Personal Security Agent: KQML-Based PKI

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

Timothy W. Finin
Dept. of Computer Science & Electrical Engineering
University of Maryland Baltimore County
Baltimore, MD 21250
finin@cs.umbc.edu

Abstract

Certificate management infrastructure, a.k.a. PKI (Public Key Infrastructure), which issues and provides access to public key certificates to preserve the integrity of a public key, is fundamental for electronic commerce and business across the Internet. To satisfy the requirements of various applications, PKI should demonstrate customization to user needs, interoperability and flexibility in its implementations so it can satisfy the needs of various applications. Particularly, due to the popularity of software agent-based applications over the Internet, security will be urgently needed by the "agent society". We propose to implement the authority of authentication verification service systems as personal autonomous software agents, called security agents. In this paper, we present two aspects of KQML-based PKI: 1. the security agent concept and its functional modules; 2. an extension of KQML, which is needed for public key management and secure communications among security agents and application agents.

Area: Software Agents

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

1 Introduction

The public key cryptosystem is playing an increasingly important role in electronic transactions on computer networks. However, whenever we 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 (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 certificate management infrastructure, or PKI (Public Key Infrastructure)[1].

However, several PKI implementations are currently evolving (such as IETF's PKIx(Public-Key Infrastructure,X.509)[3], PKCS(Public Key Crypto System)[4], PGP(Pretty Good Privacy)[5], SPKI(Simple Public Key Infrastructure) [6], SDSI(Simple Distributed Security Infrastructure)[7], etc.), and 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[8] are still incompatible with each other because of independent interpretations in their actual implementations[9][10]. So, overcoming this incompatibility and enabling wide spread authentication verification offered by the PKI is a crucial issue. It is also one of the motivations of our work.

To resolve the PKI interoperability problem, the simplest solution is to establish a uniform system with only one format of certificate, name space and management protocol. However, not only is this extreme difficult to enforce in practice, but also it is undesirable in many situations. For example, different circumstances warrant holding different certificates to satisfy a variety of authenticity verification requirements. 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 is a basic principle of security: "It should not be possible to do more or learn more than what is specified in the protocol[11]"[1]. With the increasing use of agents for different applications, increasing agent decentralization and need for agent communication and interoperation, such flexibility is essential. This has been recognized in recent security literature[7]. 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. In this context, our research effort at developing a way to flexibly implement decentralized PKI is also a basic and critical step for decentralization of trust management [12].

Another direct motivation of our research is that 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. The new generation of software, agents, 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[14]. Agent-oriented authentication verification services must be supplied for most agent-based applications. In fact, agents as primarily human-delegated software, will be an ideal application domain of

[1] In fact, only the public key and a signature are the essential parameters that must always be present in a certificate.

modern cryptography in the very near future.

Treatment of security in the agent literature has been very scant. [17, 14] discuss some issues from the point of view of cryptography. In particular, [17] discussed some useful principles, which, although well-known in the security community, could be useful to agent developers. For example, an agent developer could understand that any design which depends on secrecy of the design is guaranteed to fail and that the public cryptographic algorithms are the right approach for agent security. In [14] 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, without a scalable authentication service, all of security schemes and protocols designed for open agent society cannot make any sense.

Further more, security protocols, operations and interoperation between principals (agents), as well as public key management are really difficult burden 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 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 KQML — we propose a new ontology, several new parameters and new performatives that are necessary for public key management and secure communication among security agents and application agents. Such performatives and parameters are not currently available in the KQML specification document[2] or in KQML implementations.

2 Security Agent

Existing PKI implementations began with specifying their certificate formats and the name spaces through a pre-defined hierarchies, such as the DNS name hierarchy. This method entails inflexible implementation. In our KQML-based PKI, instead of specifying the format of certificates, name space or hierarchy structure, we are applying agent concepts and technology to *authorities of authentication service*, and developing a *security agent*. 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. What we mean by "configurable facilitator" 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[2].

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).

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 [14]. We propose a security extension of KQML in this paper and will discuss it in next section, section 3. Succinctly speaking, our extension enable agents to identify multiple certificates and cooperatively conduct security interoperations.

2.1 Function Modules and Architecture

In order for a security agent to manage public key certifications, it must be capable of performing a basic set of tasks. Although there are some difference depending on specific hierarchies, the tasks and principles are basically the same: 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. Here, however, as an initial step, only the very basic tasks are discussed, so that we can more clearly sketch the contours of a security agent's structure and functionality.

Every task the security agent performs involves communication with other agents. Therefore, a security agent needs security protocols for agent communication as well as an internal databases to store local secret information. Although when a task is carried out, several functions may be performed asynchronously, we still functionally split the system into several components, named modules, with clear boundaries so that we can easily explain how a security agent works.

The security agent architecture is based on the agent architecture we have developed in the RETSINA multiagent infrastructure[13][3]. Every RETSINA agent has the following modules: communicator, planner, scheduler, and execution monitor.

