Multi-agent system security for mobile communication

Niklas Borselius

Technical Report
RHUL–MA–2003–5
1 September 2003

Department of Mathematics
Royal Holloway, University of London
Egham, Surrey TW20 0EX, England
http://www.rhul.ac.uk/mathematics/techreports

Multi-agent system security for mobile communication

by

Niklas Borselius

Thesis submitted to the University of London
for the degree of Doctor of Philosophy

Department of Mathematics
Royal Holloway, University of London

2003

Abstract

This thesis investigates security in multi-agent systems for mobile communication. Mobile as well as non-mobile agent technology is addressed.

A general *security analysis* based on properties of agents and multi-agent systems is presented along with an overview of security measures applicable to multi-agent systems, and in particular to mobile agent systems.

A *security architecture*, designed for deployment of agent technology in a mobile communication environment, is presented. The security architecture allows modelling of interactions at all levels within a mobile communication system. This architecture is used as the basis for describing security services and mechanisms for a multi-agent system. It is shown how security mechanisms can be used in an agent system, with emphasis on secure agent communication.

Mobile agents are vulnerable to attacks from the hosts on which they are executing. Two methods for dealing with threats posed by malicious hosts to a trading agent are presented. The first approach uses a threshold scheme and multiple mobile agents to minimise the effect of malicious hosts. The second introduces trusted nodes into the infrastructure.

Undetachable signatures have been proposed as a way to limit the damage a malicious host can do by misusing a signature key carried by a mobile agent. This thesis proposes an alternative scheme based on conventional signatures and public key certificates.

Threshold signatures can be used in a mobile agent scenario to spread the risk between several agents and thereby overcome the threats posed by individual malicious hosts. An alternative to threshold signatures, based on conventional signatures, achieving comparable security guarantees with potential practical advantages compared to a threshold scheme is proposed in this thesis.

Undetachable signatures and threshold signatures are both concepts applicable to mobile agents. This thesis proposes a technique combining the two schemes to achieve *undetachable threshold signatures*.

This thesis defines the concept of *certificate translation*, which allows an agent to have one certificate translated into another format if so required, and thereby save storage space as well as being able to cope with a certificate format not foreseen at the time the agent was created.

Acknowledgements

First, I would like to thank my supervisor, Professor Chris Mitchell, for his help, constant support, and for always being available and encouraging.

Next, my thanks go to Mobile VCE for funding and motivating my research, and also for bringing together industry and academia into a fruitful and stimulating environment. Thanks to everyone, academics and industrials, with whom I have had the pleasure to work through Mobile VCE.

The staff and my fellow students within the Mathematics department, by their warmth and friendship, have provided a most pleasant and genial environment during my years of study at Royal Holloway – my thanks go to all of them.

Finally, I want to thank my parents who have supported me in whatever decisions I have taken throughout my life, and whose support has also been important during my studies in England.

Contents

List of Figures

List of Tables

Acronyms

ACC	Agent Communication Channel
ACL	Agent Communication Language or Access Control List (depending on context)
AI	Artificial Intelligence
API	Application Program Interface
ARPA	Advanced Research Projects Agency
BER	Basic Encoding Rules
CA	Certification Authority
CPU	Central Processing Unit
CTS	Certificate Translation Server
DTD	Document Type Definition
EC	Environment Certifier
EMV	Europay, Mastercard and Visa
FIPA	Foundation for Intelligent Physical Agents
GSM	Global System for Mobile communication
HSP	Home Service Provider
IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	International Organization for Standardisation
KQML	Knowledge Query and Manipulation Language

MAC	Message Authentication Code
MAS	Multi-Agent System
MExE	Mobile Station Application Execution Environment
MS	Mobile Station
PC	Personal Computer
PDA	Personal Digital Assistant
PER	Packed Encoding Rules
PGP	Pretty Good Privacy
PKI	Public Key Infrastructure
PKIX	Public-Key Infrastructure (X.509)
PRAC	Partial Result Authentication Code
PS	Proxy Service
RA	Registration Authority
RFC	Request For Comments
RSA	Rivest, Shamir, and Adleman – a public key cryptosystem named after its inventors
SCVP	Simple Certificate Validation Protocol
SIM	Subscriber Identity Module
SP	Service Provider
SPKI	Simple Public Key Infrastructure
TLS	Transport Layer Security
TSP	Trust Service Provider
UMTS	Universal Mobile Telecommunications Service
USB	Universal Serial Bus
VPN	Virtual Private Network
WAP	Wireless Application Protocol
WTLS	Wireless Transport Layer Security
XML	Extensible Markup Language

Chapter 1

Introduction

Contents

This chapter describes the context of this research, its contribution to the field of security in multi-agent systems for mobile communication, and presents the structure of this thesis. Definitions and notation used throughout the thesis are also defined.

1.1 Motivation and challenges

The concept of an agent originates from the area of Artificial Intelligence (AI) but has now gained more widespread acceptance in mainstream computer science [84]. The term 'agent' has become fashionable, and a more mature technology than currently available is often implied. This is in particular true for security in multi-agent systems. Over-simplified assumptions and references to security solutions that do not address all the issues are not uncommon in the literature. Naturally, security is not a driving force for the research and development of multi-agent systems, and therefore has not received much attention from the agent community. Nevertheless, in order for agent technology to gain widespread use and provide viable solutions for commercial applications, security issues need to be addressed.

Autonomous agents and multi-agent systems represent a relatively new way of analysing, designing, and implementing complex software systems. This thesis is only concerned with the security of the system and its components (leaving design methodologies to others). Several multi-agent systems are available as commercial products and many more have been implemented in various research projects, with varying success. Ongoing standardisation efforts [21, 48] have proven successful and are still evolving. Today there is growing interest and research in implementing and rolling out (open) multi-agent systems on a wider scale[1]. Mobile VCE (www.mobilevce.com), which funded this research, is undertaking one such project where the agent paradigm is researched in a mobile telecommunications setting.

[1]See http://www.agentcities.org for an example of an ongoing effort to implement large scale multi-agent systems.

Agents are independent pieces of software capable of acting autonomously in response to input from their environment. Agents can posses differing capabilities [80], but typically possess the required functionality to fulfil their design objectives. To be described as 'intelligent', software agents should also have the ability to act autonomously, that is without direct human interaction, be flexible, and in a multi-agent system be able to communicate with other agents, that is to be social. Agents are, to various degrees, aware of their environment, which can also often be affected by the agents' actions.

A mobile agent is a particular class of agent with the ability during execution to migrate from one host to another where it can resume its execution. It has been suggested that mobile agent technology, amongst other things, can help to reduce network traffic and to overcome network latencies [52]. The limited processing resources and power supply available to many mobile devices are also arguments for mobile agents. Mobile agents could migrate to hosts with sufficient resources. An agent's ability to move does, however, introduce significant security problems.

Future mobile communication systems are envisioned as being able to offer a much wider range of services [108], thereby putting new requirements on the communication infrastructure. The software agent paradigm is believed to be able to offer a number of important properties for applications as well as for middleware services of future mobile communication systems [83]. It is within this context that the research presented in this thesis has been pursued.

The system envisioned utilises agent technology. Agents can exist on all kinds of hosts throughout the infrastructure, from the smallest devices (e.g. watch or phone) to application servers and communication infrastructure equipment.

While some agents will have the capability to move between platforms, others will always reside on the same platform. Whichever is the case, agents need access to certain security functionality. The envisioned system also includes, and allows, devices not using agent technology.

The system will also need to cope with different kinds of information each of which may require very different levels of protection. While certain information needs extensive protection, other information might not. This must be kept in mind when providing security services.

The system needs to be scalable. Security functionality can often be easily rolled out and provided for a small system; however, to do so on larger scale and in such a way that the system is allowed to grow requires robust solutions.

Openness in this context means that there is no single authority in control of the system. This means that users and service providers are able to connect to the infrastructure and utilise it in a similar way to the Internet today. Indeed, the system is likely to utilise, and interact with, the Internet.

Flexibility, scalability, and openness are important properties for the envisioned agent system [83]. One of the motivations for the use of the agent paradigm is that it facilitates flexibility: self-contained agents can be deployed to work autonomously in the system, with minimal management from the system itself. Security services for a multi-agent system should be general enough to be used as desired by the agents without imposing too high an overhead. Furthermore, security functionality should not be implemented in such a way that it prohibits the provision of additional security functionality, as may be required within certain applications. Therefore security functionality needs to be provided in a manner that does not impede flexibility or extension.

1.2 Structure of thesis

This thesis is organised as follows. In the remainder of this chapter, the main contributions of this thesis are first described, the notation and cryptographic primitives used throughout the thesis are specified, and definitions of security terminology used are given.

Chapter 2 describes properties of agents and multi-agent systems, focusing on issues with possible security implications. Security issues for multi-agent systems are then identified.

Chapter 3 describes available technologies and research efforts addressing security issues in multi-agent systems, with emphasis on technology addressing issues for mobile agents.

In chapter 4 a security architecture for agent based mobile systems is proposed. The architecture has four levels of abstraction: the involved parties, the device structure, the agent execution environment, and the agent construction.

Chapter 5 describes security services and mechanisms for agent based mobile systems, and relates these to the architecture proposed in chapter 4.

Chapter 6 describes how security of communication between agents on different platforms can be addressed in general, as well as within the context of the security model presented in chapter 4.

Chapter 7 evaluates the FIPA standards for agent communication protocols and outlines how security can be added to them.

In chapter 8 a ‘pragmatic’ alternative to undetachable signatures is proposed, relying on the use of conventional signatures and public key certificates. Undetachable signatures let a user limit the intention of the data string to which a correct signature can be applied. We show how this can be achieved using conventional mechanisms.

Chapter 9 proposes two methods to improve the security and reliability of mobile agent based transactions in an environment which may contain some malicious hosts.

Chapter 10 presents alternatives to threshold signatures that raise questions about the value of such schemes when applied in a mobile agent setting.

Chapter 11 introduces the concept of undetachable threshold signatures, which enables constrained signing power to be distributed across multiple agents, thus reducing the necessary trust in single agent platforms.

In chapter 12 the concept of certificate translation is defined and examples of its applications are proposed.

Finally, chapter 13 gives the conclusions of this thesis.

1.3 Contribution of thesis

A security analysis based on properties of agents and multi-agent systems is presented (chapter 2). Security analyses have been carried out by others, but usually with a particular application in mind, resulting in more narrow threat scenarios compared to the security issues presented in this thesis. An overview of the security measures applicable to multi-agent systems, and in particular to mobile agent systems, is also given (chapter 3).

A security architecture, designed for deployment of agent technology in a mobile communication environment, is presented in chapter 4. The security architecture allows modelling of interactions at all levels within a mobile communication system. This architecture is then used as a basis for describing security services and mechanisms for a multi-agent system (chapter 5), with emphasis on secure agent communication (chapter 6), where we show how security mechanisms can be used in an agent system.

The FIPA agent communication protocols have become a de facto standard, but lack security functionality. This is addressed in chapter 7 where we analyse the FIPA specifications and outline how security can be added to them.

Undetachable signatures have been proposed [98] as a way to limit the damage a malicious host can do by misusing a signature key carried by a mobile agent. In chapter 8 we propose a scheme for the same purpose, relying on conventional signatures and public key certificates.

Mobile agents are vulnerable to attacks from the hosts on which they are executing. If a mobile agent is sent out to find a particular item of merchandise on a user's behalf it is exposed to a range of possible attacks from the hosts visited by the agent. In chapter 9 we propose two ways to overcome the threats posed by malicious hosts. The first approach uses a threshold scheme and multiple mobile agents to minimise the effect of malicious hosts. The second approach introduces trusted nodes into the infrastructure.

Threshold signatures can be used in a mobile agent scenario to spread the risk between several agents and thereby overcome the threats posed by individual malicious hosts. In chapter 10 we propose a rather simple alternative to threshold signatures based on conventional signatures, achieving comparable security guarantees with potential practical advantages compared to a threshold scheme.

Undetachable signatures and threshold signatures are both concepts applicable to mobile agents. In chapter 11 we propose a scheme combining the two schemes to achieve *undetachable threshold signatures*.

Agents in a mobile communications environment are likely to be exposed to a variety of applications requiring them to produce digital signatures, and supply a digital certificate that can be used to validate the signature. Several different formats for digital certificates exist today, usually optimised for a particular application or environment. If an agent is exposed to several of these applications or environments, it may need to have as many digital certificates. In chapter 12 we define the concept of *certificate translation*, which allows an agent to have one certificate translated into another format if so required, and thereby save storage space as well as being able to cope with a certificate format not foreseen at the time the agent was created.

1.4 Notation and cryptographic primitives

In this section we briefly describe some of the cryptographic tools used to provide security functionality, as well as the notation used throughout this thesis. For a more thorough introduction to all the necessary cryptography see, for example, [87].

Symmetric encryption

Symmetric encryption, or secret-key encryption, uses a secret key to encrypt a message into ciphertext and the same key to decrypt the ciphertext into the original message.

For the purposes of this thesis

$$E_K(m)$$

denotes symmetric encryption of data string m using secret key K.

There are a number of available symmetric encryption algorithms (see, for example, [87]).

Symmetric cryptography, which includes symmetric encryption and messages authentication codes (see below), requires the sender and receiver to agree on a shared secret key. Symmetric cryptography is in general more efficient in terms of computing resources than asymmetric cryptography (defined below).

Message authentication codes

Message Authentication Codes (MACs) aim to guarantee the source and integrity of a message. A MAC is sent together with the message it is protecting.

For the purposes of this thesis

$$MAC_K(X)$$

denotes a MAC computed on data X using the secret key K.

There are a variety of well-established means for computing MACs, typically either based on the use of a block cipher or a cryptographic hash function (see, for example, [87]). There are also standards for such schemes, notably ISO/IEC 9797 parts 1 and 2 [72, 73].

Asymmetric cryptography

Asymmetric cryptography, or public-key cryptography, involves the use of two distinct keys, one public and one private. The private key is kept secret by its owner, while the public key can be freely shared with everyone. There are a number of different types of asymmetric cryptographic schemes, including encryption schemes and digital signatures.

Whilst asymmetric cryptography does not, like symmetric cryptography, rely on the sender and receiver agreeing on a shared secret, the user of the public key must ensure that the correct key is used. Public Key Infrastructures (PKIs) are used for this purpose. In a PKI, Certification Authorities (CAs) issue digitally signed certificates which bind a public key to an identity and possibly other information (e.g. certificate expiry date). X.509 [76] is a widely adopted standard specifying the format of digital certificates. Standards also exists for PKIs, see for example IETF PKIX[2].

Asymmetric encryption

In an asymmetric encryption scheme, the public key is used for encryption and the private key for decryption. The most commonly used algorithm for public-key encryption is RSA (see for example [87]). Standards describing how to use asymmetric encryption include [59].

For the purposes of this thesis

$$E_X(m)$$

denotes the asymmetric encryption of data string m using the public key of entity X.

Digital signatures

Digital signatures aim at guaranteeing the origin and integrity of a message. The originator of a message uses his signing key, the private key, to sign the message and sends it along with the message to the recipient. The recipient uses the verification key, i.e. the public key of the signer, to verify the origin and integrity of the message. Typically a digital signature functions as a check value on data, and we assume the use of such a signature scheme throughout this thesis. That is, when sending a digital signature on data, both the data and the signature need to be transmitted. Signature schemes do exist where

[2] http://www.ietf.org/html.charters/pkix-charter.html

part or all of the data can be recovered from the signature itself, but these are less commonly used.

For the purposes of this thesis

$$s_X(m)$$

denotes the signature of entity X (which must be computed using the private signature key of X) on data string m.

There are many signature schemes available (see for example [87]), including a number of techniques which are international standards, see for example, [45, 59, 66, 67, 68].

1.5 Definitions

In this section security terminology used in the reminder of this thesis is defined. This is by no means intended to be a complete list of security terminology. (For further information and discussions of security terminology see, for example, [47, 49, 69].)

Access control: The means of enforcing authorisation [47].

(Entity) Authentication: The provision of assurance of the claimed identity of an entity [47].

Authorisation: The granting of rights, by the owner or controller of a resource, for others to access that resource [47].

Certification Authority (CA): A centre trusted to create and assign public key certificates. Optionally, the certification authority may create and assign keys to the entities [61].

Confidentiality: The property that information is not made available or disclosed to unauthorised individuals, entities, or processes [69].

Data integrity: The property that data has not been altered or destroyed in an unauthorised manner [69]. For the purposes of this thesis, when not specifically stated otherwise, we use *integrity* to mean *data integrity*.

Data origin authentication: The corroboration that the source of data received is as claimed [69].

Digital certificate: A digitally signed data structure containing an identifier for an entity and certain information associated with that entity, e.g. a public key or an access control attribute [40].

Non-repudiation of origin: Protects against the originator's false denial of having created the content of a message and of having sent a message [62].

Non-repudiation of receipt: Protects against a recipient's false denial of having received a message [62].

Public key certificate: A digital certificate containing a public key for an entity.

Public Key Infrastructure (PKI): System consisting of TTPs, together with the services they make available to provide certified public keys.

Traffic flow confidentiality: A confidentiality service to protect against the inference of information from observation of traffic flows [69].

Trusted Third Party (TTP): A security authority, or its representative, trusted by other entities with respect to security related activities [67].

Security policy: A set of rules that apply to all security-relevant activities in a domain [47].

Chapter 2

Security issues in multi-agent systems

Contents

This chapter describes properties of agents and multi-agent systems, focusing on properties with possible security implications. Security issues for multi-agent systems are then identified.

2.1 Introduction

This chapter briefly describes properties of agents and multi-agent systems (section 2.2). It is not intended to be a complete description of agents multi-agent systems (we are, for example, not concerned with AI properties for agents here). The focus is restricted to issues with possible security implications. Using these properties, the security issues for multi-agent systems are identified (section 2.3). Finally, the conclusions of this chapter are presented in section 2.4. Some of the work described in this chapter has been previously published in [9, 10]. For further information on agents and multi-agent systems (MAS) see, for example, [54, 80, 117].

The security analysis presented in this chapter is based on properties of agents and open multi-agent systems. Security analyses have been presented elsewhere, often with a particular application in mind, resulting in more narrow threat scenarios compared to the general security issues presented in this chapter. For a thorough analysis of mobile agent security see [78]. The malicious host problem is described in some detail in [53, 56]. Other analyses of security issues for mobile agents can be found in [26, 50, 98]. For general multi-agent systems, not focusing on *mobile* agents, security analyses are more scarce. The following publications all contain limited security analyses of multi-agent systems, mostly in the context of a particular application or system implementation, [46, 55, 104, 106, 116].

2.2 Agents and multi-agent systems

Agents are software entities that exhibit autonomy and certain ‘intelligence’. An agent is often assumed to represent another entity, such as a human. No single universal definition of agents exists, but there are certain widely agreed universal characteristics of agents; these include situatedness, autonomy, and flexibility [80].

- **Situatedness** means that the agent receives sensory input from its environment and can perform actions which change the environment in some way.
- **Autonomy** means that an agent is able to act without the direct intervention of humans (or other agents), and that it has control over its own actions and internal state.
- **Flexibility** can be defined to include the following properties:
  - **Responsive**: refers to an agent’s ability to perceive its environment and respond in a timely fashion to changes that occur in it.
  - **Pro-active**: agents are able to exhibit opportunistic, goal-driven behaviour and take the initiative where appropriate.
  - **Social**: agents should be able to interact, when appropriate, with other agents and humans in order to solve their own problems and to help others with their activities.

A number of other attributes are sometimes discussed in the context of agent systems. These include but are not limited to [80]:

- **Rationality:** the assumption that an agent will not act in a manner that prevents it from achieving its goals and will always attempt to fulfil those goals.
- **Veracity:** an agent will not knowingly communicate false information.
- **Benevolence:** an agent cannot have conflicting goals that either force it to transmit false information or to effect actions that cause its goals to be unfulfilled or impeded.
- **Mobility:** the ability for an agent to move across networks and between different hosts to fulfil its goals. For the purpose of this thesis we limit the definition of mobile agents to those agents with the ability to migrate, on their own initiative, during execution from one host to another where they can resume their execution.

A multi-agent system (MAS) is a system composed of multiple autonomous agents with the following characteristics [79]:

- each agent cannot solve a problem unaided,
- there is no global system control,
- data is decentralised, and
- computation is asynchronous.

Computer hosts, or platforms, provide agents with environments in which they can execute. A platform typically also provides additional services, such as communication facilities, to the agents it is hosting. In order for agents to be able to form a useful open multi-agent system where they can communicate and cooperate, certain functionality needs to be provided to the agents. This includes functionality to find other agents or to find particular services. This additional functionality can either be implemented as services offered by other agents or as services more integrated into the MAS infrastructure itself. Examples of such services include facilitators [19], matchmakers [20], mediators [114], and blackboards [90].

Open multi-agent systems are usually envisioned as systems, communicating over the Internet, allowing anybody to connect to a platform on which agents are running. This means that the MAS lacks a global system control and that information in general is highly decentralised.

2.3 Security implications

In this section we will discuss agent security issues based on the characteristics described in the section 2.2. Security issues in agent systems have been analysed previously, usually with a focus on a particular application or domain, see for example [7, 94]. In this section we present a general analysis based on the fundamentals of agents and MAS.

2.3.1 Agent Execution

Naturally, agents need to execute somewhere. A computer host, the immediate environment of an agent, is ultimately responsible for the correct execution and protection of the agent. This leads to the question of where access control decisions should be performed and enforced. Should an agent contain all necessary logic and information required to decide if an incoming request is authentic (originating from its claimant) and if so, is it authorised (has the right to access the requested information or service)? Or can agents rely on their execution platform to provide access control services?

The environment might also need certain protection from the agents that it hosts. An agent should, for example, be prevented from launching a denial of service attack through consuming all resources on a host, thus preventing the host from carrying out other things (such as executing other agents). Security issues related to the executing host become even more apparent for agents that are mobile, further described in section 2.3.5.

2.3.2 Situatedness

The meaning of the term 'environment' depends on the application and appears to vary somewhat arbitrarily in the agent literature; it can for example be the Internet or the host on which the agent is executing. An agent is assumed to be 'aware' of certain states or events in its environment. Depending on the nature and origin of this information, its authenticity and availability need to be considered; (confidentiality of such information might also be relevant). If an agent's 'environment' is limited to the host on which it is executing, no specific security measures might be necessary (assuming the host environment cannot be spoofed). The situation is, however, likely to be different if the agent receives environment information from, or via, the Internet. (Security of communication is further explored in section 2.3.4.)

2.3.3 Autonomy

Autonomy, when combined with other features given to agents, can introduce serious security concerns. If an agent, for example, is given authority to buy or sell things, it should not be possible for another party to force the agent into committing to something it would not normally commit to. Neither should an agent be able to make commitments it cannot fulfil. Hence, issues related to delegation needs to be considered for agents.

The autonomy property does not necessarily introduce any 'new' security concerns; this property is held by many existing systems. It is worth mentioning that Internet worms (often referred to as viruses) also hold this property, which enables them to spread efficiently without requiring any (intentional or unintentional) human interaction. The lesson to learn from this is that powerful features can also be used for malicious purposes if not properly controlled.

2.3.4 Communication

Of the flexibility properties, social behaviour is certainly interesting from a security point of view. This means that agents can communicate with other agents and humans. Just as an agent's communication with its environment needs to be protected, so does its communication with other agents and humans. The following security properties should be provided:

- Confidentiality: assurance that communicated information is not accessible to unauthorised parties.
- Data integrity: assurance that communicated information cannot be ma-

nipulated by unauthorised parties without being detected.

- Authentication of origin: assurance that communication originates from its claimant.
- Availability: assurance that communication reaches its intended recipient in a timely fashion.

In addition to these basic security properties, non-repudiation services should be considered. One can distinguish non-repudiation of many actions (eight non-repudiation services are defined in [62]). We believe the following non-repudiation services to be the ones most useful in a generic agent system:

- Non-repudiation of origin: protects against the originator's false denial of having created the content of a message and of having sent a message [62].
- Non-repudiation of receipt: protects against the recipient's false denial of having received a message [62].

Fundamental to the above-mentioned communication security properties are issues relating to the identification and authentication of the sending and receiving parties. These issues are discussed further in section 2.3.7.

