Adding Security and Trust to Multi-Agent Systems*

H. Chi Wong and Katia Sycara
Carnegie Mellon University
Pittsburgh, PA, 15213
hcwong@cs.cmu.edu, katia@cs.cmu.edu

Abstract

Multi-agent systems (MASs) are societies whose individuals are software-delegatees (agents) acting on behalf of their owners or delegators (people or organizations). When deployed in an open network such as the Internet, MASs face some trust and security issues. Agents come and go, and interact with strangers. Assumptions about security and general trustworthiness of agents and their deployers are inadequate in this context. In this paper, we present the design of a security infrastructure applicable to MASs in general. Our design addresses both security threats and trust issues. In our design, we have mechanisms for ensuring secure communication among agents and secure naming and resource location services. And two types of trusts are addressed: trust that agents will not misbehave and trust that agents are really delegatees of whom they claim to be. To establish the first type of trust, we make deployers of agents liable for the actions of their agents; to establish the second type of trust, we propose that agents prove that they know secrets that only their delegators know.

1 Introduction

Agents are software entities that have enough autonomy and 'intelligence' to carry out various tasks with little or no human intervention. They are software delegates of individuals and organizations, and can act on behalf of their delegators. In multi-agent systems (MASs), agents interact with other agents, non-agent software, and humans. When deployed in wide-area networks, such as the Internet, agents can access remote service providers, search for information on the web, and carry out sale transactions. Agent-mediated electronic commerce is seen as a major application area of agent technology [15].

Open-network MASs face some security and trust issues, however. Agents come and go, and interact with strangers. Assumptions about general trustworthiness of agents and their deployers are inadequate in this context. Instead, we should assume that agents are not trustworthy and can misbehave, unless there is reason to believe otherwise. For examples, agents may eavesdrop on message exchanges between two other agents; they may masquerade as some other agent; or they may not comply with their obligations, e.g., an agent may not want to pay for goods received. MASs should provide mechanisms that their agents can use to defend themselves against attacks of other agents and humans.

The goal of our research is to identify some security and trust issues faced by MASs, and to design a security infrastructure applicable to MASs in general. This leads us to focus on application-independent issues.

*This research has been supported in part by DARPA contract F30602-98-2-0138, and by ONR Grant N00014-96-1222.

To be concrete, the discussions in this paper are based on Retsina [20], a reusable multi-agent infrastructure for building agents with capabilities of inter-agent, message-passing communication. The Retsina infrastructure can be used by agent developers to quickly develop agents for different applications. Retsina was developed having Internet applications in mind. We use Retsina as an example because it is the most general and flexible MAS infrastructure known today.

We take the following steps to achieve the goal of introducing security and increasing levels of trust in MASs. First, we identify some of the issues that MAS applications face in general. Then we propose solutions for these problems. Finally, we design a security sub-system that embeds our solutions. An implementation is in progress.

Whenever applicable, we leverage on well-known mechanisms from the security literature, and avoid 'reinventing the wheel'.

The paper is organized as follows. In Section 2, we describe the Retsina model of MAS. In Section 3, we identify some security threats in MASs. In Section 4, we propose solutions to counteract the security threats discussed in Section 3. In Section 5, we lay out a design of a security infrastructure in which the solutions proposed in Section 4 can be implemented. In Section 6, we discuss related work, and in Section 7 our contributions and future work.

2 The Retsina model of MAS

In Retsina, there are three classes of agents: interface, task, and information agents. Interface agents interface humans with the rest of the system. They may have different degrees of automation: totally manual, semi-automated, or completely automated. It is through them that humans delegate tasks to the agent system. Interface agents are typically *users* of the services provided by other agents of the system. Task agents are agents that can perform specific tasks. An agent that advises on stock buying and selling is an example. They are typically users of information agents, and *providers* of services to interface agents and other task agents. Information agents are associated with information sources, such as stock quotes from the NYSE. They provide information to other agents in the system.

