upon receiving this performative, the agent which applies for the certificate can extract the public key in "certificate" (authority's certificate) and check the authenticity of the issued certificate by means of verifying the signature in the issued certificate.

3. **renew-certificate**

   Each time when an agent is going to change its public key, or other pieces of information in its certificate, it will send the following performative to the security agent that issued the original certificate.

```
renew-certificate
   :language {name of certificate}
   :content {content of new certificate}
   :senderCert {original certificate}
   :signature {signature on content of new certificate}
   :ontology PKCertificate
```

   When receiving the performative, the security agent will extract the public key from the original certificate and check the authenticity of the content of new certificate by verifying the signature with the public key. If the authenticity has been verified, the

security agent can sign the new certificate and issue it to the applicant by sending back an issue-certificate performative.

4. **update-certificate** If a security agent updates its public key, it should inform (1) the agents that applied for a certificate from it, and (2) the agents whose certificates were issued by the agents to whom the updated certificate has been sent. All these agents, upon receipt of the update-certificate, will update their CDB and renew their certificates. To inform others about the updated certificate, a security agent should use the following performative:

   ```
   update-certificate:
      :language {name of certificate}
      :content {updated certificate}
      :senderCert {original certificate}
      :signature {signature on updated certificate with the
           public key in the old certificate}
      :ontology PKCertificate
   ```

   Upon receiving the performative, the receiver will check the authenticity of the updated certificate by verifying signature with the public key included in the original certificate.

5. **revoke-certificate**

   A certificate could be revoked for some reasons. If a security agent is going to revoke its certificate, it will send the following performative to other agents associated with it, especially the agents that hold the certificates issued by the agent whose certificate is to be revoked. When an agent is informed of revoked certificate, it should also forward the performative to the agents that hold the certificates issued by it.

   ```
   revoke-certificate:
      :language {name of certificate to be revoked}
      :content {the certificate to be revoked}
      :signature {signature on the certificate to be revoked},
      :senderCert {certificate}
      :senderCertChain {certificate chain}]
      :ontology PKCertificate
   ```

   where the signature is signed with the public key included in the certificate to be revoked.

These are the performatives for the basic certificate management. If, more sophisticated certificate management is needed in the future, additional performatives can be developed.

4 Conclusion

In this paper, we propose to used agent-based implementation as an open standard as PKI. Comparing with traditional PKI implementation, the security agent makes the construction of scalable authentication system much more feasible by employing the security agents in a bottom up fashion, it also makes interoperation of multi-certificate authentication system possible and can relieve the workload for certificate users.

For the future work, we would like to study the construction and specification of security policy, as well as increase the robustness of security agent to be against different kinds of attacks.

References

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

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

[3] URL, *International Telcommunication Union, X.509*, http://www.itu.int/itudoc/itu-t/rec/x/x500up/

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

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

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

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

[8] 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

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

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

[11] 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.

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

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

[15] 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.

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

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