It should be noted that security usually comes at a cost. Additional computing resources as well as communication resources are required by most solutions to achieve the above-mentioned security requirements. Therefore, security needs to be dynamic. Sometimes it makes sense to protect all communication within a system to the same degree, as the actual negotiation of security mechanisms can then be avoided. However, in a large scale open multi-agent system, security services and mechanisms need to be able to fit the purpose and nature of the communications of various applications with differing security requirements.

Some implementations of MAS assume that security is provided transparently by a lower layer. This approach might be sufficient in a closed system where the agents can trust each other and the only concern is external malicious parties. However, we believe that agents in an open system may often need to be 'security aware', i.e. they need to be able to make decisions based on where information is originating from and how well protected it is. As suggested elsewhere (e.g. [55]), public key cryptography and a supporting Public Key Infrastructure (PKI) can be used as important tools for securing inter-agent communication.

With a public key infrastructure in place, security protocols and mechanisms already developed for other applications can be made to fit the requirements of multi-agent systems to provide authentication, confidentiality, and data integrity.

2.3.5 Mobility

The use of mobile agents raises a number of security concerns. Agents need protection from other agents and from the hosts on which they execute. Similarly, hosts need to be protected from agents and from other parties that can communicate with the platform. The problems associated with the protection of hosts from malicious code are quite well understood.

The problem posed by malicious hosts to agents seems more complex to solve. Since an agent is under the control of the executing host, the host can in principle do anything to the agent and its code. The particular attacks that a malicious host can make have been described in [53] and [56], and can be summarised as follows.

- Observation of code, data and flow control.
- Manipulation of code, data and flow control – including manipulating the route of an agent.
- Incorrect execution of code – including re-execution.
- Denial of execution – either in part or whole.
- Masquerading as a different host.
- Eavesdropping on agent communications.
- Manipulation of agent communications.
- False system call return values.

2.3.6 Rationality, veracity, and benevolence

These properties could at a first glance appear to be very security relevant. However, on closer consideration they seem to be too abstract for us to consider as practical security concerns. The meaning (from a security point of view) of these properties seems to be: "Agents are well behaved and will never act in a malicious manner." If we make this a genuine requirement, then the required redundancy for such a system is likely to make the system useless. It would, of course, be valuable to have a system where agents can be assumed to behave truthfully and honestly in every situation. This does not seem a likely scenario for a multi-agent system that is not under very strict control and under a single authority, and would not correspond to the assumed open system scenario.

However, measures can be taken to limit maliciously behaving agents. Assurance that only information from trusted sources is acted upon and that agents (or their owner) can be held responsible for their actions, as well as monitoring and logging of agent behaviour, are all mechanisms that can help in creating a system where the actions of malicious agents can be minimised.

2.3.7 Identification and authentication

Identification is not primarily a security issue in itself; however, the means by which an agent is identified are likely to affect the way an agent can be authenticated. For example, an agent could simply be identified by something like a serial number, or its identity could be associated with its origin, owner, capabilities, or privileges. As mentioned in section 2.3.4, authentication is often fundamental to secure communication. If identities are not permanent, security-related decisions cannot be made on the basis of an agent's identity.

Connected with identification and authentication is anonymity. While an entity's identity is of major importance to certain applications and services, it is not needed in others. An open multi-agent system would probably require some sort of anonymity service to acquire wide acceptance today. In fact, agents are likely to be ideal for providing anonymity to their owners as they are independent pieces of code, possess some degree of autonomy, and do not require direct user interaction.

2.3.8 Anonymity

As mentioned in section 2.3.7, agents appear to be ideal for providing user anonymity. However, anonymity can pose a big threat in an open system. Agents are independent pieces of code, they possess some degree of autonomy, and they do not require direct user interaction; without the ability to trace agent activities to users, agents can be used to launch denial of service attacks.

2.3.9 Trust

Agents need to be able to make decisions based on information received and collected from other entities. In order to make these decisions they need to be able to evaluate the trustworthiness of the information or the information source. A mobile agent needs to decide whether or not to transfer to, and execute on, a particular host.

The issues surrounding trust within agent systems are currently attracting much research within the agent community. Various mechanisms for agents to reason about trust have been proposed, see, for example, [23]. Trust mechanisms based on reputation are one approach suggested by a number of authors (e.g. [8, 99]).

Creating trust between entities without any, or very limited, common history or knowledge of each other, which would be the case in an open MAS, is a non-trivial task. Even though PKI technology still has to prove itself viable for an open system on a global scale, a PKI may well be the best available solution for distribution of trust in an open multi-agent system [18].

2.3.10 Authorisation and delegation

Authorisation and delegation are important issues in multi-agent systems. Not only do agents need to be granted rights to access information and other resources in order to carry out their tasks, they will also be acting on behalf of a person, organisation, or other agents, requiring transfer of access rights between different entities. With a public key infrastructure in place, delegation can be done through various types of certificates, including attribute certificates for delegation of rights, and issuing of 'traditional' public key certificates for delegation of signing rights.

2.4 Conclusions

In this chapter the security issues existing for open multi-agent systems have been identified. The security issues are mainly related to agent execution and the fact that, since agents are autonomous and need to act upon information received from various entities, the trustworthiness of this information needs to be guaranteed by the system and considered by the agent. Security issues related to agent execution, and the fact that agents are under the control of a (perhaps untrusted) executing host, are particularly relevant to mobile agents.

Chapter 3

Security measures for multi-agent systems

Contents

This chapter describes available technologies and research efforts addressing security issues in multi-agent systems.

3.1 Introduction

Based on the security analysis of chapter 2, three broad categories of necessary security services for multi-agent systems can be distinguished, 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 existing technologies and research efforts addressing these three categories. In section 3.2 we will consider measures specifically addressing communication security issues for agents and MAS. The two following sections describe available technologies and research efforts addressing the security issues arising from the mobility property of mobile agents. Section 3.3 considers mechanisms addressing various security aspects of the mobile agent, and section 3.4 examines technologies protecting the executing hosts from misbehaving agents. Section 3.5 gives the conclusions of this chapter. Note that security services specifically relevant to multi-agent systems are discussed further in chapter 5. This latter discussion is given in the context of the agent security architecture introduced in chapter 4, whereas the discussion in this chapter focuses primarily on the existing work in the area

3.2 Security measures for agent communication

Many commercial and research MAS architectures have been implemented and many are still under development[1]. 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 VPN-like (Virtual Private Network) underlying network to provide security services. This approach usu-

[1]See http://www.agentbuilder.com/AgentTools for a list of available systems.

ally 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 (or perhaps even at the link level). The agents themselves are also unprotected from malicious hosts if no other security measures are applied.

Communication security can also be implemented at the session layer. [28] proposes the use of TLS (Transport Layer Security) to protect agent communication sessions. 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[2] is a non-profit standards organisation 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 (see, for example, [91]). Earlier, today outdated, FIPA documents did consider some security issues. However, the current standards do not deal with security. FIPA has recognised this and recently initiated work in the area[3]. [92] includes a very brief attempt to add security to a FIPA agent system, where it is suggested that the agent platform implements both authentication of agents and facilitators (entities offering certain services to agents) and the use of encrypted channels. However, no details are included, and how key management and authentication should be provided is not specified.

[2]Foundation for Intelligent Physical Agents, see http://www.fipa.org

[3]See `http://www2.elec.qmul.ac.uk/~stefan/fipa-security` for the state and progress of this work.

We consider the FIPA specifications further in chapter 7.

KQML (Knowledge Query and Manipulation Language) [48] is a message protocol designed to enable software agents to communicate with each other. The protocol has been developed as part of an ARPA project. KQML does not deal with security issues but depends on security being provided by lower layers. [106] proposes a security architecture for KQML. Symmetric or asymmetric cryptography is supported and keys are assumed to be agreed beforehand. The proposed extension provides for confidentiality, authentication, and (limited) data integrity protection. However, as pointed out by the authors, it does not protect against message replay attacks. A solution using mediating agents to enable communication with crypto un-aware agents is also proposed. Another suggestion for enhancing KQML with security is proposed in [55]. Parameters for certificate management are defined leaving the format of the certificate undefined.

3.3 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 in section 2.3.5, 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.

3.3.1 Contractual agreements

The simplest solution (at least from a technical perspective) 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.

3.3.2 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 [115]. 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 [27]) 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) [107] and Next-Generation Secure Computing Base (NGSCB) [101], 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.

3.3.3 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 despatch 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. This approach is further explored in chapter 9.

3.3.4 Co-operating agents

By using cooperating agents, a similar result to that of trusted nodes can be achieved [96]. 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 [100]. 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. Techniques based on these ideas are further described in chapter 9.

3.3.5 Execution tracing

Execution tracing [110] has been proposed for detecting unauthorised 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 (PRACs) were introduced by Yee [119]. 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 MAC

(Message Authentication Code) 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.

3.3.6 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 [120] 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.

3.3.7 Environmental key generation

Environmental key generation [95] 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 in [95]. 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.

3.3.8 Computing with encrypted functions

Sander and Tschudin [98] 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 therefore suggest combining the method with undetachable signatures (see section 3.3.10).

Although the idea is straightforward, the problem is to find appropriate encryption schemes that can transform functions as intended; this remains a research topic. Recently Barak et al. [4] have shown that obtaining theoretical justification for a program's ability to completely hide its information appears infeasible.

3.3.9 Obfuscated code

Hohl [57] proposes what he refers to as Blackbox security to scramble an agent's code in such a way that no one is able to gain a complete understanding of its function. However, no general algorithm or approach exists for providing Blackbox security. A time-limited variant of Blackbox protection is proposed as a reasonable alternative. This could be applicable where an agent only needs to be protected for a short period. One serious drawback of this scheme is the difficulty of quantifying the protection time provided by the obfuscated algorithm. Nevertheless, commercial applications are available that protect software by obfuscating the code in various ways [29].

3.3.10 Undetachable signatures

By binding usage restrictions to a signature key given to the agent, we can potentially limit the damage a malicious host can do. Sander and Tschudin [98] proposed one such scheme, which they refer to as undetachable signatures. Their original scheme has since been improved [81]. The idea is to encode constraints into the signature key. If the constraints are not met a valid signature is not produced, preventing arbitrary messages from being signed.

Undetachable signatures, as proposed by Sander and Tschudin [98], are based on the idea of computing with encrypted functions. The host executes a function $s \circ f$, where f is an encrypting function, without having access to the user's private signature function s. The security of the method lies in the encrypting function f. Whilst Sander and Tschudin were unable to propose a satisfactory scheme, more recently Kotzanikolaou, Burmester and Chrissikopoulos [81] have presented an RSA-based scheme which appears to be secure.

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

$$f(x) = m \text{ and } g(x) = s(m).$$

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 (see chapter 8) 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.

3.4 Protecting the agent platform

Generally speaking, more mature technology is available to address the problem of protecting the agent platform from malicious agents than is available to protect agents against malicious platforms. This problem also appears easier to solve than the malicious host problem. Techniques similar to those used to address security issues associated with downloading software from the Internet can be applied to the mobile agent scenario. We next describe some of these techniques.

3.4.1 Sandboxing and safe code interpretation

Sandboxing [49] isolates applications (or in our case agents) into distinct domains enforced by software. The technique allows untrusted programs to be executed within their own virtual address space thereby preventing them from interfering with other applications. Access to system resources can also be controlled through a unique identifier associated with each domain.

Agents are usually developed using an interpreted script or programming language. The main motivation for this is to support agent platforms on heterogeneous computer systems. The idea behind safe code interpretation is that commands considered insecure can either be made safe or denied to the agent. Java is probably the most widely used interpretative language used today. Java [86] also utilises sandboxing and signed code (described in section 3.4.3); this makes Java well suited for development of agents.

3.4.2 Proof carrying code

Proof carrying code [89] requires the author of an agent to formally prove that the agent conforms to a certain security policy. The execution platform can then check the agent and the proof before executing the agent. The agent can then be run without any further restrictions. The major drawback of this approach is the difficulty in generating such formal proofs in an automated and efficient way.

3.4.3 Signed code

By digitally signing an agent, its authenticity, origin, and integrity can be verified by the recipient. Typically the code signer is either the creator of the agent, the agent owner (on whose behalf the agent is acting), or some party that has reviewed the agent. The security policy at the recipient platform, perhaps in conjunction with attribute certificates supplied with the signed code, would then decide if a particular signature means that the code should be executed.

3.4.4 Path histories

The idea behind path histories [25] is to let a host know where a mobile agent has been executed previously. If the agent has been running on a host that is not trusted, the newly visited host can decide not to let the agent execute or to restrict the execution privileges. Path histories require each host to add a signed entry to the path, indicating its identity and the identity of the next platform to be visited, and to supply the complete path history to the next host.

3.4.5 State appraisal

State appraisal [39] attempts to ensure that an agent's state has not been tampered with and that the agent will not carry out any illegal actions through a state appraisal function which becomes part of the agent code. The agent author produces the appraisal function which is signed by the author, together with the rest of the agent. An agent platform uses the function to verify that an incoming agent is in a correct state and to determine what privileges an agent can be granted during execution. The theory, which is still to be proven in practice, requires that the legal states can be captured and described in an efficient and secure way.

3.5 Conclusions

The security issues for non-mobile agents can, at least in theory, to a great extent be tackled through existing security technology and protocols. However, issues related to trust and delegation in a large scale multi agent system are non-trivial to solve. Although a public key infrastructure is likely to be an important part of the solution, agents need to be able to reason and make decisions based on various security parameters. Execution of agents (mobile as well as non-mobile) on untrusted platforms is another factor introducing non-trivial security concerns, in particular related to correct agent execution and confidentiality of agent data.

There does not seem to be a single solution to the security problems introduced by mobile agents unless trusted hardware is introduced, which is likely to prove too expensive for most applications. The way forward appears to lie in a range of mechanisms aimed at solving particular (smaller) problems. This could, for example, include mechanisms that depend on agents executing on several hosts rather than on only one host, mechanisms and protocols binding agent actions to hosts, generation of various types of audit information that can be used in case of disputes, and so on. Solutions to certain problems do exist, but for mobile agents to be more widely adopted this is an area that requires further research.

Chapter 4

A security architecture for agent based mobile systems

Contents

In this chapter a security architecture for agent based mobile systems is proposed. The architecture has four levels of abstraction: involved parties, device structure, agent execution environment, and the agent.

4.1 Introduction

Within this chapter a model specifically designed for analysing security issues within an agent-based mobile system is presented. The model is presented at four different levels of abstraction. This enables all types of interactions, including those within a device and those spanning the entire mobile system, to be modelled. Some of the work described in this chapter has been previously published in [11].

This chapter contains four main sections, each containing a model for a different level of abstraction, starting with the highest level and working down. Specifically, the contents are as follows.

Section 4.2 describes the *Involved parties* and their roles. This section provides the highest level of abstraction in this chapter.

Section 4.3 considers the structure of a *Device*, one component of which is an agent execution environment. Note that we use the general term ‘device’ to include any hardware component which constitutes part of the mobile system. This covers any hardware component that uses wireless networking of some kind. Examples would include mobile terminals and parts of terminals as well as non-mobile computers that are part of the agent system.

Section 4.4 describes this *Agent execution environment* in more detail, focusing on security functionality.

In section 4.5 we consider the various parts of an *Agent* and how it interacts with its environment. Section 4.6 then gives the conclusions of the chapter.

4.2 Involved parties

This section describes the high level entities that can be distinguished in the security model. The parties can be thought of as distinct individuals or organisations. However, in practice one organisation can take the roles of more than one entity. It is also possible that not every party would be involved in a particular scenario.

Device user: also referred to as user. The user is assumed to have physical control over the device, but may not necessarily be the same entity as the device owner.

Device provider: the manufacturer of the device. In order for the manufacturer to offer upgrades or additional services, the device provider will typically share a security context with the device. This security context will typically involve shared secrets and/or the provision of 'root' public keys.

Device owner: this might, for example, be the user or the employer of the user. Again there is a trust and possibly a cryptographic relationship with the device. The rationale for distinguishing the device owner from the device user is the fact that they might have different objectives. An employer might, for example, want to restrict the use of a device in order to protect itself from various threats, such as malicious code.

Service Provider (SP): provides some kind of service, (e.g. transport service, information service, payment service, etc.) including directory services and remote agent execution environments to users and other SPs. A service provider may or may not have a pre-established contract with its clients.

'Home' Service Provider (HSP): i.e. an entity with which the device owner or device user has a contractual relationship. This gives the provider of services to the device an identifiable entity from which he can extract payment (the HSP will then present a bill for all services provided to the device owner). Note that a device owner may have many HSPs.

Trust Service Provider (TSP): a special class of service provider providing trusted third party services, e.g. a CA (Certification Authority), an RA (Registration Authority), a timestamping service, an electronic notary, etc. [65].

Agent provider: provides other parties with agents. The agent provider would typically be the developer of the agent. The agent provider can rely on its reputation or can issue other guarantees concerning provided agents' behaviour. The agent provider and the agent owner can be different entities. One can envision a scenario where software developers provide (e.g. sell) agents to users. The users would then only need to provide the agent with certain credentials and perhaps specify its objectives.

Agent owner: the entity on whose behalf an agent is executing. All parties can deploy agents to act on their behalf. These agents can execute on a device under the control of the agent owner as well as in other places within the infrastructure.

4.3 Device structure

This section describes the different parts of a device, including most importantly agents and the agent execution environment. A diagram of the device model is given in figure 4.1. Only one agent is shown in the figure, but a device would typically have multiple agents executing in the agent execution environment.

It should be noted that devices and their resources can vary greatly, and depending on their purpose might not include all the components described here. The device described is a mobile one, but a similar structure can be assumed to exist within other devices where agents are executed within the infrastructure.

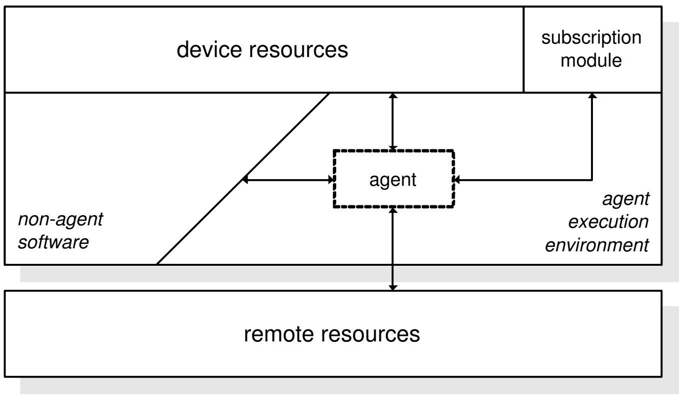

Figure 4.1: A model for the elements within a device

The constituent elements of the device model are as follows.

Agent: executable code which is acting on behalf of its owner. All parties can use agents to represent themselves. An agent can execute in an environment that is under the control of its (i.e. the agent's) owner, or it may execute in an environment provided by another party. Agents may or may not be mobile. As illustrated in figure 4.1, an agent can communicate with other entities over a network as well as with other functional blocks within the same device, including device resources, a subscription module, other agents, and other device software.

Agent execution environment: provides the resources required for agents to execute and communicate with other agents as well as with other resources and entities. The agent execution environment, further described in section 4.4, is regulated and controlled through mechanisms here referred to as agent control. Several agents can execute simultaneously within one environment.

Subscription module: a hardware device (e.g. smart card, USB token) which may interact with the device. An example of such a module is the GSM SIM [111]. This module would typically be provided by a Home Service Provider (HSP), and would share secret keys with a HSP and/or possess HSP-provided root public keys. Not every mobile device will be capable of directly interacting with such modules but some will, and hence it needs to be included in the model.

Remote resources: agents executing in a device can communicate with resources (e.g. agents and services) on other devices.

Non-agent software: software (applications and middleware functions) residing on the same device but not under the control of the agent platform control functionality. Agents can communicate with such software just as such software would be able to interact with agents.

Device resources: refers to other resources residing in the device. Examples of such resources include a user interface, a hardware crypto-engine, various cryptographic primitives that might be bound to the device (e.g. in the form of a cryptographic API), and communication resources.

4.4 Agent execution environment

In this section the security functionality of the agent execution environment is described. We are only considering security functionality here, and a complete execution environment would be more complex. A diagram of the agent execution environment is given in figure 4.2.

It should be noted that, depending on the device on which the agent execution environment is residing, not all the elements of this model may exist. A device might, for example, not support the downloading of agents – in which case the agent mobility service would not exist. The complete agent execution environment will include the following elements.

Agent management and control: is the governing security platform element. This element is responsible for managing all agents executing on the platform including monitoring and controlling access to resources as well as communication between agents executing on the local platform.

Agent communication services: provides communications facilities to agents executing within the environment. This includes secure communication services.

Agent security services: includes security services provided by the environment to executing agents. For example, the environment may add a digital signature to data (signed with the private device signature key) at the request of an agent.

Agent mobility service: enables agents to send themselves (and associated stored state) to other devices. The agent mobility service also includes functionality to assess received agents and any associated security information to decide if an agent shall be granted permission to execute on the platform. Agents requesting transfer to another platform will also be assessed for appropriate privileges by this entity. If required, the agent mobility service can add platform specific information (e.g. agent path history) before transmission. The agent mobility service is responsible for setting up secure transmission channels when required for agent transfers.

Event logging service: logs security relevant events for storage in an audit trail. It may also provide security intrusion detection based on processing of recorded events. Such detected events, or combinations thereof, would then result in a notification to another sub-system (e.g. agent management and control), which would then take appropriate action.

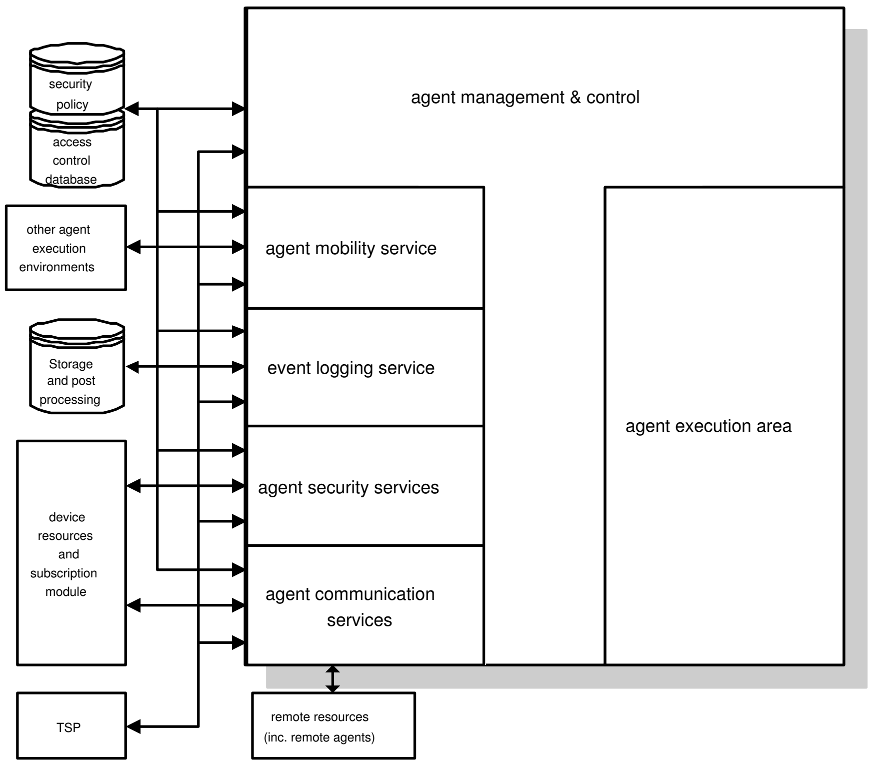
security policy
access control database
other agent execution environments
Storage and post processing
device resources and subscription module
TSP
agent management & control
agent mobility service
event logging service
agent security services
agent communication services
agent execution area
remote resources (inc. remote agents)

Figure 4.2: agent execution environment architecture

In addition to the described elements, which make up the execution environment, the elements/functionality below (which also appear in figure 4.2) are part of the architecture.

The security policy and access control database regulate the behaviour of the security mechanisms. One example of information making up the security policy could be a rule base describing how, and under what circumstances, agents can be given access to the execution environment, and can interact with each other and their environments. Other examples include the specification of security related events for which log entries should be generated, and what controls should be implemented in order for an agent to start execution. The access control database contains information governing how various resources can be accessed by the various parties. This information could, for example, be in the form of an Access Control List (ACL) [49] or a set of capabilities [49], or some combination of the two. This database can be physically located on the device, handled remotely over a communication link, or be implemented as a combination of local and remote facilities.