At a higher level, Retsina agents are bound to each other by producer-consumer relationships. Given a Retsina application, there are a number of services. Portfolio keeping and stock buy/sell advising are examples of services in a portfolio management application. Services are provided by service providers and used by consumers. A portfolio-keeper is a provider for portfolio-keeping service. Consumers of this service include stock buy/sell advisors and interface agents. As expected, a given agent can be a provider of one service and a consumer of another.

Underlying a Retsina MAS, there are two types of infra-structure entities: ANSs (agent name servers), and matchmakers [2]. ANSs keep a mapping of agent ids to agent's physical addresses, and are queried whenever an agent plans to contact another agent whose address it does not know about. Matchmakers keep a mapping of agent ids to agent capabilities. When an agent first comes up, it advertises its capabilities with a matchmaker, who stores the advertisement in its advertisement database. Matchmakers are queried when an agent needs a service from the system, and does not know who to contact. By matching requests to stored advertisements, a matchmaker can locate service providers for interested consumers. A language for advertisement and requests, called LARKS, that allows agent locating and interoperation has been developed [10].

Retsina has been instantiated to several applications [20], including a visitor hosting system [19], aircraft maintenance [16], and financial portfolio management [18].

3 Some Security Threats in MASs

Intuitively, adding security and trust to MASs is about increasing users's confidence that a system of agents will handle tasks the way they (the users) expect them (the agents) to. In a insecure MASs, there are different types of threats that can undermine the users's confidence. We identify a number of them, which we list below. In what follows, matchmakers are optional entities: Retsina uses them; other MASs may not [1].

- Corrupted naming and matchmaking services[2]:

  Naming and matchmaking are two critical services in a MAS system. Application agents depend on ANSs to locate other agents, and matchmakers to find service providers. Naming and matchmaking services can be corrupted by untrustworthy ANSs and matchmakers. For example, a misbehaving ANS may map an agent id to an address other than the one given by the registering agent, and a misbehaving matchmaker may arbitrarily delete an advertising entry. They can also be corrupted by untrustworthy application agents, who can for example unregister their competitors at the ANS and the matchmaker.

- Insecure communication channels:

  Since agents in MASs are distributed, secure communication is of paramount importance. Agents should know, with certainty, that they are talking to agents they think they are talking to (authentication); that messages they receive are not corrupted during transmission or injected by some third party (integrity); that communication is private (confidentiality) if they choose so; and sometimes that nobody can deny having sent a message they have sent (non-repudiation).

  Secure communication can be compromised by any agent in a MAS. Agents can spoof each other, listen in to third party conversations, alter messages in transit, and repudiate wha they have said.

- Insecure delegation:

  Agents act on behalf of their deployers. In certain applications, we may want to know that agents are in fact delegates of whom they claim to be. For example, before giving out my account information to an agent, my bank would want to know that the agent is in fact my delegatee. Unless some secure delegation mechanism is in place, an agent can always impersonate as a delegate of someone who did not entrust to it.

- Lack of accountability:

  In MASs deployed in open networks, where agents interact with strangers that come and go, there is no reason for agents to trust each other. Deployers may be dishonest, software may be buggy; agent wrongdoings can certainly happen. If no one is held accountable for potential problems caused by misbehaving agents, then users will hesitate to use the technology.

[1] However, if a MAS operates in an open environment where agents may appear and dissapear, it needs some sort of middle agent (e.g., a matchmaker or a broker) to allow a requester to locate a service provider.

[2] A naming service is not corrupted, or satisfies integrity if its entries correctly map agents to their physical addresses. In such a service, two agents can never be mapped to the same address. A matchmaking service is not corrupted, or satisfies integrity if its entries correctly map agents to what the agents themselves claim are their capabilities. Note that agents can lie about their capabilities; but this is a different issue.

4 Adding Security and Trust to MAS

We propose the following solutions to counteract the threats discussed in Section 3.

- Use trusted ANSs and matchmakers:

  ANSs and matchmakers that behave as they should are critical in guaranteeing the integrity of naming and matchmaking services. ANSs and matchmakers should

  - Service only valid requests. A request is valid if it comes from a rightful requester (someone that requests service for oneself), and the request itself 'makes sense' in the current state of the ANS or the matchmaker. For example, a request for unregistering an agent at a matchmaker is invalid if the agent is not currently registered.
  - Insert (remove) an entry in their address/capability databases in a way that is consistent with the request; and
  - Give responses that are consistent with the content of their databases, when queried.

