ASITA: Advanced Security Infrastructure for Multi-Agent-Applications in the Telematic Area

vorgelegt von
Diplom-Informatiker
Torge Schmidt

Von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.-Ing. –

genehmigte Dissertation

Promotionsausschuß:

Vorsitzender: Prof. Dr. Dr. h.c. R. Popescu-Zeletin
Berichter: Prof. Dr.-Ing. H. Krallmann
Berichter: Prof. Dr. P. Pepper

Tag der wissenschaftlichen Aussprache: 08.01.2002

Berlin 2002

D 83

Preface

Software agents with their specific attributes are seen as a promising approach to fulfill the conceptual and operational demands for service provisioning in the telecommunication area. Mobile agents might autonomously roam through heterogenous and international networks in their tasks to discover, filter, aggregate, and forward information.

The more information one trusts to software agents, though, the greater is the potential of abuse. This work will shed light on the inherent dangers of mobile software agents in telecommunication applications, and, based upon the available technology, several mechanisms to circumvent the dangers will be elaborated. The work results in an encompassing security infrastructure for a software agent platform.

Therefore, this work is divided into three major parts. It starts with an introduction into the domain and gives an idea for future service access. The second part is an overview of the three major domains: software security, Java programming, and software agents. In the remainder of the work, the infrastructure is conceived and discussed. The appendix gives extracts of real code from the system built.

This work has been influenced by a lot of people, whom I wish to thank. Dr.-Ing. Sahin Albayrak provided extraordinary support and domain-specific knowledge. Thanks go to Prof. Dr. Krallmann and Prof. Dr. Pepper who greatly helped in pushing the work forward. The discussions and work with Karsten Bsufka, Thomas Joachim Wilke, and Stefan Holst resulted in fruitful ideas for this thesis. The whole staff at the DAI-Lab supported me and was involved in the development of JIAC. Detlef Raben helped proof-reading this work. Most of all, I thank my mother for everything she has done for me.

Torge Schmidt
Berlin, September 2001

Abstract

An approach to solve the security aspects raised by agent systems, especially with mobile code and in an open environment, in the context of telecommunication applications is presented.

A scenario is given that represents a typical user approach to tomorrow's integrated services. Typical applications and services are thoroughly characterized, electronic markets are introduced, the players in the new market are shown, and the new market models and related security questions are put into relation to applicable law.

The technology involved in the framework is presented. Various security related aspects are shown, Java is discussed as a programming language facilitating security and providing a secure runtime environment. Software agents and their communities are given.

An advanced security infrastructure is then devised. It comprises of mechanisms on several different abstraction and realization layers. On a low level basic mechanisms are given irrespective of the "agentness" of the system. Security functionality is added to agents and they are enabled to reason about security, service provisioning is secured.

Communities of agents are enriched with security functionality to prevent malicious intruders to enter the protected domain, "own" agents are prevented from migrating to mischieveous other agent places. Crossing borders of domains, means for secured e-commerce even between unknown agents are provided.

Contents

List of Figures

List of Tables

Part I

Introduction and Domain

Chapter 1
Introduction

"I am the beginning and the end.
I bring order into chaos."
Borg Queen, Star Trek: First Contact

1.1 Synopsis

This Chapter gives an overview over the thesis. It especially constraints the work into a specific application domain, and gives the limits and boundaries of the elaborations. The Chapter closes with a structure of the work.

1.2 Domain Definition

The telecommunication market is expanding rapidly and players in that market are facing increasingly stiff competition. The key to commercial success in the telecommunication market will be the provisioning of adequate services, the focus shifting from a purely technological one to one of convenience and usefulness. An important prerequisite is the effective management of basic telecommunication infrastructure supporting the rapid deployment of new services.

These future services will determine the market shares to be gained. Not only must their time to market be reduced, but also other requirements need

to be fulfilled, e.g. dynamic service development and configuration. Due to a demand for permanent availability the motto to be followed is information "for everybody, anywhere, anytime." [ist99] Service maintenance must not interfere with continuous service usage; future services must allow for personalization, meet security demands, and provide management functionality. Furthermore, asynchronous service usage has to be supported as well as demand-driven service combination and integration. Not the least, services must allow for access independent of specific technologies and terminal equipment.

Beside the technical requirements, new business models must be developed reflecting the fact that various actors in new roles, e.g. content provider or application service provider, will need to co-operate and coordinate in order to provide these future services. Companies will provide integrated solutions with own and third-party services being bundled on their platforms. These platforms will realize required infrastructure functionality and enable various means of access by facing influencing factors and developments of the future telecommunication market such as consumer devices (mobile phones, screen phones, PDAs), networks (e.g. GPRS, UMTS), languages, and software technologies (Java, Jini), consumer demands and trends like convenience of use, mobility, and ubiquitous computing.