Remote systems can dispatch agents to the platform in order for them to be executed. In the same manner, the agent execution environment can dispatch agents to execute in other environments.

Log storage and post processing manages and processes log data once generated. This functionality can be physically located on the device, handled remotely over a communication link, or as a combination of the two.

Device resources and subscription module includes any resources (hardware and software) residing on the device.

Trust Service Provider (TSP) (as described in section 4.2) provides various trust services.

Remote resources are resources residing on other platforms with which agents can communicate, including other agents.

4.5 Agent construction

The various parts of an agent are likely to have different properties that need to be addressed through appropriate security mechanisms. The following distinctions between component parts of an agent can be made. Note that this agent model is designed for the purposes of security analysis only. As a result, important agent functionality may not be covered within this model.

Core executable part: information that can be executed. This information is distinguished from other information to allow a user to obtain an agent from an independent party (agent provider).

Payloads: An agent is likely to have various kinds of payloads. Payloads can consist of non-executable data as well as executable information required by the agent to fulfil its task. Execution state, information supplied by the agent owner, and information collected at various hosts (for mobile agents), are all examples of payloads of an agent. In addition to this, an agent can obtain executable payloads to add agent functionality that is not part of the core executable part.

By separating agent parts in this way integrity verification values can be created where appropriate. The use of the above distinctions becomes particularly important for mobile agents, but is also relevant for agents that are transferred to be executed on a platform not belonging to the agent provider. (As in section 2.2, we are here defining a mobile agent to be an agent that can move 'on its own initiative' and continue execution in the environment where it arrives.)

4.6 Conclusions

In this chapter a security architecture for agent based mobile systems has been proposed. The model includes the involved parties at the highest level, then a device structure, an agent execution environment, and finally, the structure of an agent.

Chapter 5

Security services and mechanisms for agent based mobile systems

Contents

This chapter describes security functionality for an agent based mobile system and outlines possible approaches for their realisation.

5.1 Introduction

In this chapter we specify a set of security services for agent based mobile systems within the context of the security model specified in chapter 4. The security services are divided into three main classes, namely security services to protect the execution platform, security services aimed at protecting agents, and communication security services.

Section 5.2 describes security functionality and where this functionality fits into the architecture. In section 5.3 possible approaches for how the security functionality can be implemented are outlined. Finally, section 5.4 gives the conclusions of this chapter.

5.2 Security functionality

This section specifies and motivates security services and mechanisms useful within an agent based mobile system. Section 5.2.1 defines security functionality applicable to platform protection, section 5.2.2 defines functionality for protecting agents, and section 5.2.3 defines security services for communication within the system.

5.2.1 Platform protection

In this section we will define security functionality primarily directed at addressing the security of the execution environment. Again, we note that agent execution environments will exist in various kinds of devices and the precise functionality, including security functionality, provided by the environment will also vary. Hence, the functionality described here may not be implemented in every device.

Logical access control

Multiple agents as well as other applications and system software are assumed to be executing on the same host and thereby be sharing system resources. The executing host needs to ensure that agents (and other software entities) are not interfering with each other or the system in unauthorised ways.

Logical access controls ensure that agents (or other software entities) cannot interfere with each other or the platform in unauthorised ways. The *agent management and control* element is the main entity within the agent environment architecture enforcing access control. However, access control is also part of the functionality of the *mobility service*, *event logging service*, *agent security service*, and *agent communication services.*

Authentication of foreign code

In order to provide flexibility a host will need to be able to receive, retrieve and execute agents. In fact, this applies to any kind of software, not only agents. In a mobile environment, with constant changes taking place, the ability to receive and execute software is likely to be very important functionality.

In order for an agent to be executed (with more than a very minimum of privileges), the origin of the code would need to be established. When an agent arrives at the execution environment it is handled by the *mobility service*. Here various security checks are made.

Authorisation of foreign code

Before an agent (or any other piece of downloaded code) is allowed to commence its execution, it needs to be authorised to do so. For any kind of action requested by the agent the host needs to ensure that the agent is authorised to proceed.

Authorisation issues would be resolved at various parts within the system, partly depending on the trust model and authorisation model chosen. However, the *mobility service* would make an initial decision as to whether an agent should be allowed to commence execution and a *trusted service provider* is likely to be involved in issuing authorisation credentials for agents.

Secure communication

The platform will be required to communicate with other entities within the infrastructure. For example, agents will be transferred to and from other platforms, and various trusted service providers need to be contacted. Depending on the nature and sensitivity of the communication, various levels of protection will be required. Requirements and motivation for communication security services are further developed in section 5.2.3.

Any of the elements capable of communicating with other parties would be responsible for providing communications security services.

Event logging

As opposed to most other security features, which are in place to prevent security breaches, auditing and monitoring enable follow-up when something goes wrong. The main purpose of audit trails is to provide information that can be examined at a later point in time. Examples of applications for audit data include fraud detection, intrusion detection, follow-up in case of failure, and follow-up in case of a security breach. Audit information can also be used for real-time monitoring in order to take immediate actions in case of security violation.

The *event logging service* within the agent execution environment is responsible for generating audit trails. The *security policy* governs the definition of security events to be logged. (As described in section 5.2.2 below, audit events can also be generated at the initiative of an agent.)

5.2.2 Agent protection

In this section we define security functionality that is offered to agents (mobile and non-mobile). Some functionality is offered as services by the executing platform or by a third party, while other functions would be implemented in the agent itself.

Access control

An agent's information needs to be protected from unauthorised access by other entities executing on the same host, as well as from remote entities. An agent might also need protection from the host on which it is executing.

Logical access control is enforced by various elements on the host, but mainly by the *agent management and control* element. Certain information and execution can be protected by physical means through the use of a *subscription module.*

Agent authentication

In order for an agent to be sure about the identity of another agent, agents need the ability to authenticate each other, and to authenticate themselves towards an agent platform and towards service providers.

The *agent security services* element would help an agent to prove its identity, at least to the extent of proving the agent's location.

Platform authentication

Platforms will need to authenticate themselves to agents on other hosts, agents executing on the host itself, and to other hosts.

Depending on the purpose of this authentication, the *agent mobility service*, the *agent security services* and the *agent communication services* are the elements most likely to be required to prove their identities to an agent.

Authorisation

Depending on an agent's task and purpose, it might need to determine if a platform or agent should be granted access to certain information, or whether certain actions should be taken by the agent itself.

Such authorisation decisions would need to be taken by the agent itself or specifically delegated to another party by the agent.

Secure agent mobility

The model proposed for the agent based mobile system allows for agent mobility as well as downloadable agents. A mobile agent can (on its own initiative) move between platforms, while a downloadable agent would be pulled by the device or pushed by a third party onto the device, usually to enhance the functionality of the device in some way.

Functionality involved when agents are transferring from one host to another is offered by the *mobility services.* Whatever criteria are used for agent mobility, a protected audit trail can be very important to prove that a host has acted in an illicit manner. The *event generation* functionality is explained further below.

Agent communication

Agents have capabilities to communicate with other agents as well as with other entities that exist on the same device as the agent or on remote devices. This communication will, depending on its nature and sensitivity, need various levels of protection. Agents need to be offered the ability to be 'security aware'. Even if security functionality is not directly implemented in the agent, the agent may need to be able to access certain security functionality/properties from the executing platform, as well as being informed about the circumstances under which information has been transferred. Such information is important to let agents decide the origin of data and its trustworthiness.

Inter platform communication is protected by *agent management and control* and intra platform agent communication is provided through *agent communication services.* (Intra platform agent communication is further covered in section 5.2.3.)

Event Logging

In a system where proper security is difficult to achieve or where the level of security is traded off against performance criteria, (as is likely to be the case with mobile agents) having transaction data that can be examined at a later point in time can be very important. Not only can this information be examined when needed, it can also deter illegal behaviour. Agents need the ability to generate audit trails that can be examined if needed.

Events logged locally, or processed by the local environment (and possibly then dispatched elsewhere), are handled by the *event logging service.* If the logging functionality is handled completely by the agent no other services would have to be involved apart from the *agent communication services* if events are dispatched to other agents/platforms and perhaps a *non-repudiation service.*

Anonymous execution

The possibility for users to have agents acting on their behalf without giving other parties the means to trace the actions to the user is potentially an important factor for user acceptance (anonymity also introduces vulnerabilities, including denial of service attacks, see section 2.3.8).

No specific service on the execution device is necessarily required to allow agents run without a traceable user. (Anonymous transactions are covered in section 5.2.3.)

5.2.3 Communication security services

In this section we will briefly describe communication security services and the motivation for their use within a multi-agent system.

Integrity of communication

Integrity protects against improper modifications of messages, duplication of messages, deletion of part or all of the messages in a sequence, or reordering of parts of a message sequence.

Integrity is closely linked to origin authentication (see below). Without an authenticated source for data, assurance of maintained integrity does not usually mean much. Data origin authentication also requires the provision of data integrity. However, certain integrity properties might be of less importance depending on the application (for example, message duplication may be a relatively minor threat for many applications).

Authentication

Authentication is fundamental for secure communication, and is about the assurance of the claimed identity of an entity. Two forms of authentication can be distinguished:

- Entity authentication, the corroboration that an entity is the one claimed. Applies to connection oriented communication.
- Data origin authentication, the corroboration that the source of data received is as claimed. Applies to connectionless communication.

Agents as well as platforms need to be authenticated as well as be able to ensure the identity of another party. This entity authentication service is normally provided by the use of an appropriate *authentication protocol* [87]. For the purpose of authentication protocols, no distinction need be made between agents and platforms (a platform can in practice be represented by an agent).

For the envisioned system, i.e. a multi-agent system in a mobile telecommunication environment, entity authentication as well as data origin authentication are required. Data origin authentication is, for example, required for simple message passing between agents. Entity authentication can be required when an agent is involved in real-time data provision, such as real-time video.

Non-repudiation

Non-repudiation is about preventing entities denying their actions. Non-repudiation can be regarded as an extension to authentication. Digital signatures often play an important role in non-repudiation protocols. In addition to this, the provision of non-repudiation requires some sort of agreement regulating what constitutes a digital signature and the meaning of a signature, the definition of a third party that can settle disputes, and additional functionality to bind actions to a point in time (e.g. timestamps or a trusted deposit).

Non-repudiation can be applied to various actions; for a multi-agent system, non-repudiation of origin and non-repudiation of receipt are believed to be of most relevance. Non-repudiation of origin protects against an originator's false denial of having created the content of a message and of having sent a message [62]. Non-repudiation of receipt protects against a recipient's false denial of having received a message [62].

Confidentiality of communication

Confidentiality of communication is about ensuring that unauthorised parties cannot access information in transit. For communication between agents executing on separate hosts, encryption is typically required for confidentiality protection. As for integrity, confidentiality can be applied per message or per session.

Anonymity

Anonymity services allow a user to perform actions without being tracked or associated with a transaction. The possibility of anonymous transactions is potentially an important factor for user acceptance.

5.3 Providing security functionality

In this section we will consider how the security functionality described previously in this chapter can be provided.

5.3.1 Platform protection

Logical Access Control

The platform needs to protect itself and its hosted agents against unauthorised access. Such functionality is often implemented in existing operating systems and execution environments. It can, at least partially, be implemented by using the concept of sandboxing and safe code interpretation (as described in section 3.4.1), where executable code (e.g. an agent) would be able to do anything within the sandbox while any actions involving resources outside the sandbox are closely regulated and monitored. With this approach only limited efforts need to be spent on ensuring the correctness of the received code.

However, using the sandbox technique alone does not address all the access control issues. In order to make full use of agents they need to be able to access resources outside the sandbox. Resources outside the sandbox include resources located on the same physical device as well as the ability to communicate with other devices/hosts/agents. Also, even within the sandbox, code is using resources, and hence access controls need to exist even when sandbox techniques are employed.

We assume that the execution environment has a security policy that regulates the requirements under which an access request will be granted. It is reasonable to assume that an access control list (ACL) [49] in combination with a capability-based scheme [49] will be required to provide the necessary access control services. While ACLs are typically rather static in nature (although there is nothing to stop dynamic changes to the list) a capability scheme allows a subject to provide the required information at the point of an access request.

A capability scheme based on public key cryptography and a PKI would allow for the required delegation and transfer of rights between parties.

Authentication of foreign code

Use of the sandbox technique and safe code interpretation on their own can only be used to limit agent access to local resources. This is not enough to provide more powerful functionality. Applications will need to be given access to resources that, if misused, can result in unauthorised actions. In recent years research has been performed on *proof carrying code* (described in section 3.4.2). This research aims to verify that a piece of code is secure before it begins execution. However useful this would be, this is still very much an emerging area, and it is not clear how feasible it would be to restrict agents to those which have formal proofs of security. A more pragmatic approach is to trust a particular piece of software because one decides to trust the developer/supplier of the software. This technique is used in Java [86] as well as in MExE [1]. Using this technique we need ways of verifying that a particular piece of software does originate from a party we are prepared to trust, at least for the purposes of code execution. This can be done through cryptographic means.

When an agent arrives at the execution environment it is handled by the *mobility service*. Here various security checks are made. The following information associated with the agent can, for example, be verified and used by the mobility service, in conjunction with its own policy information, in order to decide whether an agent should be granted execution rights:

- agent owner,

- agent provider,
- policy information associated with the agent,
- required resources,
- submitting host,
- path history.

Event logging

Once audit data is generated it needs to be stored and properly protected. Storage can be at the local platform but can also be sent to a trusted party or other remote storage. If the security of the platform is compromised it can potentially be valuable to have transferred the audit data prior to the attack. This does of course generate network traffic and hence is not always the preferred option.

Once audit data is generated and stored it can be analysed. The analysis can be an automatic process e.g. looking for patterns or anomalies, or it can be carried out manually, e.g. in the event of a security breach.

5.3.2 Agent protection

First note that certain aspects of agent protection, in particular focussing on protecting the means for agents to create signatures and commit to transactions, are considered in much greater detail in chapters 8 – 11.

Access control

In order to protect an agent from other agents the logical access control mechanisms described in section 5.3.1 would apply. The problems involved when trying to protect information existing on a device from a determined user who has physical control over the device are non-trivial. One possible solution to this problem is a protected chip; in the case of GSM and UMTS this takes the form of a smart card [111]. The model used for deploying agents in an open network setting should not require the use of such security measures but should nevertheless allow them to be used.

Physically protected devices exist with various functionalities and with different interfaces and formats. The simplest can only be used for protected storage while more sophisticated also have processing capabilities. The complete agent execution environment can in theory be implemented in a protected device. More realistic, however, is to have a protected device with very limited storage and processing power that can be used for critical information storage and processing.

Agent authentication

Agents need the ability to authenticate each other, and to authenticate themselves towards an agent platform and service providers. As long as the host on which the agent is executing is trusted not to misuse any authentication information, existing authentication protocols can be used for this purpose. These protocols can, for example, be based on shared secrets or public key cryptography and can be implemented independently from the agent platform. In the case where the host that is executing the agent is not trusted with agent authentication information, the solutions will greatly depend on the application. Certain applications should probably not be implemented using mobile code when the executing platform cannot be trusted.

Platform authentication

Once an agent is executing on a platform it is in theory too late to determine if the platform is authentic or not. If it is a hostile platform, the platform can trick the agent into executing anyway. However, by involving a Trusted Service Provider (TSP), certain functionality to ensure that a host will act in an honest manner can be provided. Platform authentication mechanisms similar to those used for agents can be employed.

Authorisation

As for authentication, once an agent is executing on a platform it is in theory too late to determine if the platform is authorised to execute the code or not. This can potentially also be solved by involving a TSP that carries out authentication and checks if the platform is authorised before giving the platform full access to the executable code. Other possible solutions might involve cryptographic approaches, where a host is required to possess the correct decryption key in order to be able to interpret the code correctly.

Secure agent mobility

Although mobility could be functionality built into an agent using standard communication services, this is likely to prove too cumbersome. The execution environment can therefore offer a mobility service.

An agent can carry code that should not be disclosed to an untrusted party. One way to ensure that an agent and its payloads are not disclosed is to encrypt the agent when in transit and ensure that the agent is only sent to trusted hosts. The challenge is then to decide what constitutes a *trusted* host. The agent can carry a list of hosts that are considered to be trusted. Alternatively, the agent can carry a list of trusted Environment Certifiers (ECs), generated by a TSP that vouches for the behaviour of the host (and its operator). A host would then have to show, (e.g. produce a digital certificate and signature) that it is certified. In both cases we trust that a trusted host would not let the agent move to an environment that is not trusted.

The fundamental problem that a trusted host must be trusted not to disclose the agent to an un-trusted party cannot be avoided. However, by involving TSPs we can minimise the information sent to the trusted host and/or move the decision whether a particular host should be trusted to the trusted party. In any case, a host will need the means to be authenticated, be it authentication to another host, an agent on another host, or even to an agent already executing on the host.

Agent communication

For communication between agents executing on separate hosts, encryption is typically required for confidentiality protection. An agent can include the required functionality for encryption/decryption and its associated key management functionality. This might be a desirable solution for certain applications. However, for performance (and management) reasons it is more efficient to let encryption/decryption be provided as a service by the agent's execution environment. Depending on the device, this might be implemented in software only, but may also be assisted by special hardware. Similarly, integrity functionality can be implemented within the agent, but it is likely to be more efficient when realised as a service.

For communication between agents executing on the same host, confidentiality and integrity of communication are assumed to be provided by the platform. The platform needs to separate agents in such a way that information leakage and manipulation are prohibited. To further protect the communication between agents on the same host would not make sense, since the party in physical control of the host (who would be the only other potential threat) will have direct access to agents anyway. If further protection is required physical measures must be used.

Event Logging

Agents need the ability to generate audit trails that can be examined if needed. How to handle such information for agents, and in particular for mobile agents, is not straightforward; the following options are possible for a mobile agent:

Store audit information in the agent's payload. This would generate some extra communication but would not necessarily require involvement of extra functionality or parties. The carried audit information can, however, be manipulated and possibly accessed by hosts visited after generation. Cryptographic techniques, such as those described in section 3.3.5, can be used to protect such payloads. In such a case, additional computational resources would then be required.

Store audit information at the host where it is generated. No extra network traffic would be incurred but hosts would be relied upon to store the information for a certain period. It might not be in the interest of the host to keep the information. It could also potentially become cumbersome to retrieve the information when needed if an agent's audit trail is spread out over several hosts.

Send audit information to agents on other hosts. Multiple agents can be used in cooperation in order to minimise the reliance on a single, possibly ill-behaved, host. This approach will incur additional network traffic but no additional infrastructure would be required.

Send audit information to a trusted service provider. Extra network traffic would be generated and some sort of agreement must exist between the agent owner and the TSP.

The approach chosen would depend on the application. An open MAS would probably need to provide for all such solutions.

Non-repudiation

A non-repudiation service can prove very valuable in a system where agents are representing users in various scenarios. This can be provided through cryptographic protocols that ensure one or more parties cannot later deny that a certain transaction took place. This particular security feature can be split into two separate problems, that of non-repudiation of origin and non-repudiation of receipt. Non-repudiation services can help to make audit information more credible and useful, and also to provide an infrastructure for binding contractual agreements of various kinds.

Non-repudiation can be achieved using digital signatures and a supporting public key infrastructure. It would be straightforward for an agent executing in an environment controlled by the agent's owner to produce non-repudiable material. However, when an agent is executing on a device under someone else's control this becomes non-trivial. The executing host could then make the agent produce a signature that would not have been produced if the agent was executed properly. One partial solution to this problem is to also let the executing host add a digital signature to the material. However, what such a signature would achieve in practice is not clear. In case of a dispute it might not be trivial to determine if the agent has been executed in a proper manner or not. This solution would, however, allow the host on which the agent produced the signature to be traced.

Further, it would be useful to distinguish between information that the host understands and information that the host does not understand. For example, a signature produced on some information provided by an agent could be used to show that the agent was executing on the host. However, it could not be used to hold the host (or its owner) responsible for the signed content. On the other hand, a host might, for example, be requested to sign information saying that an agent is executing on the host at a particular time, in which case the host (or its owner) should be accountable. In this case the host needs to 'understand' what it is signing. A verifier must then also be able to distinguish between such information.

A trusted service provider would need to be involved at some stage of a non-repudiation protocol. It can simply act as a settling entity in case of disputes or be involved in the actual production of non-repudiable material. A time stamping service might also be required for certain non-repudiation material (see section 6.5).

The *agent security services* would be able to generate signatures on the host's behalf. Such a signature could be produced at an agent's request or automatically as part of a protocol.

The means for agents to create non-repudiable commitments are considered in much greater detail in chapters 8 – 11.

Anonymous execution

The agent based system can provide for anonymous execution by (1) allowing execution of agents, without identification of the agent owner; and (2) by the usage 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 if such agents and their transactions would be allowed onto the device.

5.3.3 Communication security services

This topic is covered in considerably more detail in chapters 6 and 7.

Authentication and integrity of communication

By digitally signing content, information can be linked to its signer. With a supporting PKI, information can also be linked to parties without pre-established relations. This is crucial functionality for provision of authentication in a scalable multi-agent system. A digital signature will also (if properly implemented) provide integrity protection for the signed message.

An agent can be equipped with its own private key. Its signatures can then be verified by anyone who can obtain a copy of the corresponding public key. Likewise, agent platforms can be equipped with their own private keys to let them produce digital signatures. If pre-established relationships exist, MACs can be used instead. However, for mobile agents these techniques might prove insufficient. Sections 6.3 and 6.4 further describe how authentication techniques can be applied to agent communication.

Non-repudiation

Non-repudiation can be achieved by various means. The solution should, of course, depend on the application of the non-repudiation service. A service aimed at protecting transactions involving high monetary values can be allowed to have high transaction costs and delays, while these factors should be kept to a minimum when very low value transactions are involved. For agent-based middleware, a service supporting small value transactions is believed to be valuable. Although high value transactions should not be prohibited, these might be better dealt with as an 'application service'. Section 6.5 further describes how a timestamping service can be used to provide a non-repudiation service in a multi-agent environment.

Confidentiality of communication

For the application of agents, and particularly mobile agents, asymmetric cryptography has some very useful properties. An agent sending a confidential message does not need to carry a secret key. It can use the public key of the party with which it communicates to encrypt the data – this then means that access to an agent (and hence to the public key it carries) does not reveal messages previously encrypted by that agent. Depending on the application, it might be possible to give an agent the public key it needs to use for its communication. However, in a more dynamic scenario, agents need to be able to obtain the public key of an arbitrary entity (perhaps within certain domains). By using public key cryptography the problem of keeping the cryptographic keys secret is removed. However, it is crucial to ensure the relationship between a public key and the owner of its corresponding private key. Confidentiality of communication is further developed in section 6.6.

Anonymity

User anonymity can be achieved through various means. If an agent cannot be traced to its owner through any of the information carried in its communication the owner remains anonymous. However, it might be possible to trace a user through the identity of an agent, in which case the agent identity would need to be changed into one that cannot be traced to the user. This approach is further developed in section 6.7.

5.4 Conclusions

In this section basic security functionality addressing the security of agents, agent platforms, and agent communication has been identified. High level outlines of how these security features can be realised have also been given. This provides a context for the more specific discussions making up the remainder of this thesis.

Chapter 6

Agent communication security

Contents

This chapter describes, within the context of the security model presented in chapter 4, how agent communications security can be provided in a MAS.

6.1 Introduction

An agent's ability to communicate is fundamental and essential for it to function in a multi-agent system. In this chapter we describe how secure communication between agents on different platforms can be addressed within a multi-agent system in general, as well as within the security architecture presented in chapter 4. The basic cryptographic tools used to provide secure communications, as well as the notation used in this chapter have been described in section 1.4. The security services for agent communication and their motivation have been described in section 5.2.3.

Section 6.2 points out some architectural issues. Sections6.3 to 6.7 respectively describe how specific security services, namely data origin authentication, entity authentication, non-repudiation, confidentiality, and anonymity can be provided. Finally, the conclusions of this chapter are presented in section 6.8.

6.2 Architectural issues

In this section we will consider in general terms where security services can be implemented and how the architecture presented in chapter 4 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 described in chapter 4 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 CA issuing digital certificates.

In the agent execution environment architecture (see chapter 4), 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).

6.3 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 (section 6.3.1), and then for mobile agents (section 6.3.2).