- Make agents uniquely identifiable, and give them unforgeable proofs of identity:

  Spoofing can be prevented if agents have unique ids, and are required to show proofs of identity when interacting with other agents. In particular, ANSs and matchmakers can use these ids to prevent agents from registering and unregistering someone other than themselves.

- Protect communication channels:

  All messages should be authenticated and any corruptions should be detectable by their recipients. Also, communicating parties should be able to make their communication confidential.

- Make agents prove that they are delegates of whom they claim to be:

  Whenever the identity of an agent's delegator matters (e.g., when application-level services are restricted by access control), the agent should be able to prove that it is indeed the delegatee. One way of achieving this is to have delegatees show knowledge of secrets that only their delegators know about. For instance, my banking agent should show knowledge of my PIN.

- Make deployers of agents liable for the actions of their agents:

  MAS on the Internet are intended to be open systems. Anyone is allowed to join and to leave freely. Due to the openness and the expected scale of such systems, it is infeasible to ensure that only trustworthy agents are allowed in the system.

  If we make deployers liable for the actions of their agents, then they are less likely to deploy buggy agents, and to initiate cheating themselves. Also, when misbehaviors occur, we can hold someone accountable, and apply punishment, or demand indemnification. This can indirectly increase the level of trust users have on MASs.

  To establish this liability, and to make deployers aware of it, we require that agents be given proofs of identity (without which they could not interact with other agents) only when their deployers allow their own identities to be linked with those of their agents.

5 A Design of a Security Infra-structure for MASs

In this section, we propose a security infra-structure for MASs, that takes into account the guidelines in Section 4. It requires adding new modules and new protocols to existing MASs. For concreteness, we use Retsina as our running example here.

5.1 Assumptions

We assume that deployers of agents are people or organizations that have public key certificates binding their 'physical' identities (social security numbers, company names, etc) to their public keys. They also have the private keys corresponding to the public keys in their public key certificates. Deployer certificates are issued by Deployer Key Certificates (DCAs). DCAs are assumed to exist, but they lie outside our security infra-structure. ([9] discusses one possible approach of incorporating them into our system.)

The ANSs and the matchmaker are trusted agents whose ids (or names) and public keys are publicly known. The addresses of the ANSs are also publicly known.

5.2 Architecture

We describe the different components of our architecture in a way that maps them to the solutions they implement.

In what follows, D denotes agent deployers. $A \rightarrow B : m$ denotes A sending message m to B. Given an agent X (or agent deployer D), x denotes X's private key, and $pub(x)$ denotes the public key corresponding to x. m_i's denote messages. Given a message m, and a private key x, $(m)_x$ denotes m signed with key x.

5.2.1 Uniquely Identifiable Agents and Liability

In our system, we use public key cryptography to make agents uniquely identifiable. Agents are given a public key certificate and a matching private key. An agent proves its identity by signing with its private key. Such a signature is valid only if the correspondent public key is certified.

In our design, only deployers of agents are authorized to request certifications for their agents, and certifications are not granted unless the deployer proves its own identity. This requirement binds an agent id to its deployer's id, and makes the deployer aware of his or her liabilities.

Our security architecture requires adding one new module to the Retsina framework. It is a key certification authority used to certify the binding of agents's ids to their public keys. We call this certification authority ACA (Agent Certification Authority) to distinguish it from DCA.

ACA is the certification authority for agent keys and is part of our infrastructure; DCA is the certification authority for deployer keys and lies outside our infrastructure.

The following protocols are used for agent key certification and revocation. These protocols lie outside the realm of Retsina, in the sense that D is not (required to be) a Retsina agent.

Certifying An Agent's Public Key

Before starting up an agent, a deployer chooses an id (name) for the agent, generates a public key pair, and gets the public key certified by the ACA. Fig. 1 shows a high-level description of a protocol for agent key certification. Note that all protocols discussed in this section are to be used in conjunction with SSL, a security protocol from Netscape we discuss in subsection 5.2.3. Thus, all communicating parties have public key certificates of each other, and all the message exchanges in these protocols can satisfy authentication, integrity, and confidentiality.

