CHAPTER 4

SECURITY IN MULTIAGENT SYSTEMS AND AGENT COMMUNICATION

This chapter focuses the issues related to the issues security implications of MultiAgent System (MAS). Using these properties, the security issues for multiagent systems are identified. The security analysis presented in this chapter is based on properties of agents and open multiagent systems. We have also presented the security analysis for the narrow threat scenarios compared to the general security issues.

4.1 SECURITY MEASURES FOR MULTIAGENT SYSTEMS

This chapter describes the various security mechanisms that we have apply in our system to provide the required security.

4.1.1 Security Services for Multi Agent System

Based on the security analysis three broad categories of necessary security services for multiagent systems are identified, namely, (1) those addressing the communication between agents, (2) services protecting agents against malicious platforms, and (3) services protecting platforms against malicious agents. This chapter describes all of these three categories of security services applied in the system.
4.1.2 Security Measures for Agent Communication

Many commercial and research MAS architectures have been implemented and many are still under development. Several of these recognise security as an issue to be taken care of in the future, whilst others imply that security is provided for. It is common for MAS implementations to assume a Virtual Private Network (VPN) like underlying network to provide security services. This approach usually does not provide for much flexibility, since secure communication between parties without pre-established relationships becomes cumbersome. Nevertheless, this solution can use well established security protocols and be adequate for applications where all communication is protected to the same degree. Such an approach usually leaves the agents completely unaware of security services as this is handled between agent platforms. The agents themselves are also unprotected from malicious hosts if no other security measures are applied. The Transport Layer Security (TLS) is used to protect agent communication sessions against malicious hosts. The proposed implementation allows the agent to rely on the host for negotiation of security parameters, as well as the agent to supply its security parameters for the TLS sessions. FIPA is a non-profit standards organization that is developing standards for software agents to allow heterogeneous agent systems to interact. There are a growing number of agent projects, platforms and agent applications based on the FIPA standard. Earlier, today outdated, FIPA documents did consider some security issues. However, the current standards do not deal with security. FIPA has recognized this and recently initiated work in this area (Poslad and Calisti 2000).

Knowledge Query and Manipulation Language (KQML) is a message protocol designed to enable software agents to communicate with each other. The protocol has been developed as part of an Advanced Research
Projects Agency (ARPA). KQML does not deal with security issues but depends on security being provided by lower layers. Symmetric or asymmetric cryptography is supported and keys are assumed to be agreed beforehand. We propose an extension that provides confidentiality, authentication, and data integrity protection to KQML. However, this does not protect against message replay attacks. Hence we also provide a solution using mediating agents to enable communication with crypto un-aware agents. Another suggestion for enhancing KQML has been proposed by us which defines parameters for certificate management leaving the format of the certificate undefined.

4.1.3 Protecting Mobile Agents

Addressing the security threats that arise to mobile agents from potentially malicious host environments is recognised by the security community as a difficult but vitally important problem. As a result, there have been many attempts to address the threats posed to mobile agents, most addressing a particular part of the problem. As stated earlier, once an agent has arrived at a host, little can be done to stop the host from treating the agent as it likes. The problem is usually referred to as the malicious host problem. A simple example, often used to illustrate how a malicious host can benefit from attacking a mobile agent, is the shopping agent. An agent is sent out to find the best airfare for a flight with a particular route. The agent is given various requirements, such as departure and destination, time restrictions, etc., and sent out to find the cheapest ticket before committing to a particular purchase. The agent will visit every airline and query their databases before committing to a purchase and reporting back to the agent owner. A malicious host can interfere with the agent execution in several ways in order to make its offer appear most attractive. For example, a malicious host could try to: (1) erase all information previously collected by the agent, in this way the
host is guaranteed at least to have the best current offer; (2) change the agent’s route so that airlines with more favourable offers are not visited; (3) simply terminate the agent to ensure that no competitor gets the business either; (4) make the agent execute its commitment function, ensuring that the agent is committing to the offer given by the malicious host (if the agent is carrying electronic money it could instead take it from the agent directly). In addition to this, the agent might be carrying information that needs to be kept secret from the airlines (e.g. maximum price). There is no universal solution to the malicious host problem, but some partial solutions have been proposed. Many of the security mechanisms are aimed at detecting, rather than preventing, misbehaving hosts. In the following subsections we will describe some of the mechanisms proposed to address the malicious host problem.

4.1.3.1 Contractual agreements

The simplest solution is to use contractual means to tackle the malicious host problem. Operators of agent platforms guarantee, via contractual agreements, to operate their environments securely and not to violate the privacy or the integrity of the agent, its data, and its computation. However, to prove that such an agreement has been broken might be a non-trivial task.

4.1.3.2 Trusted hardware

If the operators of the available execution environments cannot be trusted, one obvious solution is to let a trusted third party supply trusted hardware, in the form of tamper resistant devices, that are placed at the site of the host and interact with the agent platform. A tamper resistant device can, for example, come in the form of a smart card. Such trusted hardware can then either protect the complete execution environment of the agent or
perform certain security sensitive tasks. However, such trusted hardware must be used carefully and might appear to offer more security than it really does. The agent must still be able to communicate with resources at the local platform (the part under control of an untrusted party), for example to interact with a local database. All such interactions can still be intercepted by the untrusted party. We note that, in the case where interactions with a database is required, techniques (e.g. Private Information Retrieval (PIR) (Chor et al 1998) have been proposed allowing retrieval of information without revealing what information is being retrieved.

If the trusted hardware is only used to protect security sensitive actions this might be even more vulnerable. It might, for example, be tempting to let the agent’s private signature key be protected such that it only will be available when decrypted inside the trusted device. A signature algorithm can then be executed within the device using the agent’s private key. In this way, the private signature key is never exposed to the host. However, the host might be able to interfere with the communication taking place between the agent residing on the host and the trusted device in such a way that a correct signature is produced on information falsely manufactured by the host. Above all else, the major drawback of trusted hardware is the cost of such a solution.

Two ongoing efforts, namely the Trusted Computing Group (TCG) and Next-Generation Secure Computing Base (NGSCB), have the goal of providing trusted computing sub-systems. These technologies appear to have the potential to provide a trusted environment for mobile agents. The functionality provided by these schemes includes, protected execution, protected storage, and platform authentication.
4.1.3.3 Trusted nodes

By introducing trusted nodes into the infrastructure, to which mobile agents can migrate when required, sensitive information can be prevented from being sent to untrusted hosts, and certain misbehaviours of malicious hosts can be traced. The owner’s host, i.e. the platform from where the mobile agent is first launched, is usually assumed to be a trusted node. In addition to this, service providers can operate trusted nodes in the infrastructure.