We will use the following scenario to illustrate the message exchanges (see figure 6.1). 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.

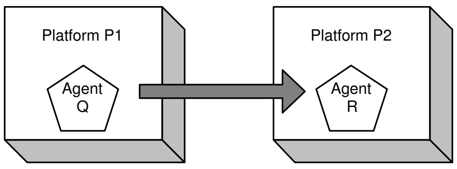

Figure 6.1: Agent Q sending a message to agent R

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.

6.3.1 Stationary agents

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

$$\mathrm{Q} \rightarrow \mathrm{R} : m \| s_Q(m)$$

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 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.

$$\text{Q} \rightarrow \text{R} : m \| MAC_{K1}(m)$$

If an agent is executing on a host that is trusted and the agent's secrets (e.g. cryptographic keys) 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.

$$\text{Q} \rightarrow \text{R} : m \| s_Q(m) \| s_{P1}(m \| s_Q(m))$$

Or alternatively, if shared secrets exist

$$\text{Q} \rightarrow \text{R} : m \| MAC_{K1}(m) \| MAC_{K2}(m \| MAC_{K1}(m))$$

where $K1$ is a secret shared between Q and R, and $K2$ is a secret shared between P1 and R.

Combinations of digital signatures and MACs are, of course, also possible.

The above protocols do 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 (number used once). 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:

$$m = m_1 \| R_{ID} \| nonce$$

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 in [75], and general mechanisms using digital signatures for entity authentication are further described in [74].

To protect against delay attacks, timestamps or a challenge-response protocol (see section 6.4) would be required.

6.3.2 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 (legally) executed, can at a later time still produce a valid agent signature. (Perhaps we have decided that we trust the platform not to perform such an attack when sending the agent there, in which case no further measures would be necessary).

Figure 6.2 shows the path of a multi-hop agent, moving from its original platform

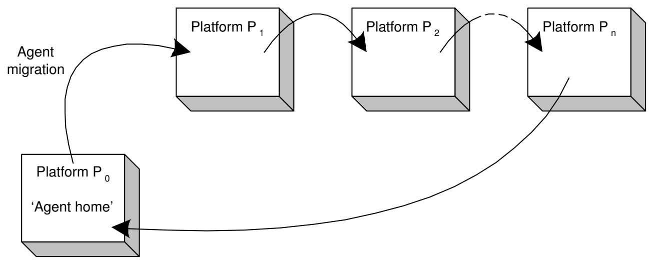

Figure 6.2: Movement of a mobile agent

(P_0), the 'agent home', to platforms $P_1, P_2, \ldots$ and finally to P_n before returning to, or sending a final message to, the 'agent home' P_0. The 'agent home' is the agent's origin and is assumed to be fully trusted by the agent (and its owner). This could, for example, be a user's mobile terminal. If the protocols described in section 6.3.1 are used, any platform from P_1 to P_n would be able to produce a 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.

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 6.3, 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.

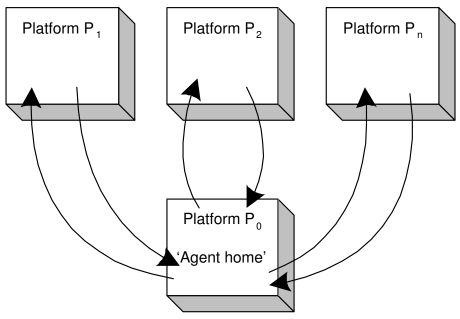

Figure 6.3: Deployment of multiple one-hop agents

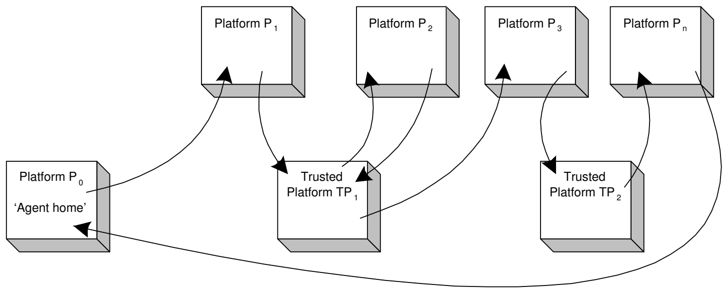

Figure 6.4: Using trusted platforms to update the agent's key

Figure 6.4 shows how trusted platforms can be inserted in the path and used to update the agent's key (or any other information that should not be unnecessarily exposed). 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.

Undetachable signatures

Another (partial) solution is to associate a signature key with certain constraints. Sander and Tschudin proposed one such scheme in [98], which they refer to as an undetachable signature. This was described in more detail in section 3.3.10. This prevents signatures form 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. This approach is further described in chapter 8.

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 [102].

Threshold cryptography was first proposed by Desmedt [33]. 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 l 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 ElGamal and RSA signatures (see, for example, [103] for a short survey). Recently Shoup [103] 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 (as it relies on a trusted party to form the shares) 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*. The construction of such a scheme is described in chapter 11.

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. This scheme is further described, along with its practical advantages, in chapter 10.

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 [96].

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 – for details see sections 3.3.7, 3.3.8, and 3.3.9. 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.

6.4 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.

As in section 6.3, we suppose that agent Q, residing on platform P1, needs to authenticate itself towards agent R, residing on platform P2 (see figure 6.1).

Below is an example of a mutual authentication protocol using digital signatures [74].

$$\begin{aligned}
\text{Q} \rightarrow \text{R} &\ :\ N_Q \\
\text{R} \rightarrow \text{Q} &\ :\ N_R \| s_R(N_R \| N_Q \| Q_{ID}) \\
\text{Q} \rightarrow \text{R} &\ :\ s_Q(N_Q \| N_R \| R_{ID})
\end{aligned}$$

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 (section 6.3) an application may also wish to authenticate the originating platform. This can be achieved by also letting the platform add its signature as follows.

$$\begin{aligned}
\text{Q} \rightarrow \text{R} &\ :\ N_Q \\
\text{R} \rightarrow \text{Q} &\ :\ N_R \| s_R(N_R \| N_Q \| Q_{ID}) \| s_{P2}(N_R \| s_R(N_R \| N_Q \| Q_{ID})) \\
\text{Q} \rightarrow \text{R} &\ :\ s_Q(N_Q \| N_R \| R_{ID}) \| s_{P1}(s_Q(N_Q \| N_R \| R_{ID}))
\end{aligned}$$

For mobile agents, as in the origin authentication case, multi-hop agents can be avoided (see section 6.3.2) 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 in section 6.3.2, *undetachable signatures* and *distributing trust among multiple platforms* do not appear to be applicable to entity authentication for mobile agents.

6.4.1 Architecture support for authentication

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

As stated above (in section 6.2), 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.

6.5 Non-repudiation

As mentioned earlier (in section 2.3.4), one can distinguish non-repudiation of many actions. Generic protocols for non-repudiation services have been specified in a series of ISO standards [62, 63, 64]. 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 in section 6.3.1. 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 TSP, as proposed immediately below.

A simple non-repudiation service can be achieved as an extension to the authentication protocols given in section 6.3.1. A TSP can offer a non-repudiation service simply by offering a *timestamp and forward* service;

$$\begin{aligned}(\mathrm{M1})\ \mathrm{Q} \rightarrow \mathrm{T} &\quad : \quad R_{ID} \| m \| s_Q(R_{ID} \| m)\\ (\mathrm{M2})\ \mathrm{T} \rightarrow \mathrm{R} &\quad : \quad m \| s_Q(R_{ID} \| m) \| \mathrm{TIME} \| s_T(m \| s_Q(R_{ID} \| m) \| \mathrm{TIME})\end{aligned}$$

where 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 (agents, platforms, etc.) 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) 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, see e.g. [3, 88]. An attractive property of the protocol proposed in [88] is that a trusted third party only needs to be involved when the protocol is failing.

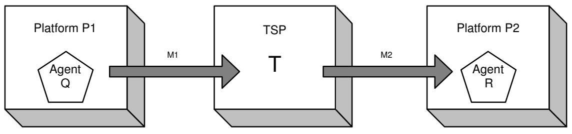

Figure 6.5: TSP offering a timestamp and forward service

6.5.1 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 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.

6.6 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.

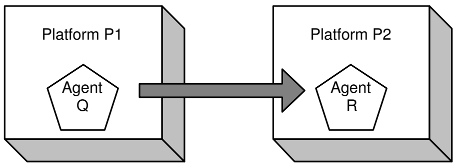

Figure 6.6: Agent Q sending a message to agent R

Q can simply, before sending the message to R, encrypt the message using R's public key as follows.

$$\text{Q} \rightarrow \text{R} : E_R(m)$$

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 (or agents), this can be a more efficient solution. That is, we have:

$$\mathrm{Q} \rightarrow \mathrm{R} : E_{K1}(m)$$

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.

$$\mathrm{Q} \rightarrow \mathrm{R} : E_R(K) \| E_K(m)$$

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 TLS protocol [35] can be used, where TLS is a standardised protocol for initialising and encrypting a session [28]. TLS also supports unilateral or mutual authentication of the communicating parties.

To achieve traffic flow 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 (hiding real communication time and frequency), or agents can send dummy traffic to multiple agents in order to conceal the real recipient.

6.6.1 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.

6.7 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 for 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 will now use the FIPA protocol, further described in chapter 7, to outline how anonymity can be provided to agent communication through a proxy service.

A FIPA 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 *from* 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 (e.g. the platform's IP address).

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 6.7.

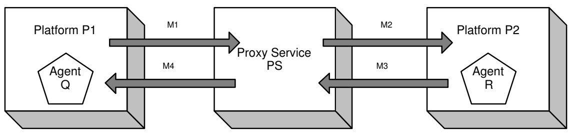

Figure 6.7: Proxy service giving anonymity

$$\begin{aligned}
&\text{(M1) Q} \rightarrow \text{PS} &:& \quad from : \text{Q@P1}\|to : \text{PS}\|xto : \text{R@P2} \\
&\text{(M2) PS} \rightarrow \text{R} &:& \quad from : \text{A@PS}\|to : \text{R@P2} \\
&\text{(M3) R} \rightarrow \text{PS} &:& \quad from : \text{R@P2}\|to : \text{A@PS} \\
&\text{(M4) PS} \rightarrow \text{Q} &:& \quad from : \text{PS}\|to : \text{Q@P1}\|xfrom : R@P2
\end{aligned}$$

In message (M1) indicates *xto* to whom PS shall forward the message and in message (M4) indicates *xfrom* who originally sent the message.

The protocol, which works in a similar way to established Internet mail remailers[1], 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 [24] or one of its variants, e.g. [51, 77, 105], 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 intercepter, 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.

[1]See: `http://www.andrebacard.com/remail.html` for general information on Internet mail remailers.

6.7.1 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 (or the owner's identity) would be exchanged for a proxy identity. For a communicating agent (mobile or stationary) 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.

6.8 Conclusions

In this chapter we have shown how security protocols can be used to provide secure communication within an agent-based system. Data integrity, data origin authentication, entity authentication, non-repudiation, confidentiality, and anonymity can all be provided for the communication.

For mobile agents it is a non-trivial task to ensure that communication originating from the agent cannot be spoofed. Several techniques to limit the threats posed to mobile agent communication exist. Nevertheless, mobile agents must be deployed with great care if the authenticity of the communication is of importance.

Chapter 7

Securing FIPA agent communication

Contents

This chapter evaluates the FIPA agent communication protocols and outlines how security functions can be added.

7.1 Introduction

FIPA[1] is a non-profit organisation promoting the use and development of intelligent agents by openly developing specifications supporting interoperability among agents and agent-based applications. FIPA's specification for agent communication has become a *de facto* standard. Although earlier, today obsolete, FIPA specifications did consider security for agent communication [41, 43], security is not covered in the current specifications [44]. It appears that the FIPA specifications and the general state of agent research are now mature enough to require security services for agent communication. FIPA has recognised the need for improved security and initiated work in the area[2]. Most of the work described in this chapter has been previously published in [13].

In this chapter we will evaluate the FIPA specifications from a security point of view and propose extensions to the specifications in order to provide security services for agent communication. We will address the following security services for agent message communication:

- **Integrity**, protects against improper modification of a message.
- **Origin authentication**, the corroboration that the source of data received is as claimed.
- **Confidentiality**, guarantees that unauthorised parties cannot access information in transit.

Non-repudiation, which can be considered as a *stronger version* of origin au-

[1]Foundation for Intelligent Physical Agents, see `http://www.fipa.org`

[2]See `http://www2.elec.qmul.ac.uk/~stefan/fipa-security` for the state and progress of this work.

thentication, will not be further addressed in this chapter. Non-repudiation (of data origin and data reception) can be achieved using authentication mechanisms in combination with timestamps and a third party to settle disputes (as described in section 6.5).

We are still assuming a multi-agent system in an *open environment*, that is, a system where no single authority is in control of the system, which means that agents (and other entities) cannot be assumed to act in a perfectly honest manner. We are also assuming the existence of a supporting Public Key Infrastructure (PKI) for management of certified cryptographic public keys. The precise requirements for a PKI for an open multi-agent system is a research topic in itself.

The chapter is structured as follows. In the next two sections we briefly describe the FIPA agent communication specifications, first the communication model and then the message structure. Section 7.4 describes the Open PGP message format and how this can be applied to agent communication. In section 7.5 we consider the required interaction between an agent and its platform if the platform is to provide secure communications services to the agent. Finally, section 7.6 gives the conclusions of this chapter.

7.2 The FIPA communication model

Figure 7.1 shows the FIPA message transport reference model [44].

The message transport service is a service typically provided by the agent platform on which an agent is executing. However the agent and the message

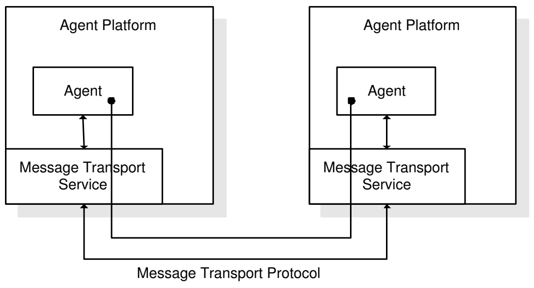
Agent Platform
Agent
Message Transport Service
Agent Platform
Agent
Message Transport Service
Message Transport Protocol

Figure 7.1: FIPA message transport model

transport service do not have to be located on the same host. The message transportation service supports the transportation of messages between agents on any given agent platform and between agents on different platforms through the provision of an Agent Communication Channel (ACC). FIPA recognises three options for an agent when sending a message to another agent residing on a remote platform (illustrated in figure 7.2) [44].

1. Agent A sends the message to its local ACC using a proprietary or standard interface. The ACC then takes care of the transmission of the message to the correct remote ACC. The remote ACC will then eventually deliver the message.

2. Agent A sends the message directly to the ACC on the remote agent platform on which B resides. This remote ACC then delivers the message to B.

3. Agent A sends the message directly to agent B, using a direct communication mechanism. The message transfer, including buffering of messages and any error messages, must be handled by the sending and receiving agents. (No further specification for this communication mode is covered by FIPA.)

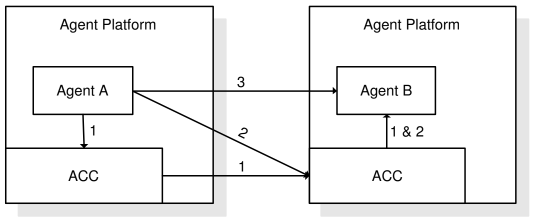

Figure 7.2: Methods of communication between agents on different platforms

7.3 FIPA Message structure

In this section we will describe the structure of a FIPA message [44] and then consider how security is addressed.

A message is made up of a message envelope, containing transport information, and a message body comprising of the agent communication data or ACL (Agent Communication Language) message. Table 7.1 shows the structure of the message envelope. An ACC should deliver the whole message, including the message envelope, to the receiving agent. However, it is possible for agent platforms to provide middleware layers to free agents from the task of processing the envelope [44].

Table 7.1: Message envelope description

Parameter	**Description**
to	Names of primary recipients of the message, mandatory.
from	Name of the agent who sent the message, mandatory.
comments	Optional comment.
acl-representation	Name of the syntax representation of the message body, mandatory.
payload-length	The length of the message body, optional.
payload-encoding	The language encoding of the message body, optional.
date	Message creation date and time, mandatory.
intended-receiver	The name of the agent to whom this instance of a message is to be delivered, optional.
received	Time received by an ACC, optional.
transport-behaviour	Transport requirements of the message, optional.

A earlier, now obsolete, version of the message transport service specification [43] contained an optional envelope parameter called encrypted. If used, the field indicates that the message is encrypted as defined in IETF RFC 822 [31]. The syntax for the parameter is two words. The first word indicates the software used to encrypt the body, and the second, optional, word is intended to aid the recipient in selecting the proper decryption key. Current FIPA platforms based on JADE (Java Agent DEvelopment Framework) [6], FIPA-OS [91], etc., do not support this field.

Table 7.2 shows the structure of an ACL message. The performative element is the only mandatory element of an ACL message; all other elements are optional. The ACL message structure does not include any security specific elements.

Table 7.2: ACL message elements

Element	**Description**
performative	The type of communicative act of the message.
sender	Name of the agent who sent the message, mandatory.
receiver	Names of primary recipients of the message, mandatory.
reply-to	Indicates that subsequent messages in this conversation are to be directed to the agent named in this element.
content	Denotes the content of the message.
language	Denotes the language in which the content element is expressed.
encoding	Denotes the specific encoding of the content language expression.
ontology	Denotes the ontology used to give a meaning to the symbols in the content expression.
protocol	Denotes the interaction protocol that the sending agent is employing with this message.
conversation-id	Used to identify the ongoing sequence of communicative acts that together form a conversation.
reply-with	Introduces an expression that will be used by the responding agent to identify this message.
in-reply-to	Denotes an expression that references an earlier action to which this message is a reply.
reply-by	Denotes a time which indicates the latest time by which the sending agent would like to have the reply.

7.3.1 Security evaluation

The current FIPA message specifications do not provide any security services. However, the presided version of the standard [43] had limited provision for security through the envelope primitive encrypted. This allows for additional software to be used to offer message confidentiality through encryption. Origin authentication and message integrity are not supported. Furthermore, the encrypted field in the envelope is intended for the ACC, not the agent itself, which means that the encryption would be under the control of the ACC, not the agent.

7.4 Open PGP

In this section we will briefly describe the Open PGP message structure in order to be able to evaluate its appropriateness for FIPA messages.

Open PGP [22] is a non-proprietary protocol for protecting email using public key cryptography. It is based on PGP as originally developed by Phil Zimmermann [121]. The Open PGP protocol defines standard formats for encrypted messages, signatures, and certificates for exchanging public keys. Over the past decade, PGP, and more recently Open PGP, have become well used de facto standards for encrypted email. By becoming an IETF standard [22], Open PGP may be implemented freely.

PGP messages are constructed from a number of records referred to as packets. A packet is a piece of data that has a tag specifying its meaning. A PGP message consists of a number of packets. Some of those packets may themselves contain other packets. Each packet consists of a packet header, followed by the packet body. The packet header has a tag which denotes what type of packets the body holds. Table 7.3 shows the defined tags.

Table 7.3: Open PGP packet types

Tag no.	**Description**
0	Reserved - a packet tag must not have this value
1	Public-Key Encrypted Session Key Packet
2	Signature Packet
3	Symmetric-Key Encrypted Session Key Packet
4	One-Pass Signature Packet
5	Secret Key Packet
6	Public Key Packet
7	Secret Subkey Packet
8	Compressed Data Packet
9	Symmetrically Encrypted Data Packet
10	Marker Packet
11	Literal Data Packet
12	Trust Packet

As can be seen from Table 7.3, PGP supports the secure message services described in section 7.1, including message confidentiality, through the use of symmetric and asymmetric encryption algorithms, and message integrity and origin authentication, through the use of digital signatures.

Some of the PGP packets listed in the table are designed for key management. This is functionality we have not considered, but which is crucial for deployment of cryptography on a large scale.

PGP is not the only standard for describing cryptographically protected message content. Another example is Cryptographic Message Syntax (CMS) (RFC 3369) [58]. CMS is used by S/MIME (Secure/Multipurpose Internet Mail Extensions) [93] and specifies syntax for representing digitally signed, hashed, authenticated, or encrypted arbitrary message content (CMS is based on PKCS

#7 [97]). Similar functionality to that described for Open PGP is available in CMS.

7.4.1 Using PGP for FIPA messages

We will now consider how the PGP message structures can be used with FIPA messages.

PGP can be used to encrypt and sign the entire ACL message without making any changes to the message envelope. This would leave the agent to take care of the encoding and decoding of PGP structured messages. This would not require any changes to the FIPA specifications, assuming the agent is able to determine if a message is PGP encoded or not (this might, for example, be achieved by using the encoding field).

If there are reasons for treating the elements within the ACL differently, for example to only encrypt or sign certain elements, additional information would need to be provided to the agent to indicate how the message should be processed.

If the ACC service is to perform encryption/decryption and process signatures 'seamlessly', additional information needs to be provided in the message envelope.

The advantage with using an existing standardised protocol such as Open PGP or CMS is that it is already defined, thereby avoiding duplication of work. The drawback might be that the already standardised protocol has features unnecessary for our purposes, possibly adding unnecessary payloads and, perhaps, confusion.

To summarise, it is rather straightforward to make use of the Open PGP message structure for FIPA agent communication. However, in order to allow for all the communication modes described in section 7.2, as well as for cryptographically protected messages, the specifications should allow encryption/decryption and signature processing to be carried out by the ACC service as well as by the agent. This would require additional information to be added in the message envelope. For full flexibility, the ACL message should also carry information describing the security mechanisms applied to the message. The information exchange between an agent and ACC service is further described in the next section.

7.5 Agent — platform interaction

In this section we consider further where the communication security mechanisms fit in the FIPA architecture. We will also highlight architectural issues not covered in the previous sections, namely key management and security policies.

Leaving the complete coding and encoding of encrypted or signed messages to the agent may not always be desirable. To keep agents simple and small, it can be more efficient to let the executing host deal with this potentially complex task. This can be done in different ways, as described in the next two paragraphs.

As shown in figure 7.3, one way is to let the agent receive the message as usual, and when the agent has determined that it is an encrypted or signed message, it forwards the message to the appropriate service to decrypt it or to verify the signature. The agent would then be returned the decrypted message or an indication of the outcome of the signature verification. Similarly when the agent wants to send an encrypted message or a signed message it would send the message to the appropriate service and be returned the processed message, which now can be sent as usual.

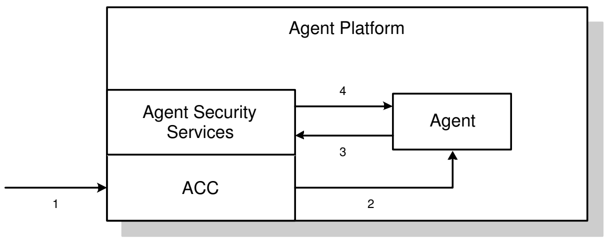

Figure 7.3: Security services separated from ACC

A second option, depicted in figure 7.4, would be to let the ACC intercept the communication and offer the security services in a more transparent way to the agent. This would require the ACC to be able to determine if incoming communication is encrypted or signed, as well as getting an indication from the agent whether encryption or signing should be done before sending the message.

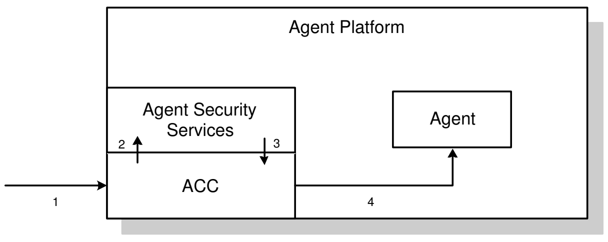

Figure 7.4: Secure communication services offered ‘transparently’ to agents

Both modes of operation should be provided to ensure that the communication modes described in section 7.2 are fully supported.

There are two important, and non-trivial, issues we have not yet covered. One is key management. When encryption is used the encryption/decryption keys need to be managed. For asymmetric cryptography a PKI is typically used to facilitate certain aspects of the key management. In a multi-agent system various key management issues arise due to the fact that the agent executes on a host which, in theory, has full control over the agent, even more so if the agent depends on the host to carry out processing involving cryptographic keys. The following questions need to be considered:

- Is the agent in control of its own key, or is this completely/partly delegated to the platform where the agent is executing?
- Does the agent need its own keys, or can agents on the same host use the same cryptographic keys?