D : – chooses an agent id *ag-id*;
– generates a public key pair $\{a, pub(a)\}$;

$D \to ACA$: m_1, where $m_1 = (certify\ [ag\text{-}id, pub(a), timestamp])_d$;

ACA : – Is the request valid?
. is *timestamp* recent?
. can the signature in m_1 be verified with the public key in D's certificate?
. has $pub(a)$ been used before?
– generates m_2, where m_2 = a ACA-signed certificate binding $pub(a)$ to *ag-id*;
– creates an entry [*D's public key certificate*, m_1] in the certification database;

$ACA \to D$: m_2

D : – verifies the signature in m_2 with the public key in ACA's certificate;
– stores the certificate.

Figure 1: Agent key certification protocol.

After getting the certification request (m_1), ACA verifies the validity of the request[3]. A request is valid if it has been recently generated (the timestamp[4] shows when the request was generated, which must be within a δ of ACA's current clock time), the signed message can be verified[5] using D's public key, and $pub(a)$ has not been used (ACA checks both its certification and revocation databases).

If the request is valid, then ACA generates a signed certificate m_2 binding $pub(a)$ to *ag-id*. With this certificate, agent *ag-id* can prove its identity by signing with the private key a. Before sending the certificate to D, ACA creates the entry [*D's public key certificate*, m_1] in its certification database. This entry allows ACA to map an agent back to its deployer. ACA does not, however, reveal the id of the deployer of an agent unless 'cheating' occurs, and the deployer needs to be tracked down for accountability.

D checks ACA's signature upon receiving m_2, because only ACA-signed certifiates are accepted in Retsina.

Revoking An Agent's Public Key

To revoke an agent's key, D engages in a key revocation protocol (Fig. 2) with ACA.

ACA only processes a revocation request if it is valid (a revocation request is valid if it comes from the same deployer that requested certification). ACA then removes *ag-id*'s certificate from the certification database, and adds it to the revocation database. The revocation request is also kept in the

[3]Note that unlike in other protocols such as SPKI [6, 5] and PKIX [14], our certification request message m_1 does not prove D's possession of private key a. This is an issue only if a is under some other deployer $\overline{D}$'s possesion, who happens to also have d (in this case, $\overline{D}$ can deploy an agent A with key a, and have D held accountable for A's actions). But $\overline{D}$ would have d only if D lends it or d is compromised.

[4]The use of timestamps to guarantee freshness of messages requires that agents have synchronized clocks. In MASs that do not have synchronized clocks, nonces can be used instead.

[5]Abstractly, a signed message $[m]_x$ can be verified with a public key y if $y = pub(x)$.

$D \rightarrow ACA:$ m_3, where $m_3 = (revoke\ [ag\text{-}id\text{'s key certificate}])_d$;

$ACA:$ – Is the request valid?

. Does the public key in D's certificate have the authority to request the revocation of the key?
(It does if it is the one used in the certification request.)

. Can the signature in m_3 be verified with the public key in D's certificate?

– move the entry corresponding to *ag-id* from the certification database to the revocation database;

– append m_3 to the newly added entry in the revocation database;

$ACA \rightarrow D:$ m_4, where $m_4 = (revoked\ [ag\text{-}id,\ pub(a)])_{aca}$.

Figure 2: Agent key revocation protocol.

revocation database for possible future disputes. For example, D may claim that it has not requested the revocation.

5.2.2 Trustworthy Naming and Matchmaking Services

To achieve integrity of naming and matchmaking services, we extend existing protocols in Retsina. We also need to assume that ANSs and matchmakers are trusted entities, in that they will follow the extended protocols described below.

Registering, Unregistering, and Looking up Addresses at an ANS

To make the Retsina naming service secure, current address registration, unregistration, and lookup protocols need to be extended. Figures 3, 4 and 5 present the Retsina registration, unregistration, and lookup protocols with security added. In these protocols, we assume that principals have public key certificates of each other.

To register its address, an agent A uses the address registration protocol (Fig. 3).