We give a brief overview of the general processing of a message by a RETSINA agent. The modules of a RETSINA agent are implemented as Java threads and operate asynchronously. However, for simplicity of presentation we present their steps sequentially.

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. 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.

Figure 2.2 shows the relationships and data flow among the security agent's functional modules. The modules in the current implementation of the security agent are as follows:

1. Communicator: it deals with communications with other agents, including security agents or application agents. More pre-

[2]Certificate formats in existing PKI implementations can be adopted if they are suitable for an application.

[3]The RETSINA project URL is http://www.cs.cmu.edu/ softagents

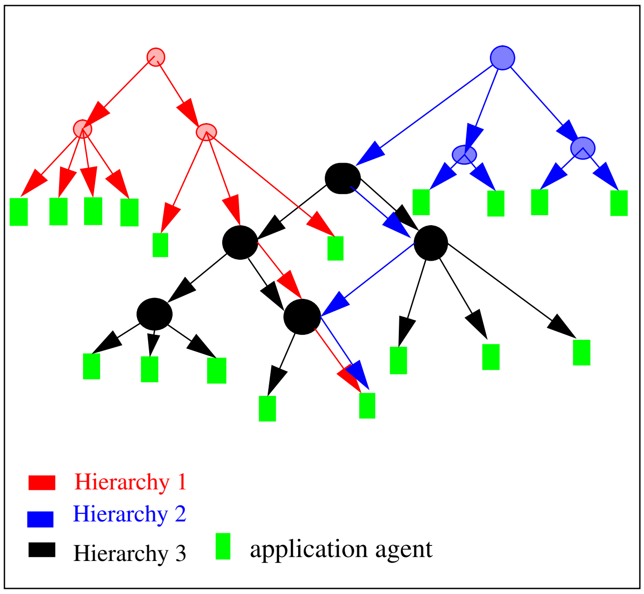

Figure 2.1 Multiple Hierarchies across a agent.

cisely, 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.

Sometimes a message could be a cipher, an encrypted message. In this case, the parser must recognize that the message is encrypted. It organizes into a task object and sends it to the planner. In some circumstances, these procedures may necessarily be repeated several times, back and forth, for example, if the original KQML message included embedded KQML messages. Similarly, outgoing messages also arrive at the parser from the agent planner or execution module. To recursively wrap an outgoing message as a KQML package or to send out a message in secure way, the outgoing message is processed by the parser. The message is finally sent out through the communicator.

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 the security agent is an agent whose services are used by many other agents, it needs to prioritize and schedule its requests for security services that 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 and verify a message.

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

   (a) design and generate public key certificates according to their applications. Through the interface, the users 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 - The procedure of applying or issuing a public key certificate is very important so that it must be done by a manual process since some judgement is required to evaluate the proposed evidence for the applicant agent owner (the user or organization which delegates the applicant agent to act on his/ her/its behalf).

   During the application procedure, the applicant needs the interface to talk with their agents about which security agent to apply for their certificates, which kind of certificate he wants. 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. During the procedure of issuing a certificate, the person who controls a security agent that issues the certificate also need the interface to verify the authenticity of the information of an application, then decide whether to validate a public key certificate for applicant 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 needed security protocols 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): There are two cases in which a security agent (even application agent) needs a CDB:

   (a) In a dynamic management of public key certification, when the agent applies for a certificate from a security agent, it will be given not only its certificate (in which the public key that has been automatically generated is included) but also 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. Then, 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 position in the name space.

   (b) To cut down communication costs, an agent (security agent or application agent) may cache some most frequently used certificates, i.e. certificates of agents it has frequent dealings with. When the agent needs to use one of the certificates, it doesn't have to communicate with any other security agent or its participant, but just looks up the CDB, gets the certificate for the particular agent and uses it. In both these cases, the CDB can reduce the overhead of communication mitigate the bottleneck in authentication service system, and simplify some secure communication protocols.

3 Extensions to KQML

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[2]. However, agent security issues were not taken into consideration in the original version of KQML specification. Some changes were made for secure communications based on KQML[14]. But it is still incomplete, especially since it does 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 new 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.

3.1 New parameters

The four new parameters are:

1. **:certificate**
   The certificate of the agent sending the message will be included as the value of this parameter in a performative. The format of the certificate depends on the certificate name included in the performative as the value of parameter "language". For example, if the name is SPKI, then the format will be: [5]

   ISSUER : a principal or a single top-level name in a principal's name space. The principal is identified as a public key or the hash of that key; the corresponding private key signs the certificate.

   SUBJECT : a principal, an object or a SDSI name reducible to either of those. The subject is the agent who receives authority from the issuer by way of the certificate.

   DELEGATION : the optional modifier, "(propagate)", giving the subject permission to delegate the authority presented in the certificate (or part of it) to some other Subject.

   AUTHORITY : the specific authorization(s) being delegated in this certificate.