Since applications will have different requirements we believe that agents should have the freedom to decide on these issues themselves. This requires the architecture to support both the above approaches.

The second important thing we have not yet discussed is that of security policies. If an agent depends on the host to perform cryptographic tasks, the agent needs to communicate its requirements to the host. Likewise the host needs to let the agent know certain information about its processing.

While PGP might be sufficient for the transport of encrypted and signed messages, further specifications need to be developed for a complete solution. Communications between the agent and the executing host need to be standardised to cover the transfer of key management and security policy data. In the remainder of this section we consider the information that needs to be exchanged between an agent and the ACC service, which we assume is residing on the platform where the agent is executing.

For outgoing communication (communication originating from the agent) the agent needs to be able to supply (explicitly or implicitly) the platform with the following instructions relating to the processing of the message being sent:

- Request message encryption.
- Supply an encryption key or indicate that the platform's encryption key should be used.
- Specify the choice of encryption algorithm (and other possible algorithm related options, including key length, padding, etc.).
- Request message signing.
- Supply a signature key or indicate that the platform's signature key should be used.
- Specify the choice of signature algorithm (and possible algorithm related options, including key length, padding, etc.).

If an agent requests encryption or signing, the platform would typically apply default values for parameters not specified by the agent.

For incoming communication (communication destined for the agent) the platform needs to be able to inform the agent regarding the following aspects of the security processing of a received message:

- If the message was protected through encryption during transit.

- If the message was encrypted, indicate the method of encryption that was applied to the message (e.g. encryption algorithm and key length).
- If the message carried a digital signature.
- If the message was signed, to what extent the signature and the public key (i.e. the certificate chain) have been verified.

The platform might need certain information from the agent in order to decrypt a message or verify a signature, including the following:

- the decryption key,
- the identities of the trusted Certification Authorities (CAs) and agents,
- the signature verification requirements (e.g. verification against revocation lists and maximum permitted length of certification chains).

The above information can be exchanged between the agent and platform in various ways. The most straightforward way appears to be to let the agent supply message specific information with every message that is passed to the ACC service. Another option, which may be more efficient, is for the agent to have a complete security policy description that can be passed to the ACC service prior to any other communication taking place. Such a policy should be allowed to be as complex as might be required. Different requirements might, for example, apply depending on the destination of a message. A third option is to combine the other two options. An agent can then carry, and supply to the platform, a complete communication security policy, but can also request a particular message treated differently by supplying specific information with the message.

7.6 Conclusions

The FIPA agent communication specifications are lacking sufficient functionality to provide secure communication. By using an existing message structure such as the Open PGP message format, sufficient protection can be achieved for the communication. We have considered where security services can be applied to agent communication within the FIPA architecture, and we have described the information exchange required between an agent and the ACC security services. Further detailed analysis and specification is, however, required for a complete solution.

Another way to achieve secure agent communication appears to be the XML[3] (Extensible Markup Language) specifications. A Document Type Definition (DTD) to carry an ACL message has been defined by FIPA [42]. Various efforts are in progress for specifying how cryptographic services can be applied to XML [36, 60, 118]. These are also likely to provide the security services required for agent message communication.

[3]See `http://www.w3c.org/XML` for information about XML and the ongoing work on XML security.

Chapter 8

A pragmatic alternative to undetachable signatures

Contents

In this chapter a 'pragmatic' alternative to undetachable signatures is proposed, relying on the use of conventional signatures and public key certificates.

8.1 Introduction

As we established in section 2.3.5 there are limits to the protection that can be offered to a mobile agent. An agent platform can potentially modify the agent code, and/or interfere with the data stored by an agent. Hence efforts to protect agents reduce to either finding ways to enhance the level of trust that can be placed in results produced by an agent, or limiting the powers given to an agent. This chapter focuses on the latter approach — in particular it considers the issue of giving an agent the power to sign on a user's behalf, without running the risk of revealing the user's private signature key to the platform on which the agent is executed. Some of the work described in this chapter has been previously published in [17]. Observe that this chapter can be seen as providing a specific solution to issues raised under the 'non-repudiation' heading in section 5.3.2.

As described in section 3.3.10, undetachable signatures can be used to limit the information over which a valid signature can be produced by encoding constraints into a function f. Whilst the original scheme proposed in [98] has proven insecure, an alternative RSA-based scheme proposed by Kotzanikolaou, Burmester and Chrissikopoulos [81], appears to be sound. However, it is a new scheme, and should perhaps be used with care. In the remainder of this chapter we describe an alternative approach which uses only well-established cryptographic techniques. The scheme is described in section 8.2. In section 8.3 we observe the relationship to delegation schemes, before giving the conclusions of this chapter in section 8.4.

8.2 Solving the problem the conventional way

We now consider an alternative solution to the problem which undetachable signatures have been introduced to solve. This solution is wholly based on conventional cryptographic primitives, and hence may be more likely to succeed in practice. It is also quite general in its specification, allowing the use of any digital signature scheme.

8.2.1 Preliminaries

Suppose user U wishes to create a mobile agent A that may run on one or more agent platforms not completely trusted by U. Suppose also that user U wishes to give A the power to sign statements on behalf of U, as long as the statement conforms to rules specified in a string R (where R is in a form agreed by all parties to the transaction). It is possible that R may be completely explicit about the rules governing the signing process by the agent, or R may simply contain one or more pointers to generally agreed policy statements, perhaps with additional parameters.

Note that it is implicit to the solution described immediately below, and also to the solutions using undetachable signatures, that the agent is transferred by U to the platform on which it is to execute by some secure means. This secure transfer should enable the receiving agent platform to check its integrity and origin, and also should protect the confidentiality of all the sensitive parts of the agent (most crucially including any embedded secret or private keys).

8.2.2 Preparing the agent

Before sending the agent A, user U performs the following steps. Note that we assume that U has a signature key pair of its own, (S_U, P_U) say, and a certificate Cert_U for its own public key, P_U, signed by a Certification Authority (CA).

1. U generates a signature key pair (S_A, P_A) specifically for use by the agent.
2. U creates a public key certificate Cert_A for the agent's public key (P_A), signed using U's own signature key (S_U). This certificate also contains a copy of the string R, which states in what circumstances A's key may be used, and which makes it clear that P_A is an agent key. It is also expected that this certificate would have a very short lifetime, i.e. it would have an expiry date very close to the time of issue.
3. U now equips the agent A with the private signature key S_A, and copies of the two certificates Cert_A and Cert_U.
4. A is now securely transferred to one or more agent platforms.

8.2.3 Executing the agent

When A executes, it may be necessary for A to sign a message of some kind (e.g. a commitment to a transaction) on behalf of user U. Such a transaction should be signed using the agent's private key S_A, and the signature should then be transferred with the two certificates Cert_A and Cert_U to the entity requiring the commitment, e.g. a merchant. The recipient of the commitment, M say, then performs the following steps.

1. The user's certificate Cert_U is verified by M using a trusted copy of the CA's public key. Note that if M does not have this CA's public key, then it will need to be derived by some means, e.g. using a certificate chain. M may also want to contact a revocation service to ensure that the certificate has not been revoked.

2. M checks that it is prepared to accept a commitment from U, and also checks that the name in the certificate is consistent with the user name received from the agent.

3. The agent's certificate Cert_A is verified by M using the copy of U's public key obtained in the first step.

4. M checks that the string R contained in Cert_A is consistent with the transaction that is taking place.

5. M finally checks the agent's signature using the copy of A's public key obtained from Cert_A.

It should be clear that, by the simple step of including R in the certificate for A's key pair, the power given to A by U can be limited to that specified by U. This has been achieved without any need for a new cryptosystem.

8.2.4 Remarks on implementation

Before attempting to compare this new scheme with the use of undetachable signatures we make some remarks about the implementation of the scheme.

- Given that the agent key pair has only a short lifetime, it may be possible to use a relatively short key. That is, if the signature scheme is RSA based, a short modulus could be used, say of 512 bits, in the knowledge that factoring the modulus and hence breaking the key would be infeasible during the key's lifetime. This would make key generation faster and would reduce the amount of key information to be transferred. It would also mean that creating and verifying agent signatures could be made significantly more efficient.

- If a 'weak' key pair was used for the agent key, or, more generally, if the certificate for the agent key pair has a very short period of validity, problems might arise if the agent's signature is required to have long term validity, e.g. to provide a non-repudiation service in the event of a dispute. The 'standard' way of resolving this problem is to use a timestamping service to sign a concatenation of the signature and a timestamp, providing evidence that the signature was generated during the key's period of validity. An alternative to using a trusted timestamping service would be to simply require the agent platform to add a timestamp and its signature to any signed commitments output by the agent. Not only would this provide evidence about when the agent signed the message, but it would also enable the recipient of the signed message to verify on which platform the agent was running. This would appear to be a valuable service in its own right, which would apply equally to the case where an undetachable signature scheme is employed.

8.2.5 A brief comparison

We now attempt to briefly compare the efficiency of the above scheme with the efficiency achievable using an undetachable signature scheme. For the purposes of the comparison we suppose that signatures for the scheme in this chapter are computed using RSA and a hash function, and we compare this with the RSA-based undetachable signature scheme of Kotzanikolaou et al. [81]. To compare the efficiencies of the two schemes we compare separately the work to be performed by the user U, the agent A, and the recipient of the commitment, M.

- *User U.* For the scheme above, the user will be required to generate a key pair and certify the public key, i.e. compute one signature and generate one key pair. For the undetachable signature scheme of [81], the user is required to perform two exponentiations, equivalent in complexity to performing two signatures. Note that, whilst key generation will typically take much longer than computing a signature, key pairs could not only be made quite small (as discussed above), but could be generated in advance. Hence, the new scheme, whilst requiring more computation overall, actually requires less computation at the time of agent creation.
- *Agent A.* For the new scheme, the agent is required to compute one signature. For the undetachable signature scheme, the agent is required to perform two exponentiations, equivalent to two signatures. Moreover, for the undetachable signature scheme these will be 'full size' signatures, whereas for the new scheme the key lengths may be reduced (as above).
- *Recipient of commitment M.* For the new scheme the recipient of the signed message will be required to verify two certificates and a signature, i.e. a total of three signature verifications. For the undetachable signatures scheme it is also necessary to verify the user's certificate, as well as performing two exponentiations. Hence the two schemes have roughly comparable efficiencies.

It would appear that the scheme of this chapter has potential efficiency advantages over the undetachable signature scheme, quite apart from the advantages inherent in using established cryptographic primitives.

8.3 Relationship to secure delegation schemes

Note that the problem which the proposed solution is designed to address appears to be closely related to the problem of secure delegation in distributed systems. Delegation refers to the situation where one entity wishes a separate entity to perform a task on its behalf. Security problems arise when the delegated entity does not have the access rights to perform the task, and hence must be temporarily given these rights in order to perform the requested actions. The issue then becomes one of giving these rights in such a way that they cannot be abused. See, for example, [30] for a general introduction to delegation issues.

An analogous approach to the one described here has been proposed by several authors as a solution to secure delegation — see, for example, [109]. However, instead of the use of a public-key certificate, special 'delegation tokens' have been proposed. Note also that issues can arise with any such solution since the originating user will have a copy of the private key generated for agent use. The user may use this key to masquerade as the agent, and then deny the transaction, blaming the platform on which the agent has run. The proposed use of countersignatures by the agent platform, as described in section 8.2.4 above, significantly reduces the seriousness of this threat.

8.4 Conclusions

A pragmatic solution to a mobile agent security problem has been proposed. This solution has potential practical advantages by comparison with the use of undetachable signatures, and appears to offer a very similar set of security guarantees. When combined with the use of signatures by the agent platform, this solution has the potential to solve certain problems relating to transaction repudiation.

Chapter 9

Mobile agent based transactions

Contents

This chapter proposes two methods to improve the security and reliability of mobile agent based transactions in an environment which may contain some malicious hosts.

9.1 Introduction

In this chapter we consider strategies for the deployment of mobile trading agents to reduce certain security threats to their operation. In a future world of co-operating mobile and fixed devices, the mobile agent computing model is expected to become an increasingly important one. In the domain of e-commerce/m-commerce transactions, mobile trading agents could play a very useful role. Users could launch such agents to make transactions on their behalf, and the agents would look for the 'best buy' by visiting multiple merchant sites without any direct user intervention. Indeed such activity could take place while the user has no current network connectivity.

The mobile agent computing model gives rise to a range of security threats. As discussed in section 2.3.5, these threats can be divided into two main classes:

- threats to the platform from malicious and/or unauthorised agents, including threats to the integrity of the platform and other agents, threats to the confidentiality of stored data, and denial of service threats, and
- threats to the agent from malicious platforms, including threats to the confidentiality of agent stored data, and threats to the integrity of the agent and its computations.

In this chapter we are concerned with the second class of threats, and in particular with threats to agents deployed for trading applications. Specifically, users will need to give trading agents certain authority to authorise transactions, whilst at the same time users will wish to protect themselves against malicious merchants forcing an agent to make a non-optimal purchase.

We consider simple ways in which deployment of multiple agents can reduce the threat to trading agents from platforms outside of their direct control. We consider two general approaches. In the first approach multiple agents are equipped with 'shares' of the means to commit to a transaction. In the second approach a single trusted host provides a location for multiple agents to 'report back' information enabling a purchasing decision to be made. Some of the work described in this chapter has been previously published in [14].

The chapter has the following structure. Section 9.2 explores threats to trading agents in more detail. This is followed in sections 9.3 and 9.4 by a discussion of the models used here for agent platforms and for trading agents. Sections 9.5 and 9.6 then explore the two approaches for enhancing trading agent security. Section 9.7 gives the conclusions of this chapter.

9.2 Threats to trading agents

The general threats to mobile agents have been described in section 2.3.5. In this section we consider the particular threats to a trading agent in more detail.

That is, we consider the particular threats to an agent which wishes to purchase an item (or a service) from a merchant.

1. A malicious host lies about its offer.

   Here a host lies about the offer it makes to an agent, in order to get the trade. The host would then charge a higher price at a later date. One way around this is to force the host to sign its bid, thereby committing to it.

2. A malicious host learns other offers and undercuts them.

If a host knows that all offers but its own have been collected and finds out the best standing offer, it can undercut the best standing offer slightly (in fact the host need not know all other offers, it could just undercut the current offers). Of course, in some circumstances letting hosts undercut each other might be considered a desirable feature.

3. A malicious host learns the maximum price a user is prepared to pay and bids just under this.

   In a similar fashion the host may charge more than its normal price, if it knows the maximum price the user is prepared to pay. Thus a host must be kept from learning the maximum price a user is prepared to pay, either by encrypting this information or by not sending this information with the agent.

4. A malicious host manipulates the requirements.

   This is when the host changes the requirements to favour its bid. For example, it could add a requirement to buy from a certain host, or remove constraints from the agent.

5. A malicious host alters the agent's route.

   Here, the host keeps the agent away from its competitors, and thus secures the agent's trade. One way to prevent this is to use more than one agent (possibly an agent per host), send each agent on a different route and combine the offers on the agent's return. Another way is to use one agent with a 'star' like route – it returns home after visiting each host before being sent out to a different host.

6. A malicious host commits to purchases that the user does not wish to make.

This happens when a host can abuse the committal function that an agent has. A method to discourage this is to force the host to sign a transaction, as well as the user (thus providing traceability).

7. A malicious host denies the agent a service.

   Here a host would stop an agent from moving further on its route. This of course could be traced if an agent reports back when it arrives at a host.

8. A malicious host captures electronic money.

   Here a host would remove the electronic money that an agent may have to purchase an item and either steals the money outright, or uses it for a different purchase.

We do not consider the payment process here, as we are concerned only with the part of a transaction involved in selecting a merchant and committing to the transaction.

9.3 Models of agent platforms

Mobile agents roam between platforms. However, they can also communicate with each other, and with other hosts. This leads to the question as to the best "platform model" to use for trading (or indeed any other) agents. There are clearly two basic approaches which we now describe.

The first approach (see Figure 9.1) is to have a designated platform (or a collection of such platforms) to which we can send an agent to execute. This agent then communicates with merchant hosts to seek information and commit to purchases.

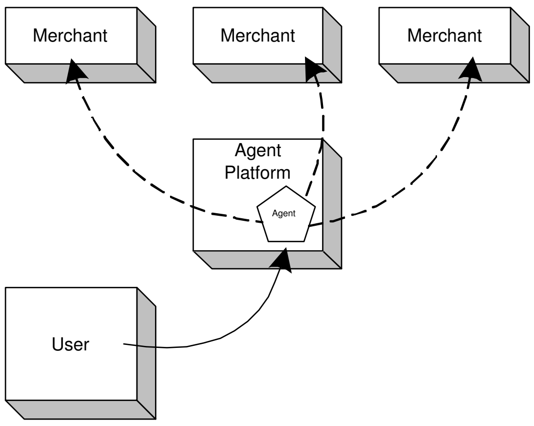

Figure 9.1: A model for agent platforms

The second model (see Figure 9.2) is to have an agent roam to each merchant host in turn and collect the information it requires. After collecting all the information the agent can then either return to the user to make the purchase, return to the chosen merchant to make the purchase or make the purchase from the final host.

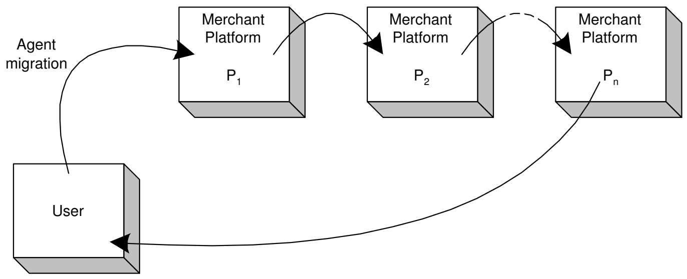

Figure 9.2: A second model for agent platforms

In a mobile telecommunications environment it may also be beneficial to have a third model. This is where the requirements for a purchase are communicated to a ‘home platform’ (the user’s home PC or a network operator controlled device) which then forms the agent and conforms to one of the above models.

In the above, any of the platforms may be malicious, with the possible exception of the home platform. The solutions proposed below can be made to fit into any of the above situations, although they both fit better into the first model.

The security risks associated with the above two models clearly differ. In the first case, the 'designated platform' might be trusted to keep secret certain agent information. An example of where this might be useful is when the agent contains details of the user 'expected' price (or maximum price), which it would be helpful not to reveal to the merchant. Of course, the threat then arises that one of the designated platforms will collude with one or more of the merchants. In the second case, it is clearly impossible to try and keep any information in the agent secret from the merchants. In both cases, however, as we will show in the remainder of this chapter, there are potential benefits to be gained from the use of multiple agents, albeit not from the confidentiality perspective.

9.4 Model for a trading agent

We consider the information that an agent wishing to trade must know. Firstly, when initiating a purchase, a user will have a set of requirements (for instance the item to be purchased, the maximum price for that item, a time limit within which the purchase to be made). We will assume that a user encodes these requirements into a string R which is understood by all parties. When a host quotes for a given purchase, it will also produce a similar string with its offer.

The agent, if it is to perform the purchase on behalf of the user, must also carry a function which will commit to the trade. This could be performed by, for instance, signing the details of the trade. One scheme to allow an agent to perform a signature operation on behalf of a user without revealing the user's private key to a host is the undetachable signature scheme proposed in [81] (see also section 11.2). In this scheme, using RSA, an agent carries both the hash value, h, of the requirements and the signed hash value, $h^d \bmod n$, where (d, n) is the user's private RSA key. To commit to a transaction for the user the agent calculates

$$(h^d)^x = h^{xd} = (h^x)^d \bmod n$$

effectively signing h^x where x is the host's offer. An alternative to this, where the agent carries its own private key which the user certifies, was given in chapter 8.

Thus we assume that a trading agent will carry the following information:

- User Identifier – U
- Requirements for purchase – R
- A committal function – $\mathcal{C}$. The committal function is used by the agent to commit to a transaction on the user's behalf. $\mathcal{C}$ could be a signature function using a special private key provided to the agent by the user (as in chapter 8), or, $\mathcal{C}$ could be a function of the type described above, derived from the user's own private signature key. In any event, we assume that the function is designed so that only transactions within constraints defined by the user can be authorised.

Note that, if a single 'trading agent' is deployed there are a number of problems which might arise. Firstly, although the committal function will typically be limited to transactions conforming to user-defined parameters, there is still the possibility that the agent platform will force the agent to commit to a transaction which is less than optimal. It may also commit to more than one transaction, even if the user only intended to make at most one purchase.

One way to reduce this threat is to deploy multiple agents, a subset of which must agree to the transaction before it can be authorised. Such an approach is the focus of the remainder of this chapter.

9.5 Threshold scheme

One means of addressing the malicious host problem is to use multiple agents each of which has a 'vote'. If one of the possible transactions receives enough votes, then a transaction will be authorised with the relevant merchant. We begin by outlining the scheme, and then consider the details of what a secure vote can consist. We assume use of a (k, n) scheme – *i.e.* a merchant will need k votes out of a possible n to 'win'.

9.5.1 The scheme

Let $P = \{P_1, P_2, \ldots, P_n\}$ be a set of agent platforms. The user then sets up a (k, n) voting scheme with shares $v_1, v_2, \ldots, v_n$. Clearly k should exceed the number of 'suspected' malicious hosts. Given no information about the system a sensible value would probably be $n/2 + 1$. The value of k reflects the level of trust in the system.

The user then forms n agents A_i $(1 \leq i \leq n)$ containing the following information

- User Identifier – U
- Requirements for purchase – R

- A vote – v_i

Each agent is then dispatched to its agent platform. At the platform there are two modes of execution:

1. The agent contacts each merchant itself, and gathers bids that meet the requirements.
2. The agent contacts a subset of the merchants and communicates the best bid to its peers.

We note that for case 2, if an agent is contacting a number of merchants, and these merchants are only contacted by one agent, a successful attack can be achieved by compromising less than k hosts.

When each agent has received all the information about each bid, the agent sends its vote to the merchant with the best offer. On receipt of the correct number of votes, the merchant or a nominated third party can construct (and verify) the authorisation for the bid from the votes. The merchant or nominated third party can then use this as evidence that the user has committed to transaction.

We now consider the security of the above scheme. The major advantage of the scheme is the need to corrupt either $n - k + 1$ agents to prevent the transaction or k hosts to divert or alter the transaction. Thus the choice of k is crucial.

This also means that a denial of service attack is harder as a host or set of colluding hosts will need to terminate (or prevent from communicating their vote) $n - k + 1$ agents. Again to force a purchase, a host or hosts must force k agents to offer their vote.

If an agent visits a subset of the hosts involved, the information could then be used to help identify any malicious hosts.

9.5.2 The votes

As mentioned above, the votes can be assembled by either the selected merchant or a nominated third party. Note that there are clear risks associated with giving votes to the merchant, since the merchant could now possibly commit the user to a transaction of the merchant's choice (within any constraints imposed by the string R). That is, the merchant is not forced to commit to the transaction as offered to the agents. Hence the use of a nominated third party to reconstruct the votes is the preferred approach. The possibility that this may not be feasible in practice leads to an alternative approach.

One approach is *threshold cryptography* as discussed in section 6.3.2. Recently Shoup [103] 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 (as it relies on a trusted party to form the shares) this scheme is ideal in our setting. (Note that an alternative scheme without a trusted dealer is given in [32]. This scheme also improves on Shoup's scheme by not relying on an RSA modulus made up of 'safe primes'). An example of an ElGamal based scheme is given in [82]. We note that an (n, n) threshold signature scheme is just a multisignature; such schemes have been studied for many years — see, for example, page 488 of [87].

We note, however, that such a threshold signature scheme does not provide a means for the shares to incorporate an encoding of the string R. Thus, if there were k colluding hosts they could sign (and reconstruct a signature) for any document. One solution to this problem, analogous to the solution described in chapter 8, is for the user to generate a special signature key pair for the particular purchase (i.e. for this particular set of agents), and then to generate a certificate for the public key incorporating a copy of R. When the signature is reconstructed from the signature shares, it can be verified using this certificate. An alternative approach is to merge the undetachable signature scheme given in [81] with the threshold signature scheme of Shoup [103], and details of this are given in chapter 11.