A submits a signed request to an ANS. Upon receiving a registration request, ANS verifies the validity of the request. A request is valid if it satisfies the following conditions. 1) The requester is registering itself (*aid* appears in A's certificate, and the signature in m_1 can be verified by the public key in A's certificate). 2) Agent A has not been registered (*aid* has not been registered with the same public key). 3) *addr* is not occupied by other agents. And 4) the request has been generated recently, and the signed message has not been used for a registration that has been unregistered (ANS checks its unregistration database for such replays).

If the request is valid, then ANS tries to confirm that A is in fact located at *addr*. It does so by sending a nonce n (challenge) to address *addr*, and requiring the message $n + 1$ (response) in return. A will not be able to receive n (and consequently return $n + 1$) unless it is running at *addr*. This challenge and response protocol prevents denial-of-service attacks, where an agent tries to register all, or a substantial number of the available addresses, or even a targetted attack with a single address.

If A responds to ANS's challenge successfully, ANS registers A (it saves a copy of the request for possible future disputes), and returns a confirmation. The confirmation indicates expiration time, after which A is automatically unregistered. To stay registered, A must submit a new registration before the expiration time. Keeping addresses at an ANS for limited time period takes care of agents that

$A \rightarrow ANS:$	m_1, where $m_1 = register\,[aid,\ addr,\ timestamp]_a$, *aid* is the agent id, and *addr* is the physical address of A.
$ANS:$	– Is the request valid? . does *aid* appear in A's certificate? . has *aid* been registered with the public key in A's certificate? . has *addr* been taken by another agent? . is *timestamp* recent? . can the signature in m_1 be verified by the public key in A's certificate? . has the signed message being used before?
$ANS \rightarrow A(addr):$	n, where n is a nonce;
$A \rightarrow ANS:$	$n + 1$;
$ANS:$	– If the request is valid, create the entry [m_1, *A's certificate*] in the address database;
$ANS \rightarrow A:$	m_2, where $m_2 = registered\,[aid,\ addr,\ timestamp',\ expiration]_{ans}$;
$A:$	– Is the confirmation valid? . can the signature in m_2 be verified by the public key in ANS's certificate? . is *timestamp'* recent?

Figure 3: Address registration protocol.

do not unregister before disappearing. Disappearing before unregistering at the ANS does not lead to security threats, since no illegal occupants of this address can impersonate the legal occupant: the illegal occupant does not have the right private key to prove that it is the legal occupant.

To unregister itself at an *ANS*, an agent uses the address unregistration protocol (Fig. 4).

Upon receiving the unregistration request, *ANS* checks its validity. An unregistration request is valid if the agent is registered, the request comes from the agent that requested the registration, and the request was generated after the agent has been registered. Some of the checks that ANS does before unregistering may actually help in detecting some abnormalities. For instance, if the submitted *aid* can not be found in the registration database, then it could be that the agent's key has been compromised, and someone took the agent down.

If the request is valid, then *ANS* moves the entry from the registration database to the unregistration database. The unregistration request itself is also added to the unregistration database. All this record keeping is intended for potential future disputes.

To look up an address, an agent A uses the protocol in Fig. 5.

Given that agents are uniquely identified by their public keys, A submits both an agent's id and its certificate when looking up the agent's address. Since address lookup is a publicly available service, A does not sign the request.

aid may be registered at this particular *ANS* or somewhere else. If it is not registered locally, *ANS* needs to contact the appropriate name server, using the same protocol. The lookup result is signed by *ANS*, for possible future disputes. For an address mapping obtained remotely, *ANS* keeps a copy of it as it was returned by its peer (with the appropriate signature), also for possible future disputes.

$A \to ANS:$ m_1, where $m_1 = \textit{unregister}\,[\textit{aid, timestamp}]_a$;

$ANS:$
- Is the request valid?
  - . Is there an entry e with *aid* and A's certificate in *ANS*'s registration database?
  - . Can the public key in e verify the signature in m_1?
  - . Is the time in *timestamp* more recent than the registration time?
- If the request is valid,
  - . move the entry e from the registration database to the unregistration database;
  - . Append the entry e just added to the unregistration database with m_1;

$ANS \to A:$ m_2, where $m_2 = \textit{unregistered}\,[\textit{aid, timestamp'}]_{ans}$;

$A:$
- Is the confirmation valid?
  - . Can the signature in m_2 be verified with the public key in *ANS*'s certificate?
  - . Is *timestamp'* recent?