This approach does not appear to be fully explored elsewhere. The approach can potentially be very valuable in a mixed wireless and fixed network, allowing users to dispatch mobile agents into the fixed network, relying on trusted nodes for processing of ‘sensitive’ information. In our example with the shopping agent, the mobile agent can be constructed so that the commitment function (e.g. the agent’s signature key) is encrypted such that it can only be decrypted at a trusted host. Once the agent arrives at the trusted host it can compare the collected offers and commit to the best offer. Alternatively, one agent containing the ability to commit to a purchase can be sent to a trusted node. From this node one or several sub-agents are sent to the airline hosts to collect offers. Depending on the threat scenario, single hop agents can be used, that is agents only visiting one host before returning back, or one or several multi-hop agents can be used. Once the sub-agent or agents have returned to the trusted node, the best offer is selected and the agent commits to a purchase. This last alternative does limit the agent’s mobility, but may be beneficial in certain scenarios.
4.1.3.4 Co-operating agents

By using cooperating agents, a similar result to that of trusted nodes can be achieved. Information and functionality can be split between two or more agents in such a way that it is not enough to compromise only one (or even several) agents in order to compromise the task. An identical scenario to that described using trusted nodes can, for example, be achieved by letting the agent residing on the trusted host be executed on any host that is assumed not to be conspiring with any of the airlines. By applying fault tolerant techniques the malicious behaviour of a few hosts can be countered. One such scheme for ensuring that a mobile agent arrives safely at its destination has been proposed in Schneider et al (1997).

Although a malicious platform may cause an agent to operate incorrectly, the existence of enough replicates ensures the correct end result. Again, referring to the shopping agent, several mobile agents can be used, taking different routes, and before deciding on the best offer the agents communicate their votes amongst each other.

4.1.3.5 Execution tracing

Execution tracing has been proposed for detecting unauthorized modifications of an agent through the faithful recording of the agent’s execution on each agent platform. Each platform is required to create and retain a non-repudiable log of the operations performed by the agent while executing on the platform.

The major drawbacks of this approach are not only the size of the logs created, but also the necessary management of created logs. In addition, privacy issues are likely to arise when this kind of information is logged. Partial Result Authentication Codes (PRAC) were introduced by Yee. The
idea is to protect the authenticity of an intermediate agent state or partial result that results from running on a server. PRACs can be generated using symmetric cryptographic algorithms. The agent is equipped with a number of encryption keys. Every time the agent migrates from a host, the agent’s state or some other result is processed using one of the keys, producing a Message Authentication Code (MAC) on the message. The key that has been used is then disposed of before the agent migrates. The PRAC can be verified at a later point to identify certain types of tampering. A similar functionality can be achieved using asymmetric cryptography by letting the host produce a signature on the information instead.

4.1.3.6 Encrypted payload

Asymmetric cryptography (also known as public key cryptography) is well suited for a mobile agent that needs to send back results to its owner or which collects information along its route before returning with its encrypted payload to its owner. This is due to the fact that the encryption key does not need to be kept secret. However, to encrypt very small messages is either very insecure or results in a large overhead compared with the original message. A solution called sliding encryption has been proposed that allows small amounts of data to be encrypted, and consequently added to the cryptogram, such that the length of the resulting ciphertext is minimised. Due to the nature of asymmetric cryptography the agent is not able to access its own encrypted payload until arriving at a trusted host where the corresponding decryption key is available.

4.1.3.7 Environmental key generation

Environmental key generation allows an agent to carry encrypted code or information. The encrypted data can be decrypted when some
 predefined environmental condition is true. Using this method an agent’s private information can be encrypted and only revealed to the environment once the predefined condition is met. This requires that the agent has access to some predictable information source; several examples of such information sources are given. Once the private information has been revealed, it would, of course, be revealed to the executing host. However, if the condition is not met on a particular host, the private information is not revealed to the platform.

4.1.3.8 Computing with encrypted functions

Sander and Tschudin (1998) have proposed a scheme whereby an agent platform can execute a program embodying an enciphered function without being able to access the original function. For example, instead of equipping an agent with function \( f \), the agent owner can give the agent a program \( P(E(f)) \) which implements \( E(f) \), an encrypted version of \( f \). The agent can then execute \( P(E(f)) \) on \( x \), yielding an encrypted version of \( f(x) \).

With this approach an agent’s execution would be kept secret from the executing host as would any information carried by the agent. For example the means to produce a digital signature could thereby be given to an agent without revealing the private key. However, a malicious platform could still use the agent to produce a signature on arbitrary data. Sander and Tschudin (1998) therefore suggest combining the method with undetachable signatures.

Although the idea is straightforward, the problem is to find appropriate encryption schemes that can transform functions as intended; this remains a research topic. Barak et al (2001) have shown that obtaining theoretical justification for a program’s ability to completely hide its information appears infeasible.
4.1.3.9 Undetachable signatures

The idea of an undetachable signature is as follows. Suppose a user wishes to purchase a product from an electronic shop. The agent can commit to the transaction only if the agent can use the signature function s of the user. However, as the server where the agent executes may be hostile, the signature is protected by a function f to obtain \( g = s \circ f \). The user then gives the agent the pair (f, g) of functions as part of its code. The server then executes the pair (f, g) on an input x (where x encodes transaction details) to obtain the undetachable signature pair \( f(x) = m \) and \( g(x) = s(m) \).

The pair of functions allows the agent to create signatures for the user whilst executing on the server without revealing s to the server. The parameters of the function f are such that the output of f includes the user’s constraints. Thus m links the constraints of the customer to the bid of the server. This is then certified by the signature on this message. The main point is that the server cannot sign arbitrary messages, because the function f is linked to the user’s constraints.

An alternative to undetachable signatures proposed in this thesis is to use digital certificates to regulate the validity of digital signatures. Digital certificates are used to let a verifier check the validity of a digital signature. Certificates usually include a validity period under which valid signatures can be produced. By extending the constraints included in the certificate to context related values such as executing host, maximum value of a purchase, and so on, certificates can be used to further restrict the usage of signature keys and thereby decrease the involved risks regarding improper use of the signature key.