9.6 Using one trusted host

We consider a second solution to the problem, which employs a single trusted host. We note that the solution described below involves a user sending out agent(s) to individual merchant platforms, whereas it could just communicate with them to ask for their bids. However, in a wireless communications setting where communication is expensive, slow, and/or unreliable, it is believed to be beneficial to be able to dispatch an agent into the fixed network. When the agent has finished its task it contacts the user or waits for the user to collect the result.

Let $P = \{P_1, P_2, \ldots, P_n\}$ be a collection of platforms offering a service that a user wishes to purchase. Let T be a host that the user trusts to act honestly in this transaction. (Note that we do not need to trust this host fully – it just needs to be neutral in this transaction). Before the transaction commences we assume that each platform P_i securely establishes a shared secret key K_i with T. Optionally, a key for message integrity checks could also be established.

The user dispatches an agent A to the trusted host T containing the information outlined in section 9.4. We note that the committal function $\mathcal{C}$ may be of any form with which the user is prepared to trust the host T. However, to reduce the trust requirements we envisage that this will be an undetachable signature scheme, e.g. the one described by Kotzanikolaou [81], or the scheme given in chapter 8.

There are now several approaches for T. The first is to form a single subagent containing the following information:

- agent identifier – I,
- requirements for purchase – R,
- host identifier – T,

which would then visit each of the platforms in P in turn. We note that the requirements sent out do not need to include pricing information (that is the maximum price the user is prepared to pay) or any other information that the user wishes to be used to help make the decision, but does not wish to communicate to the platform. Another approach is to form a single agent for each platform. A third approach has the above agent visiting a subset $P' \subset P$ of the above platforms. Whichever strategy is employed, at each host the agent performs the following actions:

1. Find out the platform's bid B_i for the item specified in the requirements

R.

2. Encrypts the concatenation of B_i, R, P_i and I using the key K_i. At this point the host could also, optionally, attach a MAC (Message Authentication Code) to protect the integrity of the host's bid. Label the encrypted string E_i.
3. The agent then stores the pair (P_i, E_i).

The agent returns to T when it has finished visiting all of its platforms. The agent on T then decides the best offer and commits to it using the committal function.

We note some of the features of the above scheme.

- Using an agent per host alleviates the need to encrypt anything, assuming that agents are always transferred between hosts in encrypted form.
- Using a single agent leaves the scheme open to some attacks.
- Using more than one agent that does not visit all the hosts could be used to (help) identify a malicious host.

If we use a single agent and it visits all the hosts, or we have an agent that visits more than one host, the agent is subject to the following attacks:

- An approach to enable a malicious host to underbid its competitors is as follows. The host forms a new agent containing the user's requirements, a fictitious user identifier, and its own host identifier. This agent would then traverse the route of the user's agent, and discover the bids offered for that set of requirements. The host could then underbid its competitors, but the user's agent would need to have been kept on the malicious host in the interim period. Thus monitoring the progress of an agent could help determine if such an attack was being used.

- A simple denial of service attack: stop the agent in its tracks. This attack is hard to defeat if there is no progress monitoring of the agent's movement.
- A malicious host could alter the pair (P_i, E_i) to read $(P_j, junk)$ (where $junk$ is a random string of the correct length) to stop the decryption of a bid. However as the host cannot read the bid, for this to be successful (*i.e.* to delete those bids more attractive than those of the malicious host) the host would have to have knowledge of all the bids, which it would have to gather itself (possibly by cloning the agent).

If multiple agents are deployed, each agent visiting a subset of the hosts, and each host is visited by several agents. It is possible to identify a misbehaving host, assuming that *enough* agents are used. This also requires careful choice of agent destinations and routing.

Note that to force T to purchase from a malicious host, the host has to lie and then be unscrupulous, or just lie and possibly not profit as much as it would expect. That is if the malicious host M wants to force a user to trade with it, then it must have the best price. So it must either charge more than its advertised price (possibly breaking the committal function) or make less profit than it expects (because the price advertised is less than the host should sell for).

We now consider the extent to which the user must trust the host T. The user must trust that T does not favour a particular platform for this transaction. However, with a sufficiently good committal function then this is the only trust requirement. For example using the Kotzanikolaou et al. undetachable signature scheme [81], as a committal function, T can be given the means to commit to the transaction without being trusted with a copy of the user's private signature key. This may be a situation where using an undetachable signature scheme has advantages over the creation of a separate signature key for each agent.

9.7 Conclusions

We have considered two different ways in which the deployment of multiple agents can reduce the threat to mobile trading agents from potentially malicious agent platforms. In the first approach multiple agents are equipped with 'shares' of the means to commit to a transaction. A method implementing this idea using a threshold signature scheme, e.g. the recently proposed scheme of Shoup, [103], was outlined. In the second approach a single trusted host is employed to collect information from multiple agents on possible transactions. This host then chooses the optimal transaction and commits to it.

The two approaches each have their own advantages. The first approach avoids the need for a single trusted host. However, implementing the first approach requires use of some potentially complex cryptographic signature functions. The second approach is potentially less complex from a cryptographic perspective, but does require a host which, if not completely trusted, is at least required to act neutrally with respect to the set of merchants. Both approaches are of potential practical importance in future mobile computing environments.

Chapter 10

A pragmatic alternative to threshold signatures

Contents

This chapter presents some rather simple alternatives to threshold signatures which raise questions about the value of such schemes, at least when applied to the mobile agent scenario.

10.1 Introduction

As discussed in section 6.3.2, threshold signature schemes enable a group of n entities to be given 'shares' of a private signature key in such a way that, for some parameter k ($1 \leq k \leq n$), any subset of k entities can collectively create a valid signature on a message, whereas any collection of $k-1$ or fewer entities cannot. Schemes of this type have been discussed widely in the literature, and a number of systems have been proposed; see for example [103] for a brief survey.

Again, as discussed in section 6.3.2, one particularly attractive scheme has been recently proposed by Shoup, [103]. This scheme is based on RSA, and the composite signature can be verified in exactly the same way as a 'regular' RSA signature. In the discussion below we use this scheme as a 'benchmark' against which alternatives can be compared. Some of the work described in this chapter has been previously published in [16].

Section 10.2 motivates the use of threshold signatures for mobile agents. Section 10.3 describes the alternative approach, relying on conventional cryptography. Section 10.4 gives the conclusions of this chapter.

10.2 Mobile agents and threshold signatures

As pointed out in section 2.3.5, there are limits to the protection that can be offered to an agent. An agent platform can potentially modify the agent code, and/or interfere with the data stored by an agent. Hence efforts to protect agents reduce to either finding ways to enhance the level of trust that can be placed in results produced by an agent, or limiting the powers given to an agent.

This chapter focuses on the former approach — in particular it considers the issue of dividing a task amongst multiple agents to increase the level of trust in the collective results of the agents.

In particular we are concerned with giving all subsets of agents of size k or more the power to sign on a user's behalf, without giving smaller groups of agents such a capability. We suppose that the agents are set up to agree to a transaction, the details of which the agents are to determine. As discussed in section 9.5 this might typically occur by giving the agents the power to visit a number of merchants, and find out which merchant is offering a particular good or service at the lowest price. Some of the agents may be modified by a malicious host, but we assume that this occurs to at most $k-1$ of them.

As discussed in more detail in chapter 9, this seems to be a natural application for the notion of threshold signatures. The user launching the agents acts as the 'dealer' in the threshold signature scheme, and creates the shares of the signature key. Each agent is equipped with a share, and when a transaction is to be signed creates a signature share. A third party (e.g. the merchant making the transaction) receives the signature shares and uses them to construct a valid signature. This distribution of trust across a multiplicity of agents reduces the threat from small numbers of malicious hosts, who might be capable of manipulating agent computations. Finally, note that this problem also relates to the well-known concept of a *multisignature* which, for our purpose as least, can be regarded as a special type of threshold signature for the case of $k = n$.

10.3 An alternative based on conventional signatures

We now consider an alternative solution to this trust distribution problem. Unlike the use of threshold signatures, this solution is wholly based on conventional cryptographic primitives, and hence may be more likely to succeed in practice. It is also quite general in its specification, allowing the use of any digital signature scheme.

Note that it is implicit to the solution described immediately below, and also to the solutions using threshold signatures, that the agents are transferred by the user U to the hosts on which they are to execute by some secure means. This secure transfer should enable the receiving host to check its integrity and origin, and also should protect the confidentiality of all the sensitive parts of the agent (most crucially including any embedded secret or private keys).

10.3.1 Preparing the agents

Before sending the agents, (which we label $A_1, A_2, \ldots, A_n$) user U performs the following steps. Note that we assume that U has a signature key pair of its own, (S_U, P_U) say, and a certificate Cert_U for its own public key, P_U, signed by a Certification Authority (CA).

1. U generates a signature key pair (S_i, P_i) specifically for use by agent A_i.
2. U creates a public key certificate Cert_i for agent A_i's public key (P_i), signed using U's own signature key (S_U). This certificate also contains policy information which indicates precisely the purpose of the certificate,

and also the parameter k, indicating the 'threshold value' (as above). It is also likely that this certificate will have a very short lifetime, i.e. it would have an expiry date very close to the time of issue.

3. U now equips agent A_i with the private signature key S_i, and copies of the two certificates Cert_i and Cert_U.
4. A_i is now securely transferred to one or more agent platforms.

10.3.2 Executing an agent

Now suppose that some subset of k agents decide that they wish to collectively sign a message (e.g. to commit to a transaction with a merchant) on behalf of user U. Each agent A_i signs the message using its own private key S_i, and the signature is then transferred with the two certificates Cert_i and Cert_U to the entity requiring the signature, e.g. a merchant. The recipient of the agent signatures, M say, then performs the following steps for each received agent signature.

1. The user's certificate Cert_i is verified by M using a trusted copy of the CA's public key. If M does not have this CA's public key, then it will need to be derived by some means, e.g. using a certificate chain. The agent then checks that it is prepared to accept a signature from U, and also checks that the name in the certificate is consistent with the user name received from the agent.

   Note that this step will only need to be performed once for the k agent signatures.
2. The agent's certificate Cert_i is verified by M using the copy of U's public

key obtained in the first step.

3. M finally checks the agent's signature using the copy of A_i's public key obtained from Cert_i.

The merchant waits until k valid agent signatures have been received (recall that k was encoded in each agent certificate). The collection of k agent signatures is then deemed to be equivalent to the signature of the user, and the merchant proceeds with the transaction. The collection of k agent signatures (with the accompanying certificates) are retained as evidence of the transaction.

10.3.3 Remarks on implementation

Before attempting to compare this new scheme with the use of threshold signatures we make some remarks about the implementation of the scheme.

- Given that the agent key pairs have only a short lifetime, it may be possible to use relatively short keys. That is, if the signature scheme is RSA based, a short modulus could be used, say of 512 bits, in the knowledge that factoring the modulus and hence breaking the key would be infeasible during the key's lifetime. This would make key generation faster and would reduce the amount of key information to be transferred. It would also mean that creating and verifying agent signatures could be made significantly more efficient.
- If a 'weak' key pair was used for an agent key, or, more generally, if the certificate for an agent key pair has a very short period of validity, problems might arise if the agent's signature is required to have long term validity,

e.g. to provide a non-repudiation service in the event of a dispute. The 'standard' way of resolving this problem is to use a timestamping service to sign a concatenation of the signature and a timestamp, providing evidence that the signature was generated during the key's period of validity. An alternative to using a trusted timestamping service would be to simply require the agent host to add a timestamp and its signature to any signatures output by the agent. Not only would this provide evidence about when the agent signed the message, but it would also enable the merchant receiving of the signature to verify on which host the agent was running. This would appear to be a valuable service in its own right.

- The scheme as described does not restrict the nature (e.g. value and/or type) of messages which the agents can collectively sign. However, a restriction could easily be imposed by including the scope of the agent keys in the respective agent public key certificates, as described in chapter 8.
- This scheme is in some respects analogous to the widely discussed notion of delegation using special delegation keys – see for example [30] for an introduction to delegation issues and [109] for one approach to the use of delegation keys.

10.3.4 A brief comparison

We now attempt to briefly compare the efficiency of the above scheme with the efficiency achievable using the Shoup threshold signature scheme [103]. For the purposes of the comparison we suppose that signatures for the scheme in this chapter are computed using RSA, and we compare this with the Shoup RSA-based threshold signature scheme [103]. To compare the efficiencies of the two schemes we compare separately the work to be performed by the user U, each agent A_i, and the recipient of the agent signatures M.

- *User U.* For the scheme above, the user will be required to generate one key pair for each agent A_i, and certify the public keys, i.e. compute n signatures and generate n key pairs. For the threshold signature scheme of [103], the user is only required to generate at most one key pair (for the secret key that is shared by the n agents). However, each agent will need to be equipped with a public key certificate for its share, so that the signature share can be verified by the merchant (or whoever combines the signature shares). Hence the user will be required to generate one key pair and compute n signatures. Note that, whilst key generation will typically take much longer than computing a signature, key pairs for the scheme described in this chapter could not only be made quite small (as discussed above), but could be generated in advance. Hence, the new scheme, whilst requiring more computation overall, actually requires comparable amounts of computation at the time of agent creation.
- *Agent A_i.* For the new scheme, the agent is required to compute one signature. For the Shoup scheme, each agent is required to perform one exponentiation, equivalent to one signature.
- *Recipient of agent signatures M.* For the new scheme the recipient of the signed message will be required to verify $k+1$ certificates (the user certificate and the k agent certificates) and k signatures, i.e. a total of $2k+1$ signature verifications. For the Shoup scheme it is also necessary to verify the user's certificate, as well the k agent certificates and the k signature shares. Hence the two schemes have roughly comparable computational efficiencies. Of course, the storage efficiency for the scheme described above will be rather less than for the Shoup scheme, unless it is necessary

to retain the signature shares for auditing purposes.

In summary it would appear that the scheme of this chapter is really quite comparable with the Shoup threshold signature scheme, with two exceptions. Initialising the scheme requires a number of key generations, which, however, could be done 'off-line'. The other difference is in the size of storage required for signatures, which is significantly larger for the new scheme.

However, this advantage of the Shoup scheme disappears if the signature shares must be kept for auditing purposes. Indeed, this is not such an unlikely scenario, since, in the event of a dispute, it will be valuable to learn which agents took part in the signing process. Thus the implementation efficiency differences, which could be rather minor in practice, could easily be outweighed by the advantages inherent in using established cryptographic primitives.

Finally observe that one other difference between the approach proposed here and the use of threshold signatures relates to the issues of anonymity and accountability. In most threshold signature schemes, the verifier of the signature cannot determine which of the k shareholders created the signature, whereas with the above approach there is no anonymity for agents. In some circumstances the anonymity property may be desirable, but in many agent applications the reverse is likely to be true. That is, the originator of the agents will in many cases wish to have the means to determine which agents performed the action on its behalf.

10.4 Conclusions

A pragmatic alternative to threshold signatures has been proposed. This alternative has potential practical advantages by comparison with the use of threshold signatures, and appears to offer a very similar set of security guarantees. Although the analysis was performed within the context of mobile agents, it is possible that the scheme described here is competitive with the Shoup threshold signature scheme in other environments.

Chapter 11

Undetachable threshold signatures

Contents

This chapter introduces the concept of undetachable threshold signatures, which enables constrained signing power to be distributed across multiple agents, thus reducing the necessary trust in single agent platforms.

11.1 Introduction

A digital signature is the electronic counterpart to a written signature. Thus one way to commit to an electronic transaction is by the use of a digital signature. It would be useful to let mobile agents be able to commit to transactions on a user's behalf. Mobile agents, however, face the problem of having to execute in a hostile environment where the host executing the agent has access to all the data that an agent has stored (for instance the private signature key).

Undetachable signatures, see section 3.3.10, can be used to limit the information over which a valid signature can be produced by encoding constraints into a function f. However, one problem with this approach is that the agent is still given the power to sign any transaction it likes, subject to the requirement that the transaction must be consistent with the constraints used to construct f. Thus, for example, whilst the constraints may limit the nature and/or value of a transaction, a malicious host may force an agent to commit to a transaction much less favourable than could be achieved.

Thus, to protect further against malicious hosts, a user may wish to use more than one agent and have the agents agree on a bid before committing to it. Hence, a user may send out n agents with the criteria that k of them must agree before committing to a purchase. The obvious solution to such a requirement is to employ a *threshold signature scheme*, meaning that agents can all sign the bid they think 'best' given the user's requirements, and then, on receipt of a sufficient number of these bids, the user's signature can be reconstructed.

However, such a scheme does not possess the means to constrain the power given to a quorum of agents. This motivates the introduction of the concept of an *undetachable threshold signature* which both distributes signature authority across multiple agents and simultaneously constrains the signatures that may be constructed. Some of the work described in this chapter has been previously published in [15].

The rest of the chapter is organised as follows. In section 11.2 we outline the undetachable signature scheme of [81], and in section 11.3 we briefly review threshold signatures and describe the method of Shoup [103] to construct such a scheme. In section 11.4 we define the concept of an undetachable threshold signature, and show how an example of such a scheme may be obtained by combining the schemes of [81] and [103]. Section 11.5 gives the conclusions of the chapter.

11.2 RSA undetachable signatures

We briefly present the RSA undetachable signature scheme given in [81]. The user sets up an RSA signature pair in the usual manner, that is the user selects an RSA modulus n which is the product of two primes p and q, and a number e such that $1 \leq e \leq \phi(n) = (p-1)(q-1)$ and $\gcd(e, \phi(n)) = 1$. Let d be such that $1 \leq d \leq \phi(n)$ and $ed = 1 \bmod \phi(n)$. The user then publishes the verification key (n, e) and keeps d as the private signing key.

Let I be an identifier for the user and R the encoded requirements of the user for a purchase (we assume that R is encoded in a manner which is understood by all parties). Let h be an appropriate hash function (*i.e.* one giving a value in ${\mathbb{Z}_n}^1$). The user then forms $H = h(I, R)$.

[1]We note that this property of a hash function is non-trivial to achieve efficiently. It is also non-trivial to obtain a uniform distribution over $\mathbb{Z}_n$ based on existing constructions that

The user then gives an agent the user identifier, the requirements, and the pair (H,G) as its undetachable signature, where $G = H^d \bmod n$. To sign a bid B (which we assume is in the same format as R), the executing host calculates $x = h(B)$. The undetachable signature is then the pair (H^x, G^x). We note that,

$$G^x = (H^d)^x = H^{dx} = H^{xd} = (H^x)^d$$

so that the server has signed the value H^x with the user's private key.

We briefly note that this scheme appears secure, and an informal proof of this fact is given in [81][2]. To forge a signature on a different set of requirements R' a malicious host would need to forge $H' = h(I, R')$, $G' = (H')^d$ and $(G')^x$. Clearly the main obstacle to the attacker is the need to forge G', and this would require knowledge of a user's private key. Having said this, there is nothing in this scheme to prevent a host from signing more than one bid, or presenting a bid that just meets the requirements of the user (as opposed to a possibly better offer).

11.3 Threshold signatures

As previously described in section 6.3.2, by using threshold signatures we can distribute trust amongst a number of agents, rather than relying on the correct execution of a single agent. Recently Shoup [103] proposed an RSA based threshold signature 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

yield mappings onto the set of n-bit strings. A slow construction of such a hash function is given in [5].

[2]The authors of the paper claim that the security of their scheme rests on the difficulty of forging RSA signatures. A short argument is given in the paper to support this claim. It appears that this argument can be developed into a more formal security proof relating the security of their scheme to that of the underlying RSA signature scheme in the random oracle model.

combiner, and must do work that is equivalent, up to a small constant factor, to computing a single RSA signature.

Although in some sense not perfect as a threshold signature scheme (as it relies on a trusted party to form the shares) this scheme is ideal in our setting, where the user dispatching the agent will always (one would hope) trust themselves. (Note that an alternative scheme without a trusted dealer is given in [32]. This scheme also improves on [103] by not relying on an RSA modulus made up of 'safe primes'). An example of an ElGamal based scheme is given in [82].

We next briefly outline the threshold signature scheme of [103].

The user (dealer) forms the following:

- An RSA modulus $n = pq$ where $p = 2p' + 1$ and $q = 2q' + 1$ are safe primes, *i.e.* p', q' are prime.
- A public exponent e, where e is prime, and a private key d, where $de \equiv 1 \pmod{p'q'}$.
- A polynomial $f(x) = \sum_{i=0}^{k-1} a_i x^i$ where $a_0 = d$ and $a_i \in \{0, 1, \ldots, p'q' - 1\}$ (selected at random) for $1 \leq i \leq k$.
- $L(n)$, the bit length of n, and L_1, a secondary security parameter — Shoup [103] suggests $L_1 = 128$.
- The l signature key shares of the scheme s_i, where each s_i is selected at random from the set $\{s | 0 \leq s \leq 2^{L(n)+L_1}, s \equiv f(i) \bmod (p'q')\}$.
- The verification keys $\mathrm{VK} = v$ and $\mathrm{VK}_i = v^{s_i}$ where $v \in Q_n$, the subgroup of squares of $\mathbb{Z}_n^*$.

- A global hash function h mapping into $\mathbb{Z}_n^*$.
- A second hash function g whose output is an L_1-bit integer.

In this scheme a shareholder signs a message m in the following manner. Firstly the shareholder calculates the hash of the message, i.e. $x = h(m)$. The signature share of a shareholder i then consists of

$$x_i = x^{2\Delta s_i}$$

and a 'proof of correctness' (note that $\Delta = l!$). The proof of correctness is basically just a proof that the discrete logarithm of x_i^2 to the base $x^{4\Delta}$ is the same as the discrete logarithm of v_i to the base v. Let $L(n)$ be the bit length of n. The shareholder then chooses a random number $r \in \{0, \ldots, 2^{L(n)+3L_1} - 1\}$ and computes

$$v' = v^r,\ x' = x^{4\Delta} r,\ c = g(v, x^{4\Delta}, v_i, v', x'),\ z = s_i c + r.$$

The proof of correctness is then (z, c) which can be verified by calculating

$$c = g(v, x^{4\Delta}, v_i, x_i^2, v^z v_i^{-c}, x^{4\Delta z} x_i^{-2c}).$$

To combine the shares the combiner acts as follows. Assume we have valid shares from a set $S = \{i_1, i_2, \ldots, i_k\}$ of shareholders. The combiner computes

$$\lambda_{0,j}^S = \Delta \prod_{i \in S \setminus \{j\}} \frac{i}{(i - j)}.$$

These values are derived from the standard Lagrange interpolation formula. These values are integers and it is clear that they are easy to compute. We also have, from the Lagrange interpolation formula that,

$$\Delta \cdot f(0) = \sum_{j \in S} \lambda_{0,j}^S f(j) \bmod (p'q').$$

In other words we have,

$$d \cdot \Delta = \sum_{j \in S} \lambda_{0,j}^{S} s_j$$

The combiner then computes,

$$\begin{aligned} w &= x_{i_1}^{2\lambda_{0,i_1}^{S}} \cdots x_{i_k}^{2\lambda_{0,i_k}^{S}} \\ &= x^{4\Delta^2 \sum_{j \in S}(s_j \lambda_{0,j}^{S})} \\ &= x^{4\Delta^5 d}. \end{aligned}$$

To check this signature we note that $w^e = x^{4\Delta^5}$ where $\gcd(e, 4\Delta^5) = 1$. As e is coprime to $4\Delta^5$ we can find a, b such that $a(4\Delta^5) + be = 1$ so that we finally have the signature

$$y^e = (w^a x^b)^e = x.$$

11.4 Undetachable threshold signatures

We now introduce the notion of an undetachable threshold signature. Suppose a user has a private signature key s and a public verification key v. Suppose also that the user has a 'constraint string' R, which will define what types of signature can be created. Then an undetachable threshold signature scheme will enable the user to provide n entities with 'shares' of the private signature key (where the shares will be a function of R), where the following properties must be satisfied:

- Each entity can use their share to sign a message m of their choice to obtain a 'signature share'.
- The 'correctness' of a signature share can be verified independently of any other signature shares.

- Any entity, when equipped with k different signature shares for the same message m, can construct a signature on the message m which will be verifiable by any party with a trusted copy of the public key of the user, and which will also enable the string R to be verified.
- Knowledge of less than k different signature shares for the same message m cannot be used to construct a valid signature on the message m.
- Knowledge of any number of signature shares (up to a bound polynomial in the key length[3]) for messages other than m will not enable the construction of a valid signature on message m.
- Knowledge of any number of different signature shares for constraints strings other than R will not enable the construction of a valid signature with associated constraint string R.