Figure 4: Address unregistration protocol.

$A \to ANS:$ *lookup* (*ans-x.aid, aid's certificate*);

$ANS:$ if *ans-x* is self then
get *aid*'s address from its registration database
else *lookup* (*ans-x.aid, aid's certificate*) at *ans-x*;

$ANS \to A:$ m_1, where $m_1 = \textit{lookup-result}\,[\textit{ans-x.aid, aid's certificate, addr, timestamp}]_{ans}$

$A:$
- Is the response valid?
  - . is *timestamp* recent?
  - . can the signature in m_1 be verified by the public key in *ANS*'s certificate?

Figure 5: Address lookup protocol.

Registering and Unregistering Capabilities, and Looking up Service Providers at the Matchmaker

The workings of a matchmaker (*MM*) are analogous to that of an ANS. The modifications we introduce to the protocols here, that agents use to interact witht the matchmaker, are therefore very similar to those introduced for the ANS.

The MM-registration protocol is identical to the ANS-registration protocol, except that while a physical address may not be shared by more than one agent, capabilities may. This difference is reflected in the validity check of a registration request. Another difference is the name agents use to register with the matchmaker; they are of the form ANSname.Agentname.

The update protocol is identical to the registration protocol, except that the requesting agent must already be in the 'directory' (registration) database. This difference is reflected in the validity check of an update request.

The lookup protocol is shown in Fig. 6. Note that since agent ids do not identify agents uniquely, the matchmaker returns both the agent id and the agent certificate for each agent.

$$A \rightarrow MM: \quad m_1, \text{ where } m_1 = \textit{recruit capabilities};$$

$$MM \rightarrow A: \quad m_2, \text{ where } m_2 = [\textit{capabilities}, ([\textit{ans-x1.aid1, cert1}], \ldots, [\textit{ans-xn.aidn, certn}]), \textit{timestamp}]_{mm};$$

A : – Is the response valid?
. can the signature in m_2 be verified by the public key in *MM*'s certificate?
. is the timestamp recent?

Figure 6: Service provider lookup protocol.

Finally, the MM-unregistration protocol is identical to the ANS-unregistration protocol.

Even though MM- and ANS-protocols are very similar in terms of message exchanges, they differ in their requirement of confidentiality. Agent deployers may not care that someone eavesdrops their agents's communication with an ANS, and finds out where the agents can be found (although some agent deployers may want to keep this confidential too). But it is more likely that they will not want eavesdroppers to find out what their agents do, specially if the agents deal with "sensitive" tasks or information. Thus, we should be able to make all message exchanges between an agent and the MM confidential.

5.2.3 Secure Communication Channels

To realize communication security in Retsina, we plan to layer the SSL (Secure Socket Layer) protocol [8] underneath the agent communication layer.

Two principles guided us towards this decision. First, security should be layered in our system, i.e., details related to communication security should be kept transparent from the application. Second, since it is hard to get cryptographic protocols right, both at the design and the implementation levels, off-the-shelf trustworthy technology should be prefered over home-brew products.

In addition to providing transparency, and trustworthiness (SSL has been carefully analyzed [22]), SSL provides extensibility. Its framework allows new public key and bulk encryption methods to be incorporated. And as new methods are incorporated, they are readily available to applications that use it.

Adding SSL to Retsina is a localized effort: it only requires incorporating SSL into the Communicator module in Retsina. The Communicator is the module that implements all inter-agent communication, and it was designed and implemented with interoperability in mind. As a result, it can be flexibly adapted to support different inter-agent communication languages (other than KQML), and is a module that can be used as a plug-in by other agent infrastructures. By incorporating SSL to the Communicator, we are building a secure communication module that can be used by other agent systems, talking in whichever language they please.