One advantage with this scheme over undetachable signatures is that it relies on already well-established cryptographic techniques.
4.2 AGENT COMMUNICATION LANGUAGE

Dogac and Cingil (2003) provide the following definition for Agent Communication Languages: ACL is a language that provides a set of application-independent primitives that allow an agent to state its intention in an attempt to communicate with other agents.

4.2.1 Need for Agent Communication Language

Agents may have numerous objectives and goals, and although they are individual and autonomous they still often have to collaborate and communicate with other agents in order to achieve their objectives. Dogac and Cingil (2003) state that in order to collaborate with others, agents are required to:

- Discover the existence, network addresses, capabilities and/or roles of other agents
- Communicate with other agents through an agent-independent, that is, a standard

Agent communication language. Nameservers and facilitators are provided by Multi Agent Systems to support agent discovery. Agents register their addresses to a name server and their features and abilities to a facilitator. Agents can then use those name servers and facilitators as a reference to find out abilities and addresses of other agents. Dogac and Cingil (2003) state that after the agents have discovered each other they need to communicate in order to achieve their goals. Communication can be divided into two fundamental parts: Pitkaranta (2004). First, agents need to agree on a common agent communication language (ACL) that provides the basis for stating intentions to other party Pitkaranta (2004). Second, mere common language is not enough but agents must also have common vocabularies for representing
shared domain concepts and application-dependent content. This includes both a shared ontology and the content that is defined by a Content Interchange Format (CIF). Therefore, agent communication languages basically consist of these three layers that are shown in Figure 4.1.

**Error!**

![Diagram of agent communication language layers](image)

**Figure 4.1 Parts of agent communication languages**

### 4.2.2 Theory of Agent Communication Languages

Speech act theory is the basis of most popular agent communication languages. Speech act theory was originally developed by linguists in an attempt to understand how humans use language in everyday situations Nwana and Wooldridge (1996). In speech act theory, human expressions are viewed as actions, in the same way as actions performed in the everyday physical world (e.g. picking up a block from table) Nwana and Wooldridge (1996). As a result of this, speech act theory uses the term performative to identify the intention behind spoken communication. Examples include verbs like request, tell or inform. Performatives are used as constraints of semantics in communication between agents, and they simplify how agents should react
to messages they receive (Poslad et al 2000). Two most successful agent communication languages so far have been Knowledge Query and Manipulation Language (KQML) and FIPA-ACL.

Knowledge Query and Manipulation Language is a language and protocol for exchanging information and knowledge. It is part of a larger effort, the Advanced Research Projects Agency (ARPA). Knowledge Sharing Effort (KSE) which is aimed at developing techniques and methodologies for building large-scale knowledge bases which are sharable and reusable. KQML is both a message format and a message-handling protocol to support run-time knowledge sharing among agents. KQML can be used as a language for an application program to interact with an intelligent system or for two or more intelligent systems to share knowledge in support of cooperative problem solving.

KQML focuses on an extensible set of performatives, which defines the permissible operations that agents may attempt on each other's knowledge and goal stores. The performatives comprise a substrate on which to develop higher-level models of inter-agent interaction such as contract nets and negotiation. In addition, KQML provides a basic architecture for knowledge sharing through a special class of agent called communication facilitators which coordinate the interactions of other agents. The ideas which underlie the evolving design of KQML are currently being explored through experimental prototype systems which are being used to support several testbeds in such areas as concurrent engineering, intelligent design and intelligent planning and scheduling.

KQML was one of the first initiatives to specify how to support communication of agents using a protocol based on speech acts theory. However, currently no single consensus or true de facto standard exist on
KQML specification. As a result, many different KQML “dialects” exist and different agent systems may speak slightly different language and not fully understand each other. KQML has following key features:

- KQML messages are opaque to the content they carry. KQML messages do not merely communicate sentences in some language but rather communicate an attitude about the content (assertion, request, query, basic response, etc.).
- The language primitives are called performatives.
- At the agent level, the communication appears as point-to-point message passing.

Special agents, called facilitators, may exits in KQML environment. Facilitators provide agents additional networking-related functions such as association of physical addresses with symbolic names, registration of agents and/or services offered and sought by agents, enhanced communication services such as forwarding, brokering and broadcasting. Poslad et al (2000) state that communication between agents involves three aspects: the method of message passing, the format or syntax, of the information being transferred, and the meaning, or semantics of the information and message. Any KQML message has the following syntax

(KQML-performative
 :sender <word>
 :receiver <word>
 :language <word>
 :ontology <word>
 :content <expression>
 ...)

A KQML message may have many different performatives. It depends on performative of the message what parameters, such as “sender”, “language” etc, are introduced in the actual KQML message. Table 4.1 introduces KQML performatives divided into several categories Nwana and Wooldridge (1996).

**Table 4.1 KQML performatives**

<table>
<thead>
<tr>
<th>Category</th>
<th>Reserved performative names</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic informational performatives</td>
<td>tell, deny, untell, cancel</td>
</tr>
<tr>
<td>Basic query performatives</td>
<td>evaluate, reply, ask-if, ask-about, ask-one, ask-all, sorry</td>
</tr>
<tr>
<td>Multi-response query performatives</td>
<td>stream-about, stream-all</td>
</tr>
<tr>
<td>Basic effector performatives</td>
<td>achieve, unachieved</td>
</tr>
<tr>
<td>Generator performatives</td>
<td>standby, ready, next, rest, discard, generator</td>
</tr>
<tr>
<td>Capability definition performatives</td>
<td>advertise</td>
</tr>
<tr>
<td>Notification performatives</td>
<td>subscribe, monitor</td>
</tr>
<tr>
<td>Networking performatives</td>
<td>register, unregister, forward, broadcast, pipe, break</td>
</tr>
<tr>
<td>Facilitation performatives</td>
<td>broker-one, broker-all, recommend-one, recommend-all, recruit-one, recruit all</td>
</tr>
</tbody>
</table>

In addition to the lack of agreed specification there are also some other problems related to KQML language. According to Poslad et al (2000), one of these problems is the lack of well defined semantics. The use of performatives alone is insufficient to guarantee that other agents will interpret messages correctly.