As discussed above, the motivation for introducing this concept is that the use of a threshold signature scheme or a detachable signature scheme on its own would not protect against all possible attacks in a mobile agent scenario. We now describe an example of such a scheme. For brevity, we only give the necessary changes to the threshold scheme in section 11.3 to form the undetachable threshold signature scheme.

Recall that the secret share for shareholder i consists of a number s_i. Let h be an appropriate hash function. The signature share of this shareholder for a message m is then

$$x_i = x^{2\cdot\Delta\cdot s_i}.$$

[3]The bound is necessary to make the scheme realisable. Without such a bound, attacks such as those of Desmedt and Odlyzko [34] will apply. The exact nature of the bound will depend on the details of the scheme.

where l is the total number of shares, $\Delta = l!$ and $x = h(m)$ is a hash of the message.

As in section 11.2 let I be the identifier of a user and let R be the user requirements. Let $H = h(I, R)$ be a hash of the requirements. We replace the share s_i with a pair $(H, t_i = H^{2\cdot\Delta\cdot s_i})$. To sign a bid B the shareholder calculates $C = h(B)$ and

$$t_i^C = (H^{2\cdot\Delta\cdot s_i})^C = H^{2\cdot\Delta\cdot s_i C} = (H^C)^{2\cdot\Delta\cdot s_i}.$$

Thus, when all the shares are combined the combiner will have a signed copy of H^C, thus achieving a signed undetachable signature.

We observe that a proof of security is given for the scheme in section 11.3 provided that k is one greater than the number of corrupt servers (in the case where k exceeds the number of corrupt servers by a greater number a slightly adapted scheme is used). With this information to hand we note that this scheme is secure as long as the undetachable scheme given in [81] is secure, and that this scheme appears to be sound.

11.5 Conclusions

By combining threshold signatures and undetachable signatures, the concept of undetachable threshold signatures was introduced. Undetachable threshold signatures enable constrained signing power to be distributed across multiple agents, thus reducing the necessary trust in single agent platforms even further, compared to only using threshold signatures or undetachable signatures.

Chapter 12

Certificate translation

Contents

In this chapter the concept of certificate translation is defined and examples of its applications are proposed.

12.1 Introduction

As we have argued in chapters 5 and 6, asymmetric cryptography and a Public Key Infrastructure (PKI) can be used to achieve secure agent communication. Digital certificates have a central role in any PKI. The traditional way to enable communication between entities in different PKIs is to utilise cross certification where a Certification Authority (CA) issues a certificate for a CA in another PKI. However, such certificates may not exist or may not be directly useable for various reasons.

Certificates used in different PKI implementations tend to have different structures and different information stored in them. This can make communication between entities of different PKIs tedious or even impossible. Although many PKIs make use of standard formats for certificates such as X.509 [76], this can still create problems since, as in the case of X.509, the standard supports extension fields that are not defined within the standard.

If agents are deployed to represent users, they are likely to encounter applications and circumstances requiring various formats of digital certificates. Since many agents would reside on mobile devices with limited memory and processing resources, the number of private keys and digital certificates carried by the agents should be kept to a minimum. In these circumstances the concept presented in this chapter, certificate translation, is likely to be useful. Other applications where certificate translation can be used include WTLS [113] and MExE [1], which are both examples of applications using public key certificates designed for a wireless mobile environment. Some of the work described in this chapter has been previously published in [12].

The rest of this chapter has the following structure. In section 12.2 we define and describe the concept of *certificate translation.* Section 12.3 describes security considerations that are introduced with translated certificates. In section 12.4 we describe some possible applications for certificate translation. In section 12.5 we describe how certificate translation can be used for WAP and MExE. Section 12.6 proposes certificate translation as an extension to SCVP. Section 12.7 outline a protocol for certificate translation. Section 12.8 gives the conclusions of the chapter.

12.2 General concept

By electronically signing a public key along with an entity identifier and possibly various other attributes, a certification authority provides assurance regarding the relationship between the public key and the included attributes. If any of these attributes are changed a new CA signature must be applied to the certificate.

In certain circumstances, such as where a certificate carries an incompatible certificate field or is of an incompatible type from that which the end user understands, it would be useful to make changes to existing certificates. Such changes can be achieved using certificate translation, a concept that forms the main focus of this chapter. We now define the terms underlying this translation concept. A *translated certificate* is a certificate to which changes have been made since it was originally issued. Changes to a certificate's content as well as to its format (e.g. structure, coding) may have been made. The value of the public key is the only certificate field that may not be altered. A *certificate translation service* is able to accept a certificate and create a new (translated) certificate with a modified structure and/or content. A *certificate translation server* (CTS) is a server that offers a certificate translation service to clients. A certificate translation server would have to be trusted by its clients, the entities using the service, just like any traditional CA has to be trusted by its users.

If the CA who signed the original certificate is 'accessible' (and willing) it can issue a new certificate including the public key of the original certificate and any other attributes that apply. On other occasions the CA who issued the original certificate will not be available or able to issue a certain certificate. On such occasions a CA acting as a CTS, who is able to verify the original certificate, could issue a new certificate including the public key from the original certificate.

Another application of certificate translation is to translate a chain of certificates into a single certificate. This can in particular be useful in environments where the CPU, memory, or bandwidth resources of the certificate recipient are constrained, and can also be used to centralise trust and policy management in a domain.

12.3 Security considerations

In this section security considerations that are introduced by translated certificates are described. Certificate content assurance (section 12.3.1), certificate revocation (section 12.3.2), and liability (section 12.3.3) are all issues that need to be considered.

12.3.1 Certificate content assurance

A certificate translation service will in practice act as a CA. A CA usually takes certain measures to ensure that the information it puts into a certificate is correct. This typically includes measures to ensure that a claimed identity actually belongs to the claimant and that an entity that supplies a public key to be included in a certificate is also in possession of the corresponding private key. The effort a CA puts into ensuring the correctness of this information is usually defined either implicitly or explicitly in a certification practice statement. A certificate translation service would have to rely on measures taken by other CAs for the purposes of verifying identity and validating public keys. The signature on the certificate that is to be translated, as well as any intermediate certificates required to form a certificate chain to a trusted root, have to be validated. A certificate translation service would have to publish its own certification practice statement specifying under what circumstances a translated certificate will be produced.

12.3.2 Revocation

By translating a certificate the translator is in practice issuing a new certificate and therefore acting as a CA. If the original certificate is revoked the translated certificate should also be revoked. In certain environments and applications this problem can be tackled through giving translated certificates a minimal validity period. Where this is not possible, revocation must be dealt with in a proper way, just as in any other PKI.

If clients are going to be able to validate the status of the original certificate even when only in possession of the translated version, enough information must be provided in the translated certificate (or along with it) to validate it against certificate revocation lists, or any other means used to advertise revocations for certificates issued by the original CA. In the case where X.509 v3 [76] is used in the translated certificate and the original certificate conforms to X.509 v1, v2, or v3, the issuer name and certificate serial number can be stored in an extension of the translated certificate, in order to allow traceability of the original certificate.

12.3.3 Liability

A certification authority may place limitations on the use of its certificates, in order to control the risk that it assumes as a result of issuing certificates. For instance, it may restrict the community of certificate users, the purpose for which they may use its certificates, and/or the type and extent of damages that it is prepared to make good in the event of a failure on its part, or that of its end-entities. These matters can be defined in a certificate policy.

It is most likely that by translating a certificate any liabilities undertaken by the original CA will no longer apply. For certain applications, or where the certificate policy refers to the public key rather than to the certificate as such, liabilities undertaken by the CA originally issuing a certificate might still apply. If the translating service is prepared to do so, it can issue translated certificates under similar policies as the original certificate was issued; otherwise it can issue the certificate using a more appropriate policy for the situation.

12.4 Applications for certificate translation

In this section we will describe some situations where certificate translation can be used.

12.4.1 Translating between incompatible certificate types

Many types of certificate exist today. A few examples are: X.509 [76], X9.68 [2], Open-PGP certificates [22], SPKI certificates [37], EMV certificates [38], and WTLS certificates [113]. Applications designed to use one type of certificate usually do not work very well with a different type of certificate. In some cases certificates are very application-oriented and using a different type of certificate does not make any sense. For other applications different types of certificates are used for the same purpose, possibly in different domains. Under such circumstances translation of certificates from one format into another would allow entities using different types of certificates to communicate using the advantages of public-key cryptography and digital certificates.

12.4.2 Translating incompatible certificate fields

Although many PKIs make use of standard formats for certificates such as X.509 [76], this can still create problems since, as in the case of X.509, the standard supports extension fields that are not defined within the standard. Two different CAs can be issuing certificates carrying extensions with the same purpose but with different names. It is also possible that different CAs are issuing certificates carrying extensions with the same name and syntax but with different purposes.

This interoperability issue is made significantly more serious if the proprietary extensions are marked as critical. That is, if indications are made in the certificate that the verifier is obliged to process these extensions when verifying the certificate. Otherwise ignoring unrecognised extensions is always an option for the verifying party.

12.4.3 Delegating path validation

If a client does not have sufficient processing or networking resources to perform path validation for each certificate it receives, path validation can be delegated to a certificate translation server. The CTS validates the certificate chain and issues a new certificate carrying the public key of the last certificate in the chain.

12.4.4 Centralised trust and policy management

For organisations requiring a centrally imposed policy and management function, it is unacceptable to allow a client to manage its own set of trusted roots, or the policies that it accepts during path validation. A certificate translation server can enforce policy decisions while performing path validation. After validating a certificate chain the certificate translation server can, if appropriate, issue a translated version of the last certificate in the chain, at the same time imposing restrictions regulated through its policy.

12.5 Scenarios

In this section we motivate the use of translated certificates by showing how the concept can be used within WAP (section 12.5.1) and MeXE (section 12.5.2).

12.5.1 WAP

Need for certificate translation in WAP

WAP (Wireless Application Protocol) [112] is considered here as a possible application for which certificate translation could prove to be of advantage.

In WTLS (Wireless Transport Layer Security) [113], the security protocol designed for WAP, certificates are used for server authentication as well as for client authentication when so requested. Digital certificates can also be used in WAP for key agreement. The WAP specification specifies three supported formats for certificates and allows additional certificate types to be added in the future. The currently specified certificate types are X.509v3 [76], X9.68 [2], and a WTLS certificate, which is a certificate optimised for size.

WAP is intended to be used in a wireless environment by handheld devices with limited storage, processing resources and transmission bandwidth. Security parameters are negotiated during the WTLS handshake. This negotiation may also require transmission of certificates between server and client and vice versa. When certificates are known in advance, no certificates need to be transmitted between the two parties. When a certificate is transmitted the sender indicates the type of certificate that it supplies. However, there is no way for the receiving party to indicate which type of certificate it prefers or understands.

It therefore appears that, in order to be compatible with this version of WTLS, an implementation must be able to handle all three of the specified certificate types. A certificate chain can be transmitted along with a certificate. In a certificate chain, all certificates must use algorithms appropriate for the negotiated key exchange suite. E.g. if RSA has been selected, all certificates must carry RSA keys signed using an RSA signature.

Certificate translation in WAP

Certificate translation could be used in WAP in order to minimise the processing and storage requirements of certificates, as well as to provide compatibility with types of certificates other than those defined within the current WTLS specification.

When a WAP client receives a certificate with an unknown type, it can simply forward it to a server that offers a certificate translation service. The certificate translation server interprets the certificate, validates it, and rewrites it in a format that the client has requested, putting its own signature on it. However, WAP is designed to be used in an environment where large time delays exist, and it is possible that a connection would time out during the time it takes for the WAP client to establish a session with a server that offers the translation service and has the certificate translated. This should not, however, lead to any serious complications. The client can initiate a new WTLS session and, during this handshake, indicate that it already has the server certificate. The corresponding scenario where a WAP server does not understand the client certificate type would work in the same manner.

Certificate translation could also be used to reduce the computational load on a handheld device with limited CPU resources. Every certificate that needs to be verified requires some computing resources. Given that a certificate chain can contain quite a few certificates and that the processing power on some handheld devices will be very limited it may be desirable to let a CTS do the computations required. By doing this, resource requirements will be shifted from CPU resources (on the handheld device) to bandwidth requirements, assuming that CPU resources at the CTS are not limited. The CTS would, after receiving a certificate chain, verify the certificates and, if the chain terminates in a root public key trusted by the CTS, create a new certificate. This new certificate would, according to our definition, be a translation of the last certificate in the chain. After receiving the translated certificate the user could store it for future use, if applicable.

12.5.2 MExE

Need for certificate translation in MExE

MExE is another application where a certificate translation service could be advantageous.

MExE (Mobile Station Application Execution Environment), as specified by 3GPP, provides a standardised execution environment for mobile stations (typically a mobile phone with a smart card). MExE specifies three security domains [1]:

- MExE security operator domain (MExE executables authorised by the HPLMN (Home Public Land Mobile Network) operator, i.e the operator whose network the user has a subscription to).

- MExE security manufacturer domain (MExE executables authorised by the terminal manufacturer).
- MExE security third party domain (MExE executables authorised by trusted third parties).

Untrusted MExE executables are not in a specific domain, and have very reduced privileges. For each domain a root public key is installed in the MS (Mobile Station). In order for an executable to run, it has to carry a signature that can be validated using root public keys and digital certificates (certificate chains are supported). An optional mechanism, involving storing a hash of the executable along with its expiry date/time in a protected verified application list, is defined to avoid the need for signature verification each time an executable is run.

The MExE specification [1] mentions WTLS certificates and X.509 certificates but does not rule out other types of certificates.

Certificate translation in MExE

Just as for WAP, certificate translation can be used in MExE in order to minimise the processing and storage requirements for certificates as well as to provide compatibility with other types of certificates.

Since no certificate type is mandated for MExE it is possible that problems with incompatible certificate types will arise. Certificate translation can, in a very similar way as described for WAP, be used to overcome such obstacles.

The problem with recurrent signature validation of previously executed code has been taken care of through the verified application list as mentioned above. However, multiple applets downloaded and executed from the same site are likely to share the same certificate chain. A translated certificate can be used to shorten such a certificate chain in order to preserve CPU and memory resources. It is likely that an MS aware of multiple instances of signed code from the same site could do such a verification even more efficiently in terms of CPU resources. A certificate translation approach, however, could be more general, and would not require the terminal to store any intermediate states or results, and therefore would require less memory resources.

12.6 Extension to SCVP

The Simple Certificate Validation Protocol (SCVP) is currently an IETF Internet draft [85]. The protocol allows a client to offload certificate handling to a server. The server can give a variety of information about a certificate, such as whether or not a certificate is valid, a chain to a trusted certificate, and so on. SCVP has many purposes, including simplifying client implementations and allowing companies to centralise their trust and policy management.

SCVP allows a client to request the status of a certificate. This requires applications using the SCVP service to be aware of the protocol. Applications designed before the finalisation of SCVP, or which for some other reason do not support SCVP, will not be able to make use of the SCVP protocol. If a certificate translation server were used instead, standalone software able to communicate with a CTS would be able to interact with existing software without any changes to the certificate-using software. The certificate-using software would have to load the CTS public key as a trusted root public key, and certificates signed by the CTS would therefore be verifiable. Such a solution would allow a company to centralise their trust and policy management, requiring minimal changes to existing systems.

Another advantage with a certificate translation solution over the current SCVP protocol is that only a certificate would need to be stored by the client, as opposed to SCVP where a certificate and associated status information need to be stored by the client if required for future use.

Since many features of SCVP are potentially useful in a certificate translation service, certificate translation could be implemented as an extension to, or in combination with, SCVP.

12.7 Outline of a certificate translation protocol

In this section we will outline a protocol for certificate translation. The protocol described is a standalone protocol in order to show how certificate translation can be implemented. As described in section 12.6 the protocol could be incorporated into another protocol. However, for environments with restricted bandwidth, having a dedicated protocol is likely to reduce unnecessary communication overhead.

This protocol uses a simple request response model. That is, a client creates a single request and sends it to the server; the server creates a single response and sends it to the client. The client is assumed to be in possession of a trusted copy of the CTS public key prior to use of the protocol.

Certificate translation is expected to be of particular importance in wireless environments, where bandwidth is limited, and where clients have restricted processing and memory resources. Hence, in order to keep the data sent over the communication path to a minimum, the CTS can keep a database of its clients and their preferences. This will minimise the information that is sent in every translation request. The server can, for example, store the preferred certificate type, client certificate, and possibly certain certificate field information that might be specific to a client.

12.7.1 Request

A translation request is made up of the following information, of which some will not always be required:

- client identification,
- original certificate (certificate to be translated),
- original certificate type,
- new certificate type,
- new certificate content,
- certificates for path validation,
- client certificate,
- client signature.

We now consider each of these information types in a little more detail.

Client identification

Client identification is used to identify the client requesting translation. Identification can be used for things such as accounting, locating the client's certificate (if not supplied in the request), or to find the client's preferences in a database if such a database exists. In environments where this information is not required, such as in a protected private network where the translation service is available to all connected users, and there is no need for the server to keep a user database of any kind, this information can be omitted.

Original certificate

This is the certificate for which translation is requested. The complete certificate or a certificate identifier must be supplied as part of a certificate translation request.

Original certificate type

If known by the client, the type of the certificate that is submitted for translation is indicated here. Since one purpose of the protocol is to enable clients not aware of certain types of certificate to have certificates of those types translated into a known type, it must be assumed that the client is not always aware of the type of the certificate that is submitted for translation. The translation server should therefore be able to analyse the supplied certificate and come to a conclusion regarding the supplied certificate type. In environments where the translation service will be used for the purpose of translating from certificate types unknown to the client, the translation server could be configured to know which types of certificates its clients are aware and not aware of, and which types the client is likely to be requesting translation for. The certificate type field is also, when applicable, used to indicate the certificate encoding, such as BER [70] or PER [71], if known by the client.

New certificate type

This field indicates the type of certificate, and encoding if applicable, that the client expects to receive back from the translation server.

New certificate content

This part allows the client to describe any specific information that needs to be included in the new certificate. The client can specify the content of any fields it wishes to have included, or may only indicate which fields are to be included in the new certificate and let the translation server get the information from the translated certificate. Another approach would be for the client to indicate the intended usage for the new certificate. The detailed specification of this particular section requires further research so as to allow enough flexibility without requiring too great an overhead.

Certificates for path verification

Many protocols utilising digital certificates let the communicating parties include a certificate chain when exchanging certificates, in order for the other party to verify the certification path. If the client receives such a certificate chain it can forward it to the translation server.

Client certificate

When the server is not expected to already possess the requesting client's public key, or is not able to retrieve it by other means, the client should also supply its digital certificate in the request in order to enable the server to verify the signature.

Client signature

When the translation service need to be restricted to pre-registered clients only, when the service is being charged for, or when clients need to be held accountable for their translation requests for other reasons, the client signs the complete request.

12.7.2 Response

The certificate translation response is sent back to the client in response to its request. If the requested certificate translation fails the server returns an error code indicating why the request has not been fulfilled. If the request is processed successfully, a new translated certificate is returned to the client. Since the certificate will carry the translation server's signature, no further message authentication will be required in many cases. However, on occasions when the client who requested the translation will not be the end user, it is possible that the client will not be able to verify the certificate signature, and another signature would be required in the response to enable the recipient to verify that the message originates from the certificate translation server and has not been tampered with.

12.8 Conclusions

We have described the concept of certificate translation and how it can be used in a variety of scenarios. Not only can the concept prove useful to convert certificates of different types, but it could also be particularly useful in wireless environments in order to preserve bandwidth, memory, and CPU resources. A protocol for certificate translation can be implemented as a standalone protocol or as an extension to, or in combination with, existing certificate status management protocols. SCVP is a good example of an existing protocol that can be extended to incorporate certificate translation. In other environments a standalone protocol is more appropriate to keep communications overhead to a minimum.

Agents deployed in a wireless environment are believed to have access to very limited resources, and are hence prevented from carrying several digital certificates, and may be exposed to applications requiring different types of certificates. Certificate translation therefore seems to be well suited for agents in a wireless environment.

Chapter 13

Conclusions

Contents

In this chapter we summarise the main conclusions and original contributions of this thesis, and give suggestions for future work in the area.

13.1 Summary and conclusions

This thesis deals with security in multi-agent systems in general, and in particular as applied to mobile communication.

The security issues existing for open multi-agent systems have been identified. The security issues are mainly related to agent execution and the fact that, since agents are autonomous and need to act upon information received from various entities, the trustworthiness of this information need to be guaranteed by the system and verified by the agent. Security issues related to agent execution, and the fact that agents are under the control of a (perhaps untrusted) executing host, are particularly relevant to mobile agents.

The security issues for non-mobile agents can, at least in theory, to a great extent be tackled through existing security technology and protocols. However, issues related to trust and delegation in a large scale multi-agent system are non-trivial to solve. Although a public key infrastructure is likely to be an important part of the solution, agents need to be able to reason about, and make decisions based on, various security parameters. Execution of agents (mobile as well as non-mobile) on untrusted platforms is another factor introducing non-trivial security concerns, in particular related to correct agent execution and confidentiality of agent data.

There does not seem to be a single solution to the security problems introduced by mobile agents unless trusted hardware is introduced, which is likely to prove too expensive for most applications. The way forward appears to lie in a range of mechanisms aimed at solving particular (smaller) problems. This could, for example, include mechanisms that depend on agents executing on several hosts rather than on only one host, mechanisms and protocols binding agent actions to hosts, generation of various types of audit information that can be used in case of disputes, and so on. Solutions to certain problems do exist, but for mobile agents to be more widely adopted this is an area that requires further research.

A security architecture, designed for deployment of agent technology in a mobile communication environment, has been proposed in this thesis. The security architecture allows modelling of interactions at all levels within a mobile communication system. The model includes the involved parties at the highest level, then a device structure, an agent execution environment, and finally, the structure of an agent. Basic security functionality addressing the security of agents, agent platforms, and agent communication has been identified and high level outlines of how these security features can be realised have also been given.

We have also shown how conventional security protocols can be used to provide secure communication within an agent-based system. Data integrity, data origin authentication, entity authentication, non-repudiation, confidentiality, and anonymity can all be provided for agent communications.

For mobile agents it is a non-trivial task to ensure that communication originating from the agent cannot be spoofed. Several techniques to limit the threats posed to mobile agent communication exist. Nevertheless, mobile agents must be deployed with great care if the authenticity of the communication is of importance.

The FIPA agent communication specifications lack sufficient functionality to provide secure communication. By using an existing message structure such as the Open PGP message format, communications security services can be added to FIPA. We have considered where security services can be applied to agent communication within the FIPA architecture, and described the information exchange required between an agent and the ACC security services. Further detailed analysis and specification is, however, required for a complete solution.

A pragmatic solution to undetachable signatures has been proposed, relying on conventional signatures and public key certificates. Our solution has potential practical advantages by comparison with the use of undetachable signatures, and appears to offer a very similar set of security guarantees. When combined with the use of signatures by the agent platform, this solution has the potential to solve certain problems relating to transaction repudiation.

We have considered two different ways in which the deployment of multiple agents can reduce the threat to trading mobile agents from potentially malicious agent platforms. In the first approach multiple agents are equipped with 'shares' of the means to commit to a transaction. A method implementing this idea using a threshold signature scheme was outlined. In the second approach a single trusted host is employed to collect information from multiple agents on possible transactions. This host then chooses the optimal transaction and commits to it. The two approaches each have their own advantages. The first approach avoids the need for a single trusted host. However, implementing the first approach requires use of some potentially complex cryptographic signature functions. The second approach is potentially less complex from a cryptographic perspective, but does require a host which, if not completely trusted, is at least required to act neutrally with respect to the set of merchants. Both approaches are of potential practical importance in future mobile computing environments.

Threshold signatures can be used in a mobile agent scenario to spread the risk between several agents and thereby overcome the threats posed by individual malicious hosts. A pragmatic alternative to threshold signatures has been proposed. This alternative has potential practical advantages by comparison with the use of threshold signatures, and appears to offer a very similar set of security guarantees. Although the analysis was performed within the context of mobile agents, it is possible that the scheme is competitive with threshold signature schemes in other environments.

Undetachable signatures and threshold signatures are both concepts applicable to mobile agents. By combining threshold signatures and undetachable signatures, the concept of undetachable threshold signatures has been introduced. Undetachable threshold signatures enable constrained signing power to be distributed across multiple agents, thus reducing the necessary trust in single agent platforms even further, compared to only using threshold signatures or undetachable signatures.