   VALIDITY : date ranges and/or on-line validity tests for determining certificate validity.

   SIGNATURE : a digital signature signed by ISSUER.

2. **: certificateName**
   The value of this parameter will indicate the name of the certificate used in the performative, so that the agent receiving the KQML message will be able to parse the information as certificate.

3. **: signature**
   The value of this parameter is the sender's signature signed at the end of the content of the KQML message. This signature can be verified with the public key included as the value of the parameter certificate mentioned above.

4. **: certificateChain**
   For the dynamic management of certificates, the certificateChain, in which the certificates of the agents along the path from the root security agent through the agent that is the holder of the certificateChain, will be needed as parameter in the performative as mentioned in 2.1.(7).

3.2 New performatives

1. **apply-certificate**
   In order to securely communicate with others, when an agent is created, it will apply for a certificate in which a public key automatically generated 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 (with the interference of humans, see also 2.1 (5)), the security agent in charge of issuing certificates will send back a performative as follow:

   ```
   issue-certificate:
      :certificateName {name of
      certificate}
      :content {issued certificate}
   ```

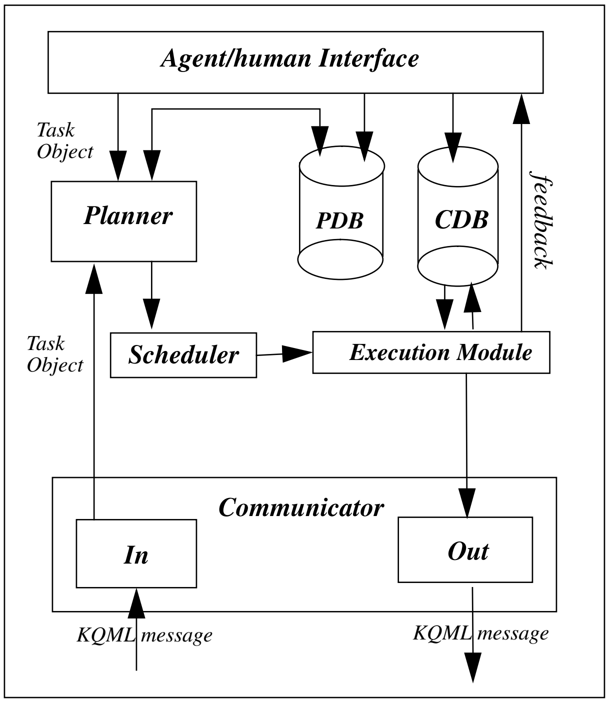

Figure 2.2 Structure of Security Agent.

```
:certificate {authority's
certificate}
[:certificateChain {the certificate
chain of authority}]
[:signature {signature signed by the
security agent}]
:ontology PKCertificate
```

Where the content of "certificateName" 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 is applying for its 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}
   :certificate {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}
   :certificate {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},
[:certificate {certificate}]
[:certificateChain {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, in the future more sophisticated mcertificate management is needed, additional performatives can be developed.

4 Conclusion

In this paper, we discussed an agent-based implementation of PKI. Unlike the traditional way of PKI implementation, we propose to implement the authorities of authentication verification service systems as personal 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. Two aspects of the implementation, the functional structure and communication language extension, were discussed. To summarize:

From the viewpoint of application of public key cryptosystem, our work:

1. Makes the construction of scalable authentication system much more feasible by employing the security agents in a bottom up fashion.

2. Makes interoperation of multi-certificate authentication system possible.

3. Can help customize certificate management while relieving the workload for certificate users.

From the viewpoint of agent applications,

4. The implementation will lay a authentication foundation for agent security. This is significant for application of agent technology especially in electronic commerce.

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

5 Future Work

With the KQML-based PKI, the software agents, including security agents and application will be able to efficiently manage their certificates, prove/verify the authentication of communications, and encrypt/decrypt messages. However, there are some remaining issues that we are going to address in future work.

Which kind of security policy can be specified so that security agents will automatically perform transmission of credentials among different authentication verification service systems?

How to define a suitable language for the users to describe their security policy and security protocols so that the agent delegates of a user can safely transact electronic business on his behalf?

Under what circumstances should a message, or part of the message, be encrypted, or signed, or signed and encrypted?

Now that we allow the users to define the formats of their certificates and management protocols, it is necessary for agents to be able to check the correctness and robustness against attacks. This is another important future work. Interested readers may like to refer[15][16].

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

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

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

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

[6] 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

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

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

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

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

[11] Bruce Schneier,*Applied Cryptography*, Second Edition, John Wiley and Sons, Inc., 1996.

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

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

[14] 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.

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

[16] Nevin Heintze, Doug Tygar, Jeannette Wing, and Hao-Chi Wong, Model Checking Electronic Commerce Protocols, Proc. of the USENIX 1996 Workshop on Electronic Commerce, November 1996.

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