The problems of KQML are the driving forces behind the FIPA specification of its own agent communication language FIPA ACL. The purpose of FIPA ACL is to provide a standard way to package messages in
such a way that the meaning of the messages is clear to other compliant agents (Poslad et al 2000). FIPA ACL tries to achieve this goal by reducing the number of performatives. Although there are huge amount of verbs in English corresponding to performatives, the FIPA-ACL defines only the minimal set of these verbs for agent communication (it consists of approximately 20 performatives). This way more flexible approach for agent communication is achieved. Poslad et al (2000) state that FIPA ACL provides some benefits including:

- Dynamic introduction and removal of services.
- Customized services can be introduced without a requirement to recompile the code of the clients at run-time.
- Allowance for more decentralized peer-to-peer realization of software.
- A universal message-based language approach providing a consistent speech-act based interface throughout software (flat hierarchy of interfaces).
- Asynchronous message-based interaction between entities.

Table 4.2 introduces the FIPA ACL performatives, called communicative acts. As can be seen from the Table 4.2, FIPA ACL has 22 communicative acts, which is considerably less than what KQML has. FIPA ACL communicative acts can be divided to primitive and composite communicative acts (Pitkaranta 2004). Composite communicative acts can be composed of primitive communicative acts either by substitution or sequencing. An agent only has to implement the communicative acts it needs and the “not-understood” act (Pitkaranta 2004). However, performative is only one part of FIPA-ACL message, and the message also contains many other elements shown in Table 4.3.
### Table 4.2 FIPA-ACL performative set

<table>
<thead>
<tr>
<th>Accept proposal</th>
<th>Inform if</th>
<th>Refuse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agree</td>
<td>Inform Ref</td>
<td>Reject proposal</td>
</tr>
<tr>
<td>Cancel</td>
<td>Not understood</td>
<td>Request</td>
</tr>
<tr>
<td>Call for proposal</td>
<td>Propagate</td>
<td>Request when</td>
</tr>
<tr>
<td>Confirm</td>
<td>Propose</td>
<td>Request whenever</td>
</tr>
<tr>
<td>Disconfirm</td>
<td>Proxy</td>
<td>Subscribe</td>
</tr>
<tr>
<td>Failure</td>
<td>Query if</td>
<td></td>
</tr>
<tr>
<td>Inform</td>
<td>Query ref</td>
<td></td>
</tr>
</tbody>
</table>

### Table 4.3 FIPA-ACL message elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performative</td>
<td>Denotes the type of the communicative act of the ACL message</td>
</tr>
<tr>
<td>Sender</td>
<td>Denotes the identity of the sender (the name of the agent) of the message</td>
</tr>
<tr>
<td>Receiver</td>
<td>Denotes the identity of the intended recipient of the message</td>
</tr>
<tr>
<td>Reply-To</td>
<td>This element indicates that subsequent messages in this conversation thread are to be directed to the agent named in the reply-to-element, instead of the agent named in the sender element</td>
</tr>
<tr>
<td>Content</td>
<td>Denotes the content of the message</td>
</tr>
<tr>
<td>Language</td>
<td>Denotes the language in which the content element is expressed</td>
</tr>
<tr>
<td>Encoding</td>
<td>Denotes the specific encoding of the content language expression</td>
</tr>
<tr>
<td>Ontology</td>
<td>Denotes the ontology(s) used to give a meaning to the symbols in the content expression</td>
</tr>
<tr>
<td>Protocol</td>
<td>Denotes the interaction protocol that the sending agent is employing with this ACL message</td>
</tr>
<tr>
<td>Conversation-id</td>
<td>Introduces an expression (a conversation identifier) which is used to identify the ongoing sequence of communicative acts that together form a conversation</td>
</tr>
<tr>
<td>Reply-with</td>
<td>Introduces an expression that will be used by the responding agent to identify this message</td>
</tr>
<tr>
<td>In-reply-to</td>
<td>Denotes an expression that references an earlier action to which this message is a reply</td>
</tr>
<tr>
<td>Reply-By</td>
<td>Denotes a time and/or date expression which indicates the latest time by which the sending agent would like to have received a reply</td>
</tr>
</tbody>
</table>
4.2.3 Content Languages

As we have seen earlier, both the KQML and FIPA-ACL messages contain parts that are marked with content parameter. This content part of the message defines the actual information about the matter agents are trying to communicate. Agent communication languages typically define the outer language (ACL) and an inner language Content Interchange Format (CIF), as shown in Figure 4.1. The inner language, representing the content part of the message, allows an agent to express its actual application-dependent content to other agents. In the previous examples this was the cost of the service or the price offer for the movie Gladiator. Agents must understand each other in this content language level to be able to successfully interact with each other. Two content languages among the most popular ones Pitkaranta (2004) are presented here. They are Knowledge Interchange Format (KIF) and FIPA Semantic Language (FIPA SL). The syntax of both these languages is inherited from Lisp programming language.

4.2.3.1 Knowledge Interchange Format

The Knowledge Interchange Format (KIF) provides a high-level intermediate language for communication between agents. KIF provides for the encoding of simple data, constraints, rules, quantified expressions, metalevel information and procedures. KIF also provides the capability to define objects, functions, and relations related to knowledge representation. In this section of the thesis, we provide examples for representing simple data, called facts, with Knowledge Interchange Format:

(wage John 80 10)
(wage George 56 20)
These examples state that “John has worked for 80 hours with an hourly wage of 10 Dollars” and “George has worked for 56 hours with a hourly wage of 20 Dollars”. Second example enhances the use of more complex ways of representing knowledge and relations of information:

\[
(* \text{hours\_worked}
\text{Employee1})(\text{hourly\_wage}
\text{Employee1)})
\]

\[
(* \text{hours\_worked}
\text{Employee2})(\text{hourly\_wage}
\text{Employee2)})
\]

This example states that total wage of Employee1, which is calculated by multiplying the hours\_worked with his hourly wage is greater than the total wage of Employee2.

Dogac and Cingil (2003) state that there are some tools used to manipulate KIF encoded information. These include Prologic and Epilog, which are libraries of Lisp subroutines. In addition, there are also many Java-based KIF parsers available.

4.2.3.2 Foundation for intelligent physical agents semantic language

FIPA Semantic Language (FIPA SL) is a formal language developed to define the content of the FIPA-ACL. FIPA SL can be used to express objects, propositions and actions (Pitkaranta 2004). Object expression is used to declare variables and make assertions (Alagar et al 2002). Action expressions describe some action that is either already performed, intended to be performed in the future or is currently being performed. Propositions are used to represent the behavioral aspects of agents like goals, intents, beliefs and uncertainty (Alagar et al 2002). For example, agents may have persistent goals stated in the form

\[
(PG <\text{agent}> <\text{expression}>)
\]
This states that an agent holds a persistent goal that expression becomes true but will not necessarily possess a plan to achieve this Dogac and Cingil (2003).