6 Related Work

Even though agents have been the subject of intense research for some time, only recently have researchers in the area examined security in the context of agents and multiagent systems ([15, 7, 21, 9, 13]).

In [7], a security architecture was proposed for Yenta, a matchmaking agent designed to find people with similar interests on the Internet and introduce them to each other. The main security concern in Yenta is privacy, and the main issues addressed in [7] are traffic analysis, gathering of one's private data by other Yenta agents, and how to find out the profile of the real person behind a Yenta agent (in case a user decides to reveal his or her identity to another person). For communication security, the author suggested using PGP-based schemes, but did not provide details.

In [21], KQML is extended with performatives and parameters that allow an agent to manage communication security at the application level. Agents can use appropriate performatives to authenticate each other, and appropriate parameters to specify details of cryptographic algorithms and keys used for each message. This approach sharply departs from ours in that all the details of securing communication channels are exposed to application agents.

[9] proposed an agent-based public key infrastructure (PKI). In this proposal, specialized agents called security agents are used as key certification authorities. These agents can handle multiple certificate formats and trust hierarchies. The goal is to enable interoperability of agent systems using multiple PKIs.

[17] is a framework and set of tools that can be used for developing multiagent systems. It provides mechanisms that enable agents to authenticate each other and to establish encrypted communication channels. The design of the security component, however, is not provided.

In [13], the focus is on secure marketplaces for mobile Java agents. In addition to security issues specific to mobility, [13] discusses some of the issues we cover here. The discussion there, however, stay at a fairly high level. In fact, they only suggest possible approaches to solve the problems. For example, they vaguely suggest the necessity of having different kinds of ids for agents and their deployers. But they do not provide further details.

Outside of agent research, however, there have been extensive research on security, specially as applied to distributed systems ([11] is such an example). And ever since the Internet (and later on the World Wide Web) showed promises of becoming universally accessible, there have been work on different aspects of security systems that are intended to be deployed in global scale. X.500 name directory [1], secure DNS [3, 4], and public key infra-structures such as PKIX [14] and SPKI [6, 5] are some of the examples. Our design has been inspired by them.

7 Contributions and Future Work

We have identified a number of security threats faced by MASs in general, and proposed a security architecture that counteract these threats and increase the level of trust one can have on a MAS.

In our security architecture, communication security is dealt with by giving agents unique ids, and transparently layering SSL underneath the communication interface used by agents. To guarantee the integrity of system-level services (such as naming and matchmaking services), we rely on unique ids of agents and add access control mechanisms to these services (we assume that ANSs and matchmakers are trusted entities in the system). Finally, to establish accountability for what agents do, we use a certification mechanism that requires deployers to register their agents before the deployment. This mechanism links deployers to their agents for the purpose of accountability.

The building blocks we use in our architecture are well-known in the security literature. In fact, our goal was to use off-the-shelf, well-understood, and trustworthy security technology, and add security to MASs in a way that is as transparent as possible.

A Java implementation of our design is underway, and will be added to the Retsina infrastructure.

In this paper, we proposed making delegations secure by having delegatees show knowledge of secrets that only their delegators know. This can be accomplished straightforwardly by allowing delegatees to have access to these secrets. This solution is not satisfactory, however, because nothing prevents a delegatee from misusing a secret. We are working on coming up with a more secure mechanism for doing delegation.

So far, we have assumed that agents are able to decide whether a certificate from a DCA is valid. Agents may not accept a certificate as valid because there may be multiple DCA's, not all of which are trusted by all agents. This problem is general to any public-key-based system, and we are investigating how to deal with it in the context of MASs. We believe that [9] may offer solutions, but the topic requires further research. Also, we have assumed a single ACA in our system. We plan to looking into having multiple distributed ACAs.

Finally, we would like to find other security issues faced by MASs in general. We plan to do this by examining concrete Retsina applications in different domains. But even within this paper, there is an issue that we have not brought to the foreground; and it has to do with "honesty" of agents themselves. Briefly, even if agents are who they say they are, and are legitimate delegatees for tasks at hand, they may still not carry out a task or a transaction as expected. This is so because software may be buggy or the agents themselves may be cheating. In our current system, such failures cannot be prevented, but only dealt with by holding deployers of "dishonest" agents accountable for the wrongdoings of their agents. A solution that can prevent such failures from occurring in the first place would be desirable.