We have described the concept of certificate translation and how it can be used for different purposes. Not only can the concept prove useful to convert between certificates of different types, but it can also be particularly useful in wireless environments in order to preserve bandwidth, memory, and CPU resources. A protocol for certificate translation can be implemented as a standalone protocol or as an extension to, or in combination with, existing certificate status management protocols. SCVP is a good candidate of an existing protocol that can be extended to incorporate certificate translation. In other environments a standalone protocol is more suitable to minimise communications overheads.

Agents deployed in a wireless environment are believed to have access to very limited resources, and are hence prevented from carrying several digital certificates, and may be exposed to applications requiring different types of certificates. Certificate translation therefore seems to be well suited for agents in a wireless environment.

13.2 Suggestions for future work

Although security mechanisms and services used in today's distributed systems and computer platforms are also applicable to multi-agent systems, for agents to fully live up to their potential and be able to represent humans for more complex tasks, issues related to trust needs to be further studied. Building trust, describing trust, and reasoning about trust are all issues that need to be addressed, and that are currently the subject of ongoing research.

Mobile agents potentially have an important role to play in future communication systems. However, to be of real use beyond very trivial tasks, security mechanisms are required. In this thesis we have proposed security mechanisms useful for mobile agents, but there is still scope for much more research in this area.

One related area not investigated in this thesis where research is ongoing is that of using mobile agents for security purposes. Mobile agents can be deployed to enhance the security of a network. Applications that have been suggested include, intrusion detection, virus detection, and security management. Applications like these do, of course, require that mobile agents themselves do not introduce unmanageable security threats. This is still an emerging area, but appears to offer interesting possibilities.

The FIPA specifications need to be further developed to include security functionality. We have proposed one way forward, but further details are needed for a complete specification. XML should also be fully evaluated for the purpose of securing FIPA agent communication.

Bibliography

[1] 3rd Generation Partnership Project (3GPP). *Mobile Station Application Execution Environment (MExE), Functional description, 3GPP, 3G TS 23.057*, stage2 (release 1999) edition, March 2000.

[2] *ANSI X9.68. Digital Certificates for Mobile/Wireless and High Transaction Volume Financial Systems: Part 2: Domain Certificate Syntax.* American National Standard Institute, 2002.

[3] Giuseppe Ateniese, Breno de Medeiros, and Michael T. Goodrich. TRICERT: A distributed certified e-mail scheme. In *Proceedings of ISOC 2001 Network and Distributed System Security Symposium (NDSS'01)*, pages 45–56. CA, 2001.

[4] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In J. Kilian, editor, *Advances in Cryptology – Crypto 2001 proceedings*, number 2139 in LNCS, pages 1–18. Springer-Verlag, Berlin, 2001.

[5] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography: the case of hashing and signing. In Y. Desmedt, editor, *Advances in Cryptology – Crypto '94 proceedings*, number 839 in LNCS, pages 216–233. Springer-Verlag, Berlin, 1994.

[6] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE: a FIPA2000 compliant agent development environment. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 216–217. ACM Press, 2001.

[7] J. Bigham, A.L.G. Hayzelden, J. Borrell, and S. Robles. Distributed control of connection admission to a telecommunications network: Security issues. In Alex L.G. Hayzelden and Rachel A. Bourne, editors, *Agent Technology for Communication Infrastructures*, chapter 6. Wiley, 2001.

[8] A. Birk. Learning to trust. In R. Falcone, M. Singh, and Y. H. Tan, editors, *Trust in Cyber-societies*, number 2246 in LNAI, pages 27–54. Springer-Verlag, Berlin, 2001.

[9] Niklas Borselius. Mobile agent security. *Electronics & Communication Engineering Journal*, 14(5):211–218, October 2002.

[10] Niklas Borselius. Security in multi-agent systems. In Y. Mun and H. R. Arabnia, editors, *Proceedings of the 2002 International Conference on Security and Management (SAM'02)*, pages 31–36. CSREA Press, Nevada, 2002.

[11] Niklas Borselius, Namhyun Hur, Marek Kaprynski, and Chris J. Mitchell. A security architecture for agent-based mobile systems. In *Proceedings – 3G2002, Third International Conference on Mobile Communications Technologies*, number 489 in IEE Conference Publication, pages 312–318. IEE, London, 2002.

[12] Niklas Borselius and Chris J. Mitchell. Certificate translation. In *Proceedings of NORDSEC 2000 – 5th Nordic Workshop on Secure IT Systems*, pages 289–300. Reykjavik University, 2000.

[13] Niklas Borselius and Chris J. Mitchell. Securing FIPA agent communication. In H. R. Arabnia and Y. Mun, editors, *Proceedings of the 2003 International Conference on Security and Management (SAM'03), Vol. 1*, pages 135–141. CSREA Press, Nevada, 2003.

[14] Niklas Borselius, Chris J. Mitchell, and Aaron Wilson. On mobile agent based transactions in moderately hostile environments. In B. De Decker, F. Piessens, J. Smits, and E. Van Herreweghen, editors, *Advances in Network and Distributed Systems Security, Proceedings of IFIP TC11 WG11.4 First Annual Working Conference on Network Security, KU Leuven, Belgium*, pages 173–186. Kluwer Academic Publishers, Boston, 2001.

[15] Niklas Borselius, Chris J. Mitchell, and Aaron Wilson. Undetachable threshold signatures. In *Cryptography and Coding - Proceedings of the 8th IMA International Conference, Cirencester, UK*, number 2260 in LNCS, pages 239–244. Springer-Verlag, Berlin, 2001.

[16] Niklas Borselius, Chris J. Mitchell, and Aaron Wilson. On the value of threshold signatures. *ACM SIGOPS Operating Systems Review*, 36(4):30–35, October 2002.

[17] Niklas Borselius, Chris J. Mitchell, and Aaron Wilson. A pragmatic alternative to undetachable signatures. *ACM SIGOPS Operating Systems Review*, 36(2):6–11, April 2002.

[18] K. P. Bosworth and N. Tedeschi. Public key infrastructures — the next generation. In Robert Temple and John Regnault, editors, *Internet and wireless security*, BTexact Communications Technology series 4, pages 95–120. IEE, London, 2002.

[19] Jeffrey M. Bradshaw. An introduction to software agents. In Jeffrey M. Bradshaw, editor, *Software Agents*, chapter 1, pages 3–46. AAAI Press / The MIT Press, 1997.

[20] Jeffrey M. Bradshaw, Stewart Dutfield, Pete Benoit, and John D. Woolley. KAoS: Toward an industrial-strength open agent architecture. In Jeffrey M. Bradshaw, editor, *Software Agents*, chapter 17, pages 375–418. AAAI Press / The MIT Press, 1997.

[21] Bernard Burg. Towards the deployment of an open agent world. In Hermes, editor, *Journées Francophones d'Intelligence Artificielle Distribuée et de Systèmes Multi-Agents (JFIADSMA2000)*, October 2001.

[22] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer. *OpenPGP Message Format, RFC 2440*. IETF, November 1998.

[23] Cristiano Castelfranchi and Yao-Hua Tan, editors. *Trust and Deception in Virtual Societies*. Kluwer Academic Publishers, The Netherlands, 2001.

[24] David L. Chaum. Untraceable electronic mail, return address, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, February 1981.

[25] David Chess, Benjamin Grosof, Colin Harrison, David Levine, Colin Parris, and Gene Tsudik. Itinerant agents for mobile computing. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 267–282. Morgan Kaufmann, San Francisco, CA, 1997.

[26] David M. Chess. Security Issues in Mobile Code Systems. In Giovanni Vigna, editor, *Mobile Agents and Security*, number 1419 in LNCS, pages 1–14. Springer-Verlag, Berlin, 1998.

[27] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. *Journal of the ACM*, 45(6):965–981, 1998.

[28] Joris Claessens, Bart Preneel, and Joos Vandewalle. Secure communication for secure agent-based electronic commerce applications. In J. Liu and Y. Ye, editors, *E-Commerce Agents: Marketplace Solutions, Security issues, and Supply and Demand*, number 2033 in LNAI, pages 180–190. Springer-Verlag, Berlin, 2001.

[29] Cloakware Corporation. *Protecting Digital Content using Cloakware Code Transformation Technology, white paper*, 1.2 edition, 2002.

[30] Bruno Crispo. Delegation of responsibility (position paper). In B. Christianson, B. Crispo, W.S. Harbison, and M. Roe, editors, *Security protocols: 6th International Workshop, Cambridge, UK*, number 1550 in LNCS, pages 118–124. Springer-Verlag, Berlin, 1998.

[31] David H. Crocker. *Standard for the format of ARPA Internet text messages, RFC 822*. IETF, August 1982.

[32] Ivan Damgård and Maciej Koprowski. Practical threshold RSA signatures without a trusted dealer. In Birgit Pfitzmann, editor, *Advances in Cryptology – Eurocrypt 2001 proceedings*, number 2045 in LNCS, pages 152–165. Springer-Verlag, Berlin, 2001.

[33] Y. Desmedt. Society and group oriented cryptography. In C. Pomerance, editor, *Advances in Cryptology – Crypto '87 proceedings*, number 293 in LNCS, pages 120–127. Springer-Verlag, Berlin, 1988.

[34] Y. Desmedt and A.M. Odlyzko. A chosen text attack on the rsa cryptosystem and some discrete logarithm schemes. In H.C. Williams, editor, *Advances in Cryptology – Crypto '85 proceedings*, number 218 in LNCS, pages 516–522. Springer-Verlag, Berlin, 1986.

[35] T. Dierks and C. Allen. *The TLS Protocol Version 1.0, RFC 2246*. IETF, January 1999.

[36] D. Eastlake 3rd, J. Reagle, and D. Solo. *XML-Signature Syntax and Processing, RFC 3275*. IETF, March 2002.

[37] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. *SPKI Certificate Theory, RFC 2693*. IETF, Septtember 1999.

[38] *EMV 4.0 Book 2, EMV Integrated Circuit Card Specification for Payment Systems - Book 2: Security & Key Management, Version 4.0*, 2000.

[39] William Farmer, Joshua Guttmann, and Vipin Swarup. Security for mobile agents: Authentication and state appraisal. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, *Proceedings of the European Symposium on Research in Computer Security (ESORICS 96)*, number 1146 in LNCS, pages 118–130. Springer-Verlag, Berlin, 1996.

[40] Jalal Feghhi, Jalil Feghhi, and Peter Williams. *Digital Certificates – Applied Internet Security*. Addison-Wesley-Longman, 1999.

[41] *FIPA 98 Specification Part 10, Version 1.0, Agent Security Management*. Geneva, October 1998. Obsolete.

[42] *FIPA ACL Message Representation in XML Specification, Document no. XC00071B*. Geneva, June 2000.

[43] *FIPA Agent Message Transport Service Specification, Document no. XC00067D*. Geneva, August 2001.

[44] *FIPA Agent Message Transport Service Specification, Document no. SC00067F*. Geneva, December 2002.

[45] *FIPS PUB 186-2, Digital Signature Standard (DSS).* Gaithersburg, MD, January 2000.

[46] Leonard N. Foner. A Security Architecture for Multi-Agent Matchmaking. In M. Tokoro, editor, *Proceedings of the Second International Conference on Multi-Agent Systems*, pages 80–86. AAAI Press, menlo Park, CA, 1996.

[47] Warwick Ford. *Computer Communications Security — Principles, Standard Protocols and Techniques.* Prentice-Hall, New Jersey, 1994.

[48] M. Genesereth and R. Fikes. Knowledge interchange format, version 3.0 reference manual. Technical Report Logic-92-1, Computer Science Department, Stanford University, 1992.

[49] Dieter Gollman. *Computer Security.* John Wiley & Sons, Chichester, 1999.

[50] Robert S. Gray, David Kotz, George Cybenko, and Daniela Rus. D'Agents: Security in a multiple-language, mobile system. In Giovanni Vigna, editor, *Mobile Agents and Security*, number 1419 in LNCS, pages 154–187. Springer-Verlag, Berlin, 1998.

[51] Ceki Gulcu and Gene Tsudik. Mixing e-mail with BABEL. In *Symposium on Network and Distributed System Security (NDSS'96)*, pages 2–16. IEEE, 1996.

[52] Colin G. Harrison, David M. Chess, and Aaron Kershenbaum. Mobile agents: Are they a good idea? Computer science research report, IBM Research Center, New York, NY, 1995.

[53] Vesna Hassler. *Security Fundamentals for E-commerce.* Artech House, 2000.

[54] Alex L. G. Hayzelden and Rachel A. Bourne, editors. *Agent Technology for Communication Infrastructures*. John Wiley & Sons, Chichester, 2000.

[55] Qi He, Katia P. Sycara, and Timothy W. Finin. Personal security agent: KQML-Based PKI. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents*, pages 377–384, New York, NY, 1998. ACM Press.

[56] Fritz Hohl. A model of attacks of malicious hosts against mobile agents. In *Proceedings of the ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems: Secure Internet Mobile Computations*, pages 105–120, 1998.

[57] Fritz Hohl. Time limited blackbox security: Protecting mobile agents from malicious hosts. In Giovanni Vigna, editor, *Mobile Agents and Security*, number 1419 in LNCS, pages 92–113. Springer-Verlag, Berlin, 1998.

[58] R. Housley. *Cryptographic Message Syntax, RFC 3369*. IETF, August 2002.

[59] *IEEE P1363, Standard specifications for public key cryptography*, 2000.

[60] Takeshi Imamura, Blair Dillaway, and Ed Simon. XML encryption syntax and processing, W3C candidate recommendation, August 2002.

[61] *ISO/IEC 11770-1, Information Technology — Security techniques — Key management —Part 1: Framework*. Geneva, 1996.

[62] *ISO/IEC 13888-1. Information technology — Security techniques — Non-repudiation — Part 1: General*. Geneva. 2nd edition, to be published.

[63] *ISO/IEC 13888-2. Information technology — Security techniques — Non-repudiation — Part 2: Mechanisms using symmetric techniques*. Geneva, 1998.

[64] *ISO/IEC 13888-3. Information technology — Security techniques — Non-repudiation — Part 3: Mechanisms using asymmetric techniques.* Geneva, 1997.

[65] *ISO/IEC 14516 / ITU-T X.842. Information technology — Security techniques — Guidelines for the use and management of Trusted Third Party services.* Geneva, 2002.

[66] *ISO/IEC 14888-1. Information technology — Security techniques — Data signatures with appendix — Part 1: General.* Geneva, 1998.

[67] *ISO/IEC 14888-2. Information technology — Security techniques — Data signatures with appendix — Part 2: Identity-based mechanisms.* Geneva, 1999.

[68] *ISO/IEC 14888-3. Information technology — Security techniques — Data signatures with appendix — Part 3: Certificate-based mechanisms.* Geneva, 1998.

[69] *ISO/IEC 7498-2 / ITU-T X.800, Data Communication Networks: Open System Interconnection (OSI); Security, Structure and Applications — Security Architecture for Open Systems Interconnection for CCITT Applications.* Geneva, 1991.

[70] *ISO/IEC 8825-1 / ITU-T X.690, Information Technology — ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).* Geneva, 1998.

[71] *ISO/IEC 8825-2 / ITU-T X.691, Information Technology — ASN.1 Encoding Rules: Specification of Packed Encoding Rules (PER).* Geneva, 1998.

[72] *ISO/IEC 9797-1. Information technology - Security techniques — Message Authentication Codes (MACs) — Part 1: Mechanisms using a block cipher.* Geneva, 1999.

[73] *ISO/IEC 9797-2. Information technology — Security techniques — Message Authentication Codes (MACs) — Part 2: Mechanisms using a hash-function.* Geneva, 2002.

[74] *ISO/IEC 9798-3 Information technology — Security techniques — Entity authentication mechanisms — Part 3: Mechanisms using digital signature techniques.* Geneva, 1998. 2nd edition.

[75] *ISO/IEC 9798-4, Information technology — Security techniques — Entity authentication — Part 4: Mechanisms using a cryptographic check function.* Geneva, 1999. 2nd edition.

[76] *ITU-T Recommendation X.509, Information technology — Open Systems Interconnection — The Directory: Public-key and attribute certificate frameworks.* Geneva, 2000. 4 edition, Also ISO International Standard 9594-8.

[77] Markus Jakobsson. Flash mixing. In *The Eighteenth annual ACM symposium on Principles of distributed computing*, pages 83–89. ACM press, 1999.

[78] Wayne Jansen and Tom Karygiannis. *NIST Special Publication 800-19 – Mobile Agent Security.* National Institute of Standards and Technology, 1999.

[79] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):275–306, 1998.

[80] N. R. Jennings and M. Wooldridge. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[81] Panayiotis Kotzanikolaou, Mike Burmester, and Vassilios Chrissikopoulos. Secure transactions with mobile agents in hostile environments. In E. Dawson, A. Clark, and C. Boyd, editors, *Information Security and Privacy, Proceedings of the 5th Australasian Conference ACISP 2000*, number 1841 in LNCS, pages 289–297. Springer-Verlag, Berlin, 2000.

[82] Susan K. Langford. Threshold DSS signatures without a trusted party. In D. Coppersmith, editor, *Advances in Cryptology – Crypto '95 proceedings*, number 963 in LNCS, pages 397–409. Springer-Verlag, Berlin, 1995.

[83] O. Lazaro, J. Irvine, D. Girma, J. Dunlop, A. Liotta, N. Borselius, and C.J. Mitchell. Management system requirements for wireless systems beyond 3G. In *Proceedings - IST Mobile & Wireless Communications Summit 2002*, pages 240–244, 2002.

[84] Michael Luck and Mark d'Inverno. A conceptual framework for agent definition and development. *The Computer Journal*, 44(1):1–20, 2001.

[85] A. Malpani, R. Housley, and T. Freeman. *Simple Certificate Validation Protocol (SCVP), Internet Draft.* IETF, June 2002.

[86] Gary McGraw and Edward W. Felten. *Securing JAVA: Getting Down to Business with Mobile Code.* John Wiley & Sons, New York, NY, 2nd edition, 1999.

[87] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* Discrete Mathematics and Its Applications. CRC Press, 1996.

[88] S. Micali. *Simultaneous electronic transactions.* US Patent 5666420, 1997.

[89] George C. Necula and Peter Lee. Safe, untrusted agents using proof-carrying code. In Giovanni Vigna, editor, *Mobile Agents and Security*, number 1419 in LNCS, pages 61–91. Springer-Verlag, Berlin, 1998.

[90] H. Penny Nii. Blackboard systems. In A. Barr, P.R. Cohen, and E.A. Feigenbaum, editors, *The Handbook of Artificial Intelligence, Volume IV*, pages 1–82. Addison-Wesley, New York, 1998.

[91] S. Poslad, P. Buckle, and R. Hadingham. The FIPA OS agent platform: Open source for open standards. In Jeffrey Bradshaw and Geoff Arnold, editors, *Proceedings of the 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents*, pages 355–368, UK, 2000.

[92] S. Poslad and M. Calisti. Towards improved trust and security in FIPA agent platforms. In *Autonomous Agents 2000*, June 2000.

[93] B. Ramsdell. *S/MIME Version 3 Message Specification, RFC 2633.* IETF, June 1999.

[94] H. Reiser and G. Vogt. Security requirements for management systems using mobile agents. In *Proceedings of the Fifth IEEE Symposium on Computers and Communications: ISCC 2000*, pages 160–165, 2000.

[95] James Riordan and Bruce Schneier. Environmental key generation towards clueless agents. In G. Vigna, editor, *Mobile Agents and Security*, number 1419 in LNCS, pages 15–24. Springer-Verlag, Berlin, 1998.

[96] Volker Roth. Secure recording of itineraries through co-operating agents. In *Proceedings of ECOOP Workshop on Distributed Object Security and 4th Workshop on Object Systems: Secure Internet Mobile Computations*, pages 147–154, France, 1998. INRIA.

[97] RSA Laboratories. *PKCS #7: Cryptographic Message Syntax Standard*, 1993. version 1.5.

[98] Tomas Sander and Christian Tschudin. Protecting mobile agents against malicious hosts. In Giovanni Vigna, editor, *Mobile Agents and Security*, number 1419 in LNCS, pages 44–60. Springer-Verlag, Berlin, 1998.

[99] Michael Schillo, Petra Funk, and Michael Rovatsos. Using trust for detecting deceitful agents in artificial societies. *Applied Artificial Intelligence*, 14(8):825–848, September 2000.

[100] Fred B. Schneider. Towards fault-tolerant and secure agentry. In M. Mavronicolas and P. Tsigas, editors, *Proceedings of the Eleventh International Workshop on Distributed Algorithms*, number 1320 in LNCS, pages 1–14. Springer-Verlag, Berlin, 1997.

[101] *Security Model for the Next-Generation secure computing Base.* white paper, Microsoft Corporation, 2003.

[102] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[103] Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *Advances in Cryptology – Eurocrypt 2000 proceedings*, number 1807 in LNCS, pages 207–220. Springer-Verlag, Berlin, 2000.

[104] Masakazu Soshi and Mamoru Maekawa. The Saga Security system: A security Architecture for open Distributed systems. In *Proceedings of the 6th IEEE Workshop on Future Trends of Distributed Computing Systems*, pages 53–58. IEEE, 1997.

[105] P F Syverson, D M Goldschlag, and M G Reed. Anonymous connections and onion routing. In *Proceedings: IEEE Symposium on Security and Privacy*, pages 44–54. IEEE Computer Society Press, 1997.

[106] Chelliah Thirunavukkarasu, Tim Finin, and James Mayfield. Secret agents - a security architecture for the KQML agent communication language. In *Proceedings of the Intelligent Information Agents Workshop* held in conjunction with *Fourth International Conference on Information and Knowledge Management CIKM'95*, pages 176–184, Baltimore, December 1995. IEEE Computer Society Press.

[107] *Trusted Computing Group — Main Specification, Version 1.1a*, 2001.

[108] P.J. Turner, D.R. Basgeet, N. Borselius, E. Frazer, J. Irvine, N. Jefferies, N.R. Jennings, M. Kaprynski, O. Lazaro, S. Lloyd, C.J. Mitchell, K. Moessner, T. Song, E. Homayounvala, D. Wang, and A. Wilson. Scenarios for future communications environments, technical report ECSTR-IAM02-005. Technical report, Department of Electronics and Computer Science, Southampton University, October 2002.

[109] V. Varadharajan, P. Allen, and S. Black. An analysis of the proxy problem in distributed systems. In *Proceedings: 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 255–275, Los Alamitos, CA, May 1991. IEEE Computer Society Press.

[110] Giovanni Vigna. Protecting mobile agents through tracing. In *Proceedings of the Third ECOOP Workshop on Operating System support for Mobile Object Systems*, pages 137–153, Finland, June 1997.

[111] Michael Walker and Tim Wright. Security. In Fridhelm Hillebrand, editor, *GSM and UMTS – The Creation of Global Mobile Communication*, chapter 15, pages 385–406. John Wiley & Sons, Chichester, 2002.

[112] WAP Forum. *Wireless Application Protocol, Architecture Specification*, July 2001. WAP-201-WAPArch-20010712.

[113] WAP Forum. *Wireless Application Protocol, Wireless Transport Layer Security*, April 2001. WAP-261-WTLS-20010406-a.

[114] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, March 1992.

[115] U. G. Wilhelm, S. Staamann, and L. Buttyàn. Introducing trusted third parties to the mobile agent paradigm. In J. Vitek and C. Jensen, editors, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, number 1603 in LNCS, pages 471–491. Springer-Verlag, Berlin, 1999.

[116] H. Chi Wong and Katia Sycara. Adding Security and Trust to Multi-Agent Systems. *applied Artificial intelligence*, 14(9):927–941, 2000.

[117] Michael Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, Chichester, 2002.

[118] XML key management specification (XKMS 2.0), March 2002.

[119] Bennet Yee. A sanctuary for mobile agents. In Jan Vitek and Christian Jensen, editors, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, number 1603 in LNCS, pages 261–274. Springer-Verlag, Berlin, 1999.

[120] A. Young and M. Yung. Sliding encryption: A cryptographic tool for mobile agents. In Eli Biham, editor, *Proceedings of the 4th International Workshop on Fast Software Encryption, FSE' 97*, number 1267 in LNCS, pages 230–241. Springer-Verlag, Berlin, January 1997.

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