The FIPA SL specification has three subsets: FIPA-SL0, FIPA-SL1 and FIPA-SL2. FIPASL0 is the most simple and FIPA-SL2 the most complex related to available logical features for representing information Pitkaranta (2004).

4.2.4 Ontology

Even if agents speak the same language, they require some common understanding of the meaning of the message content. This is provided by specialized knowledge component called ontology that specifies the objects, concepts and relationships in a given domain Poslad et al (2000). In other words, agents have to use similar vocabularies for representing shared concepts in a domain, and agents are also able to extract information from various sources when these sources share the same underlying ontology.

According to Dogac and Cingil (2003), ontologies are usually built using schema definition or a knowledge presentation language. Dogac and Cingil (2003) also state that creating and expressing ontologies of any size is difficult and time-consuming work and many tools have been developed for analyzing and developing ontologies. One of these tools is called Ontolingua.

4.2.5 Message Transport

Now that we have defined how software agents communicate in agent-level we will take a look at the wider picture in order to understand how these agent messages actually move in the network. This includes linking
these agent interaction protocols to network data transfer mechanisms. Multiple layered agent messages are finally encoded into transport messages that, among with other information, include the information about message receiver and sender. These transport messages are sent through the network using various transport mechanisms. There are many transport services available that are able to transfer messages from one agent to another in the network. Higher-level protocols such as Common Object Request Broker Architecture (CORBA), DCOM and Java Remote Method Invocation (RMI) allow the use of remote procedure calls between distributed objects that hide the lower level complexity of the transfer mechanisms. Here we don’t dig further to the world of network and transport protocols, but following list summarizes some of the most common ones of these protocols:

- CORBA
- DCOM
- Remote Method Invocation using Java RMI
  - Web Services
  - HTTP
- Wireless Access Protocol

These are network-level protocols, and Transmission Control Protocol / Internet Protocol (TCP/IP) is a common example of a lower-level data-transfer protocol when transferring data through network.

4.2.6 The Actual Construction: Platforms and Tools

Now that we have provided the theory for agent interaction, communication and general agent architectures, we still haven’t answered the question how to actually construct software agents? Agent systems are difficult to build from scratch. Chira (2003) states that numerous languages
and platforms have been created by different research groups and companies to support the development of agent-based applications. However, traditional languages are still usually used to construct agent applications. Currently the most used programming language for developing agent applications is Java. This is mainly because of its rich library of functions related to concurrency, security, support for object-oriented programming techniques and multithreading.

Agent implementors don’t have to start building agents from scratch since numerous platforms and toolkits have been introduced for building agent applications, and they are widely available for third-party developers, application developers and end-users under various licensing arrangements Poslad et al (2000). Agent toolkits are defined as sets of components from which to build agent systems and sets of tools to help operate agent systems Poslad et al (2000). Most agent toolkits are based on Java programming languages, and many implementations exist both on FIPA and KQML agent communication architectures. Some of these toolkits are presented in Table 4.4.

**Table 4.4 Agent toolkits**

<table>
<thead>
<tr>
<th>Product Name (Company)</th>
<th>Mobile/Stationary</th>
<th>Language</th>
<th>Standards</th>
<th>Availability/Licensing</th>
<th>Example applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>JatLite (Stanford Uni.)</td>
<td>Stationary</td>
<td>Java</td>
<td>KQML</td>
<td>Open source</td>
<td>Design decision tracking, constraint mgt, enterprise control</td>
</tr>
<tr>
<td>ZEUS (BT UK)</td>
<td>Stationary</td>
<td>Java</td>
<td>FIPA KQML</td>
<td>Open source</td>
<td>Supply chain mgt, service provisioning, network resource mgt.</td>
</tr>
<tr>
<td>FIPA-OS (Nortel Networks)</td>
<td>Stationary</td>
<td>Java</td>
<td>FIPA</td>
<td>Open source</td>
<td>Personalized services, VPN, VHE, meeting scheduler, etc.</td>
</tr>
<tr>
<td>JADE (CSELT)</td>
<td>Stationary</td>
<td>Java</td>
<td>FIPA</td>
<td>Open source</td>
<td>Travel assistant, audio-visual entertainment, meeting scheduler, etc.</td>
</tr>
</tbody>
</table>
4.3 AGENT COMMUNICATION SECURITY

This chapter describes how agent communications security can be provided in MAS. An agent’s ability to communicate is fundamental and essential for it to function in a multiagent system. In this chapter we describe how secure communication between agents on different platforms can be addressed within a multiagent system. The basic cryptographic tools used to provide secure communications, as well as the notation used in this chapter have been described. The security services for agent communication and their motivation have been described as well as how specific security services, namely data origin authentication, entity authentication, non-repudiation, confidentiality, and anonymity can be provided.

4.3.1 Architectural Issues

In this section we will consider in general terms where security services can be implemented and how the architecture can support secure agent communication. There are two main options for the location of secure communication services. One option is for the agents themselves to directly implement security services. This approach is rather independent from the architecture and the functionality offered by the execution platform, allowing the developer to make choices as to how security services are implemented and used. The other option is to place security services within the execution platform as services to communicating agents. In order to make the system scalable and manageable we believe that security services should be implemented as part of the architecture on the execution platform. However, this should be done in such a way that the first option is not prevented (although preventing agents providing security services is probably not something that can easily be enforced by the platform).
The security architecture contains entities and components that will facilitate secure agent communication. Of the involved parties the Trust Service Provider (TSP) is of particular importance for establishing secure communications, but other entities may also be involved. One role for a TSP is to act as a Certification Authority (CA) issuing digital certificates.

In the agent execution environment architecture, the agent security services and agent communication services are the main elements involved in the provision of secure communication. A subscription module can also be used to facilitate secure communication. This is assumed to be a small tamper proof module (e.g. a smart card) provided by a service provider. Such a device can be used for protected (agent) execution and protected storage of various information (including private and secret keys).

4.3.2 Data Origin Authentication

In this section we will describe how, and to what degree, data origin authentication can be provided, first for stationary (i.e. non-mobile) agents and then for mobile agents. We will use the following scenario to illustrate the message exchanges are shown in Figure 4.2. Agent Q, residing on platform P1, needs to send a message to agent R, residing on platform P2. R requires assurance that the message originates from Q.