Each role participating in the future telecommunication world will have specific requirements to such service platforms. These demands differ in the extent of infrastructure being needed for service usage and provisioning, according to different necessities regarding aspects such as security, personalization, asynchronous usage, mobility, device-independency, and supporting tools.

The issues outlined above open up new perspectives for services and applications, especially in the domains of service and network-management, electronic business, mobility supporting services, and "Intelligent Home." The future of telecommunication services will be highlighted in Chapter 2.

Agent-oriented technology seems to be an adequate way to combine high quality of service with reduced costs of service development, rollout, and provisioning. (cf. Chapter 5) It enables quick reaction to changing demands and looks like an ideal candidate to realize the telematic and telecommunication applications of the future. [Dr.01]

Very important aspects of the acceptance of the new technology by the users are the questions of security involved. [Aud01] This covers aspects of reliability, trust, non-repudiation, protection against eavesdropping and ma-

nipulation, electronic money, protection against fraud, and a lot more. (cf. Chapter 3)

But there are more roles involved in the new telecommunication market than the end-user. Service provider, platform provider, information broker, etc. each have their own requirements, which will be discussed in more detail in Section 2.5.

1.3 What This Is About

The ultimate goal of this work is to present a comprehensive security infrastructure for an agent toolkit. The infrastructure will comprise of basic security functionality in software agents, supplemented by mechanisms on the level of agent communities, and the integration of non-agent-aware protocols. Supporting tools are envisioned as well, but are neither regarded as an integral part of the infrastructure nor are they fully discussed. The infrastructure has been prototypically implemented within a research project.

The security infrastructure to be developed will be based on the agent architecture CASA. [Ses02] The conceptualization of the ASITA work was co-developed with the CASA architecture and its prototypical implementation.

The single aspects of the presented solution are motivated by the specific requirements of the application domain, the employed agent technology, common security mechanisms and good practices, and the programming language as will be used.

The result of the work is an infrastructure for running software agents in an open environment, but secured against various attacks and fraud attempts. The acceptance of users and providers of services will be raised and that will foster the penetration of agents into the telecommunication and service provisioning domain on a large scale. As will be seen in Section 11.3, the proposed framework will "boldly go where no agent platform has gone before."

1.4 What This Is Not About

This work does not deal with more hardware-oriented security techniques. For instance, hardware leaking information about the performed operations, as e.g. exploited by differential power analysis [KJJ99], is not in the scope of this paper.

Neither is steganography, i.e. how to hide messages in other messages, covered in this work. Covering safety aspects like backups and availability is generally out of discussion as well. An exception is the notion of the Java programming language, which provides for secure programs due to its safety features. (cf. Chapter 4)

This thesis doesn't deal with network security or host system security in general. No mechanisms to overcome today's shortfalls in system architecture, hardware, networks, or protocols are discussed, like fighting denial of service attacks, host address spoofing, or traffic analysis. Specific aspects especially with respect to mobile agent technology or for providing a secure execution environment will be mentioned, though.

Beside security considerations, other management aspects of the FCAPS service model, [ITU97a] being fault, configuration, accounting, and performance management, are not covered.

To embed software agents into a working electronic market environment, there have to be means for exchange of goods and of money, based of accounting data. All of these aspects are not covered in this work. The exchange of money is a matter of either electronic cashing or conventional business billing processes. Goods exchange is dealed with in the service provisioning framework of the respective agent platform if the discussion is about electronically delivery, or is again done by already deployed business processes. Collection of billing data is to be handled by the service provisioning procedure as well.