References

[1] D Chadwick. *Understanding X.500 - The Directory.* Chapman and Hall, 1996.

[2] K. Decker, K. Sycara, and M. Williamson. Middle-agents for the internet. In *Proceedings of the Fifteenthe International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 578–584, Nagoya, Japan, August 1997.

[3] D. Eastlake. Domain name system security extensions. IETF Network Working Group RFC 2065, January 1997.

[4] D. Eastlake. Secure domain name system (dns) dynamic update. IETF Network Working Group Internet Draft: draft-ietf-dnssec-update2-00.txt, August 1998.

[5] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. Spki certificate theory. IETF Internet Draft: draft-ietf-spki-cert-theory-04.txt, November 1998.

[6] Carl Ellison. Spki requirements. IETF Internet Draft: draft-ietf-spki-cert-req-02.txt, October 1998.

[7] Leonard N. Foner. A security architecture for multi-agent matchmaking. In *Proceedings of the Second International Conference on Multi-Agent Systems*, pages 80–86, 1996.

[8] Alan O. Freier, Philip Karlton, and Paul Kocher. The ssl protocol version 3.0. March 1996.

[9] Qi He, Katia Sycara, and Tim Finin. Personal security agent: Kqml-based pki. In *Proceedings of the Second International Conference on Autonomous Agents*, May 1998.

[10] Sycara K., Lu J., , and Klusch M. Interoperability among heterogeneous software agents on the internet. Technical Report CMU-CS-92-131, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 1998. Technical report.

[11] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. In *ACM Trans. Computer Systems 10 (4)*, pages 265–310, 1992.

[12] Kay Neuenhofen and Matthew Thompson. A secure market place for mobile java agents. In *Proceedings of the Second International Conference on Autonomous Agents*, pages 212–218, May 1998.

[13] Kay Neuenhofen and Matthew Thompson. A secure marketplace for mobile java agents. In *Proceedings of the Second International Conference on Autonomous Agents*, pages 212–218, May 1998.

[14] Housley R., W. Ford, W. Polk, and D. Solo. Internet x.509 public key infrastructure certificate and crl profile. IETF Network Working Group RFC 2459, January 1999.

[15] Jeff Rosenschein, Tuomas Sandholm, Carles Sierra, Pattie Maes, and Rob Guttmann. Agent-mediated electronic commerce: Issues, challenges and some viewpoints. In *Proceedings of the Second International Conference on Autonomous Agents*, pages 189–196, May 1998.

[16] O. Shehory, K. Sycara, G. Sukthankar, and V Mukherjee. Agent aided aircraft maintenance. In *Proceedings of the Third International Conference on Autonomous Agents*, Seattle, Washington, May 1999.

[17] Rahul Sukthankar, Antoine Brusseau, and Ray Pelletier. Jgram: Rapid development of secure multiagent systems. In *submitted to Agents '99 – Third International Conference on AUTONOMOUS AGENTS*.

[18] K. Sycara, K. Decker, and D. Zeng. Intelligent agents in portfolio management. In N. Jennings and M. Woolridge, editors, *Agent Technology: Foundations, Applications, and Markets*, chapter 14, pages 267–283. Springer, 1998.

[19] K. Sycara and D. Zeng. Coordination of multiple intelligent software agents. *International Journal of Intelligent and Cooperative Information Systems*, 5(2 and 3):181–211, 1996.

[20] Katia Sycara, Anandeep Pannu, Mike Williamson, Dajun Zeng, and Keith Decker. Distributed intelligent agents. In *IEEE Expert*, pages 36–45, December 1996.

[21] Chelliah Thirunavukkarasu, Tim Finin, and James Mayfield. Secret agents – a security architecture for kqml. In *CIKM'95 Intelligent Information Agents Workshop*, December 1995.

[22] David Wagner and Bruce Schneier. Analysis of the ssl 3.0 protocol. In *Proceedings of the Second USENIX Workshop on Electronic Commerce*, pages 29–40, November 1997.