![Figure 4.2. Agent Q sending a message to agent R](image-url)
Whether authentication and integrity should be provided between agents (Q-R), between agent and platform (Q-P2), or between platforms (P1-P2) should not be dictated by the system. This will be an application decision which is likely to differ depending on the nature of the application and the capability of the platforms for which the application is intended.

4.3.2.1 Stationary agents

The simplest solution involves Q sending a signed message to R as follows:

\[ Q \rightarrow R : \text{mks}_Q(m) \]

Of course, this solution requires Q to have its own signature key pair, and R to have a trusted copy of Q’s public signature verification key. This latter property can, for example, be achieved by transferring a public key certificate for the necessary public key, signed by a mutually trusted CA. Alternatively, if agent Q and agent R have a pre-established relationship in the form of a shared secret (K1), a Message Authentication Code (MAC) can be used instead. The computation of a MAC is generally less computationally intensive than that of a digital signature. In a scenario where an agent is frequently communicating with the same agent (or agents), this can be a more efficient solution.

\[ Q \rightarrow R : \text{mkMAC}_{K1}(m) \]

If an agent is executing on a host that is trusted and the agent’s secrets have not been compromised, the above protocols are enough to ensure data origin. However, in the case where it is possible that the agent’s key has been compromised, the executing host can also add its digital signature. This
would prevent a party who has obtained the agent’s key from impersonating
the agent in an undetectable fashion. It can also serve to bind the executing
platform to certain actions.

\[ Q \rightarrow R : \text{mks}_Q(m)\text{ks}_P(\text{mks}_Q(m)) \]

Or alternatively, if shared secrets exist

\[ Q \rightarrow R : \text{mkMAC}_K(I(m)\text{kMAC}_K(\text{mkMAC}_K(I(m))) \]

where \( K_1 \) is a secret shared between \( Q \) and \( R \), and \( K_2 \) is a secret shared
between \( P_1 \) and \( R \).

Combinations of digital signatures and MACs are, of course, also
possible. The above protocols does not protect against message deletion,
duplication, or delay attacks. To protect against message duplication and
message deletion the message \( m \) should be constructed to also include the
intended recipient and a freshness value. Freshness values usually take the
form of a timestamp or a nonce. Clock based timestamps require synchronised
clocks which would probably be infeasible to provide securely for all the
platforms in our system.

That is, we should use a construction of the following form:

\[ m = m_1kR_{ID}knonce \]

where \( m_1 \) is the original message, \( R_{ID} \) is the identity of the recipient, and nonce
is a unique message number. The nonce must be supplied by the recipient;
alternatively sequence numbers can be used which must be maintained by
both sender and recipient. General mechanisms using MACs for entity
authentication are further described, and general mechanisms using digital signatures for entity authentication are further described. To protect against delay attacks, timestamps or a challenge-response protocol would be required.

4.3.2.2 Mobile agents

For mobile agents, the above protocols can be insufficient even if platforms are required to add their signatures. An agent platform, on which the agent has previously executed, can at a later time still produce a valid agent signature. Figure 4.3 shows the path of a multi-hop agent, moving from its original platform (P₀), the ‘agent home’, to platforms P₁, P₂, ... and finally to Pₙ before returning to, or sending a final message to, the ‘agent home’ P₀. The ‘agent home’ is the agent’s origin and is assumed to be fully trusted by the agent (and its owner).

![Figure 4.3 Movement of a mobile agent](image)

This could, for example, be a user’s mobile terminal. If the protocols described are used, any platform from P₁ to Pₙ would be able to produce valid signature simply by extracting the agent’s cryptographic key
and using it to sign a message we will now describe some partial solutions to this problem. They are not complete solutions since they introduce new, sometimes significant, constraints or communication overheads and will not suit every situation.

### 4.3.2.2.1 Avoiding multi-hop agents

A partial solution to this problem, but with likely significant increased complexity, is to only use single-hop agents or to make the agent visit a trusted services provider between every visited platform. By making the agent return to the ‘agent home’ between every move, as shown in Figure 4.4, the need to use the same secret or private key on more than one platform can be avoided by updating the agent’s key every time it leaves $P_0$.

![Figure 4.4 Deployment of multiple one-hop agents](image)

Figure 4.5 shows how trusted platforms can be inserted in the path and used to update the agent’s key. The trusted platforms would be run by a
TSP. Depending on the application an agent might visit more than one platform before it needs to move to a trusted platform in order to obtain a new signature key.

Figure 4.5 Using trusted platforms to update the agent’s key

4.3.2.2.2 Undetachable signatures

Another solution is to associate a signature key with certain constraints. Sander and Tschudin (1998) proposed one such scheme, which they refer to as an undetachable signature. This prevents signatures from being applied to arbitrary messages.

Restricting signatures through digital certificates: An alternative to undetachable signatures is to use public key certificates to regulate the validity of digital signatures. Public key certificates are used to link an identity with a public verification key by which a verifier can check the validity of a digital signature. Certificates usually include a validity period under which valid signatures can be produced. By extending the constraints included in the certificate to context-related values such as executing host,
maximum value of a purchase, and so on, certificates can be used to further restrict the use of signature keys and thereby decrease the risks involving improper use of the signature key. One advantage with this scheme over undetachable signatures is that it relies on already well-established cryptographic techniques.

Distributing trust among multiple platforms, by using threshold signatures we can deploy agents in such a way that a single compromised platform or agent will not compromise the security of the scheme. The scheme can be tailored in such a way that a number of agents can be compromised without compromising the scheme. The idea of a threshold scheme is to take a secret, and divide it into pieces called shares which are distributed among a group of entities. Then any subset of these entities of a given size can reconstruct this secret, but a smaller group can learn no information about the secret. An example of such a scheme is given in Shamir (1979).

Threshold cryptography was first proposed by Desmedt (1988). One important type of threshold cryptosystem is known as a threshold signature. In such a scheme, any set of k parties from a total of parties can sign a document, whereas any coalition of less than k parties cannot. Such schemes tend to rely on a combiner which is not necessarily trusted. Several schemes have been proposed based on both El Gamal and Revest Shamir Adleman (RSA) signatures. Recently Shoup (2000) proposed an RSA-based scheme which is as efficient as possible; the scheme uses only one level of secret sharing, each server sends a single part signature to a combiner, and must do work that is equivalent, up to a constant factor, to computing a single RSA signature. Although not perfect as a threshold signature scheme this scheme is ideal in a mobile agent setting, where the user would be responsible for generating the shares for his agents.
By combining undetachable signatures and threshold signatures, we can achieve undetachable threshold signatures. An alternative to using threshold signatures is to generate unique signature keys for a number of agents and accompany these with public key certificates. Encoded in the certificates is also a threshold value. In order for a verifier to produce a valid signature a number of ‘agent-signatures’ meeting the threshold value must be collected.

A different approach, which also avoids complete trust in one execution environment, is to use cooperating agents. While one agent might reside on a platform where the main processing is carried out, other agents residing and executing on other platforms can carry sensitive or important information which is only released to the first agent once certain conditions have been fulfilled. This approach assumes that the platforms carrying the different agents will not cooperate, an assumption which is reasonable for our envisioned system. Such protocols have been proposed by Roth (1998).

Hiding signature key from executing hosts: If the ability to produce a signature can be incorporated into the agent in such a way that a malicious platform cannot misuse this information, the problem of agent authentication would have been solved. Techniques have been invented to hide information or execution code even from an executing host. Environmental key generation, computing with encrypted functions, and obfuscated code are all examples of such techniques. In the case of environmental key generation the agent’s private signature key would be held encrypted and only revealed if certain conditions are met. In the case of obfuscated code, it is the private signature key that would be embedded in scrambled form.
4.3.3 Entity Authentication

In this section we will look at how entity authentication can be achieved. The same mechanisms used to support data origin authentication can be used for entity authentication. Entity authentication can be unilateral or mutual. As for message authentication, asymmetric or symmetric cryptographic mechanisms can be used. If no pre-established secret exist between the agents, public-key cryptography can be used in combination with a public key infrastructure to facilitate authentication. For agents with regular communication, mechanisms not requiring public-key cryptography might be preferable as they are likely to be less computationally intensive. we suppose that agent Q, residing on platform P1, needs to authenticate itself towards agent R, residing on platform P2 (see Figure 4.2). Below is an example of a mutual authentication protocol using digital signatures.

\[ Q \rightarrow R : N_Q \]
\[ R \rightarrow Q : N_R ks_R(N_RkN_QkQ_ID) \]
\[ Q \rightarrow R : s_Q(N_QkN_RkR_ID) \]

where \( N_Q \) and \( N_R \) are nonces, \( Q_ID \) and \( R_ID \) are the identities of \( Q \) and \( R \) respectively.

As for origin authentication an application may also wish to authenticate the originating platform. This can be achieved by also letting the platform add its signature as follows.

\[ Q \rightarrow R : N_Q \]
\[ R \rightarrow Q : N_R ks_R(N_RkN_QkQ_ID)ks_R2(N_Rks_R(N_RkN_QkQ_ID)) \]
\[ Q \rightarrow R : s_Q(N_QkN_RkR_ID)ks_R1(s_Q(N_QkN_RkR_ID)) \]
For mobile agents, as in the origin authentication case, multi-hop agents can be avoided in order to eliminate misuse of the agent’s keys. The process of restricting the scope of signatures through digital certificates can also be applied to the entity authentication case. However, of the mechanisms described undetachable signatures and distributing trust among multiple platforms do not appear to be applicable to entity authentication for mobile agents.

4.3.3.1 Architecture support for authentication

In this section we will consider how the security architecture can support and facilitate authentication.

As stated above, agents can include all the functionality they need, including that necessary to support entity authentication. It is also possible to build applications independent of the supporting security architecture.

However, by providing security services to agents, a more efficient and manageable solution is accomplished.

An agent can request the execution platform to verify authentication information and be returned a success or failure message and, if requested, some sort of receipt that can be stored in a log or sent to a trusted platform for safe storage in case of future disputes.

If the executing platform does not have the certificates, or other supporting information, required for validating the information it can contact a trusted service provider offering this service.
In a highly distributed environment, certificates and certificate revocation information will be generated, maintained, and stored in various places. It is also likely that, for performance reasons, such information will be cached in various places in the infrastructure. An agent should therefore also be able to express various degrees of assurance in authentication verification when requesting services from the platform.

A subscription module can be used as protected storage and facilitate distribution of root certificates and shared secrets. The subscription module can also be used to protect processing involving these secrets, thereby minimising the risk that they are misused by an adversary.

4.3.4 Non-Repudiation

As mentioned earlier, one can distinguish non-repudiation of many actions. Generic protocols for non-repudiation services have been specified in a series of ISO standards. In this section we will outline a simple non-repudiation of origin service applicable in an agent context. The simplest form of non-repudiation protocol is to use digital signatures as described. Some kind of agreement would need to exist in order to be able to settle disputes at a later point in time. One major problem with such a protocol is the absence of any indication of time in the message. If accurate clocks are available at the point of the message creation, the time can be included in the message. To provide accurate clocks is a non-trivial task, and a possible point of attack; one way of addressing this is through the use of a Trust Service Provider (TSP), as proposed immediately below.

A simple non-repudiation service can be achieved as an extension to the authentication protocols given. A TSP can offer a non-repudiation service simply by offering a timestamp and forward service;
(M1) \( Q \rightarrow T : R_{\text{ID}} \text{km} \text{ks}_Q(R_{\text{ID}} \text{km}) \)

(M2) \( T \rightarrow R : \text{mks}_Q(R_{\text{ID}} \text{km}) k\text{TIME} \text{ks}_T(\text{mks}_Q(R_{\text{ID}} \text{km}) k\text{TIME}) \)

where \text{TIME} is the time of the transaction. The recipient’s identity is included in the first message to indicate to whom \( T \) should forward the message. For the protocol to work, the two communicating entities like agents, platforms need to be able to communicate with a TSP offering the service. The protocol further requires both communicating parties to trust the same TSP for the service. The protocol can be extended to offer non-repudiation of delivery by requesting the recipient (\( R \)) to send a signed reply, either directly to the sender (\( Q \)) as shown in Figure 4.6 or through the trusted service provider in the same manner as the protocol just described. More sophisticated protocols have been proposed that can be used also in an agent context to achieve non-repudiation. Examples include protocols proposed for certified e-mail Ateniese et al (2001), Micali (1997). An attractive property of the protocol proposed in Micali (1997) is that a trusted third party only needs to be involved when the protocol is failing.

![Platform P1](Platform P1)

Agent

Q

![Platform P2](Platform P2)

Agent

R

Figure 4.6 TSP offering a timestamp and forward service

4.3.4.1 Architecture support for non-repudiation

The protocol described above requires the support of a trusted service provider to timestamp and forward messages. This can be
implemented as part of the middleware. Users might get access to the service and ‘agree to trust’ the Trust Service Provider (TSP) simply by having a contract with their home service provider (other business models are also possible). A supporting micropayment scheme could be used to charge for the service. If more sophisticated protocols are required for certain applications, e.g. for high value transactions, this might be better implemented as an application service above a middleware layer.

4.3.5 Confidentiality of Communication

In this section we will describe how confidentiality of communication can be achieved. For communication between agents executing on separate hosts, encryption is required for confidentiality protection.

Q can simply, before sending the message to R as shown in Figure 4.7, encrypt the message using R’s public key as follows:

\[ Q \rightarrow R : E_R(m) \]

![Figure 4.7 Agent Q sending a message to agent R](image)

Only R will then be able to decrypt the message using the corresponding private key.

If agent Q and agent R have a pre-established relationship in the form of a shared secret, symmetric cryptography can be used instead.
Symmetric encryption/decryption is in general less computationally intensive than corresponding asymmetric operations. Again, in a scenario where an agent is frequently communicating with the same agent this can be a more efficient solution.

That is, we have:

$$Q \rightarrow R : E_KI(m)$$

where $K1$ is a shared secret between $Q$ and $R$. Since asymmetric cryptography has advantages when it comes to key distribution, and symmetric key cryptography has efficiency advantages, it is common to use a combination of the two, i.e.

$$Q \rightarrow R : E_R(K)kE_K(m)$$

where $E_R(K)$ is a secret key $K$ encrypted using $R$’s public key. The secret key $K$ is then used to encrypt the message.

The same mechanisms can be used to initialise a protected session. Alternatively, and perhaps more likely to be implemented in a real system, the Transport Layer Security (TLS) protocol (Dierks and Allen 1999) can be used, where TLS is a standardised protocol for initializing and encrypting a session (Claessens et al 2001). TLS also supports unilateral or mutual authentication of the communicating parties.

To achieve traffic confidentiality at the communication layer where agent communication takes place, traffic padding can be used. Agents would then generate dummy traffic in order to hide the real communication that is taking place. Depending on the threat scenario, it might be enough to
generate dummy traffic between two communicating agents or agents can send dummy traffic to multiple agents in order to conceal the real recipient.

4.3.5.1 Architecture support for confidentiality

In this section we will consider how the security architecture can support and facilitate confidentiality of communication. Like the functionality supporting integrity, the security architecture can help in finding, retrieving, and verifying public key certificates for agents. Further, the executing platform can provide local agents with encryption and decryption of messages as well as entire sessions. As for authentication support, the subscription module can be used for storage and distribution of root public keys.

4.3.6 Anonymity

In this section we describe how anonymity can be offered within the multi-agent system.

The possibility of anonymous transactions is potentially an important factor for user acceptance. Agents are in many ways ideal or providing anonymity to their owners as they are independent entities, possessing some degree of autonomy, and do not require direct user interaction.

The agent based system can, for example, provide for this by (1) allowing communication with, and execution of, agents not carrying the identity of the agent owner, and (2) by the use of proxy services that would allow a user to be traced only if the user or her agents are misbehaving. This service can be provided in the network by a service provider. It would be the security policy of a device that regulates whether or not such agents and their transactions would be allowed onto the device.
We now use the KQML protocol, to outline how anonymity can be provided to agent communication through a proxy service. A KQML message envelope carries, amongst other information, the parameters to and from, indicating the identity of the intended recipient and the identity of the originating agent in the format: agent@host.com. The form parameter is mandatory but can be changed to one that does not reveal the agent’s true identity or the host on which the agent is running. This would provide anonymity to the agent as well as to its owner. However, if the parameter is used for replies, then the use of an arbitrary agent identity will prevent replies being delivered. One should also be aware that an underlying transport services might reveal the physical address of such a message.

A proxy service would remove these problems. Suppose that agent Q, residing on platform P1, needs to send an anonymous message to agent R, residing on platform P2, to which R should be able to reply. A proxy service (PS) can then be used as follows, and as shown in Figure 4.8.

![Diagram](image)

**Figure 4.8 Proxy service giving anonymity**

(M1) $Q \rightarrow PS : f_{rom} : Q@P1kto : PSkto : R@P2$

(M2) $PS \rightarrow R : f_{rom} : A@PSkto : R@P2$

(M3) $R \rightarrow PS : f_{rom} : R@P2kto : A@PS$

(M4) $PS \rightarrow Q : f_{rom} : PSkto : Q@P1kxf \ f_{rom} : R@P2$
The protocol, which works in a similar way to established Internet mail remailers, does require more messages than the standard non-anonymous version, and requires access to the proxy service. The agent’s identity is only kept secret from P2 and R, not from the proxy service. Hence, depending on the application, the proxy service might also be required to be trusted (to various degrees).

By using mixes, as first proposed by Chaum (1981) or one of its variants, Gulcu and Tsudik(1996), Jakobsson(1999), Syverson et al(1997) the level of anonymity can be further enhanced. A network of mixes can, in combination with cryptography, be used to split, and route, a message through a network to its recipient, in such a way that any single mix-node, an interceptor, or the recipient cannot identify the initiator. The recipient can still use the same path through the mix-network to route a replay to the initiator; this is achieved by assigning unique identifiers to individual message transfers between mix-nodes, that can be used to route a reply back to the initiator.

4.3.6.1 Architecture support for anonymity

The proxy anonymity service can be provided as part of a middleware layer. It can be operated by a TSP who has a relationship with the agent owner and, if required, can provide certain assurance regarding the behaviour and actions of the agent. If the agent is mobile the agent’s identity would be exchanged for a proxy identity. For a communicating agent the proxy service would forward messages to their real owners.

If necessary, and if such a function is supported by the TSP, the proxy solution can also be used to trace the real owner.
4.4 THESIS CONTRIBUTION IN AGENT BASED SECURITY

The major contribution of this thesis is the introduction of the security for multiagent system. Because in our architecture agents from different platforms communicate and exchange their information through KQML. Hence security plays a vital role in multi agent systems to make the agent communication a secure one.

In this chapter we have shown how security protocols can be used to provide secure communication within an agent-based system. The security protocols proposed in this thesis work provide services like, data integrity, data origin authentication, entity authentication, non-repudiation, confidentiality, and anonymity which are required for the proper functionality of the system.