1.5 Structure of the Thesis

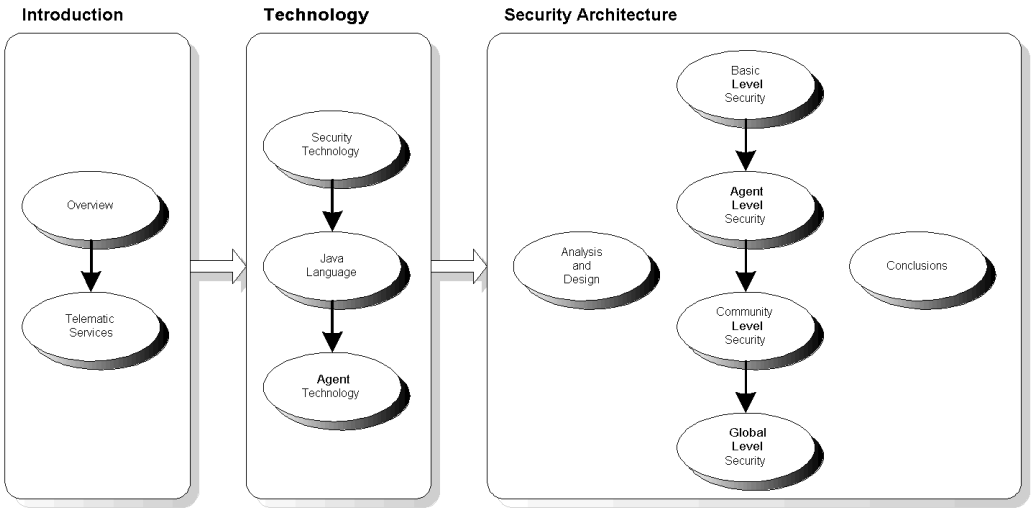

Figure 1.1: Structure of the Thesis

The work is divided into three major parts. (cf. Figure 1.1) Part I in this Chapter 1 gives an introduction into the paper, the aims, and prospected solutions. Chapter 2 delves into the requirements of the future landscape of telematic services by presenting an example scenario and discussing relevant parts of the future development. Applications and services are then more thoroughly characterized, electronic markets are introduced, the players in the new market are shown, and the new market models and related security questions are put into relation to applicable law.

In Part II the state of the art of the relevant technologies used to fulfill the requirements as analysed in Part I are shown. Chapter 3 discusses related security considerations and the mechanisms of security that will be employed later in this thesis. The basics of security technology are summarized to enable the reader to understand the complex and difficult layers, algorithms, and protocols involved. Further, requirements for secure systems are discussed and common attacks are given to demonstrate how and against which threats to protect these systems.

Following that, the Java programming language is introduced with special considerations of its features pertaining to the production of secure programs as well as its capabilities for providing a secure runtime environment and to protect the executing host against mischief.

Last in that Part, software agents and their communities are presented and their commonly perceived properties and advantages are shown. Platforms and toolkits as realizations of agent concepts are discussed and often heard misunderstandings and glorifications are put into perspective.

Based on the intermediary results and presented knowledge, a security system for agents is designed in Part III. According to engineerical practices, the system is designed bottom-up: After an analysis phase, the founding building blocks are developed, thereupon in several iterations creating more complex systems synthesized by the lower layers, resulting in an encompassing total.

Part III begins with an analysis of the needs of the system and available prerequisites to begin with in Chapter 6. Other approaches of related works are analyzed and an advanced security framework is designed.

Chapter 8 describes mechanisms on the agent level, so that a single agent is enabled to deal with security mechanisms. Agents are endowed with functionality to secure themselves against intruders or prying eyes, and to restrict service provisioning.

The Chapter 9 deepens the considerations of security aspects of multi-agent systems, where several agents form one application. To facilitate secure communities of agents, on that level agents for security related tasks are introduced, respectively existing infrastructure management agents are enhanced.

The following Chapter 10 then describes mechanisms to be provided to agents independent of their community deployment, crossing the borders of their administrative domain. Functionality is provided to allow for secure agent-based e-commerce on a global level even between unknown and untrusted parties.

The concluding Chapter 11 points out the achievements of this work. Related work is described and set into context for this paper. Possibilities for further enhancements to be derived from this thesis are shown.

The Appendices give listings of program code and other detail not appropriate for inline text.

Chapter 2
Telematic Services

"If you have walked all these days
with closed ears and mind asleep,
wake up now!"
Gandalf, The Lord of the Rings

2.1 Synopsis

This Chapter shall shed some light on the problem domain, the rest of the work is placed in. It gives a discussion of the application area which is the target field for the systems that will be created using the framework as developed in Part III. The word "telematic" is a synthesis of telecommunication and information technology.

The description begins with a very informal presentation of a typical scenario in the telematics domain, how a future service and its use is envisoned by experts in the domain. Though it is very colloquial, important aspects of the security demands can be identified.

The discussion is continued by describing characteristical features of applications and service usage in that domain. Following, electronic markets as means to model real-world market situations are introduced. Role models for players in the telematic area are shown that allow to categorize tasks and interrelationships in the domain. The Chapter closes with a discussion of jurisdictional questions in the domain of electronic markets, privacy, and multi-country distributed software.

2.2 Demonstration Scenario

In this Section, a typical application for agent-based technology of electronic commerce in the traffic telematic area is given. It is provided to give an ostensive view on the domain and inherent requirements, which will be elaborated in the following Sections. The security aspect is given specific attention. The explanations of specific security terms and mechanisms is delved into in Chapter 3.

The initial installation of the relevant software isn't considered either. It is assumed, that all installed platforms were installed securely. The process must have happened before the service usage as described below is begun.

The first installation might require a service registration as well. For simplicity reasons, this isn't treated either. Ultimatively, the registration boils down to out-of band communication for authentication. That might be personal appearance at a service provider to prove one's existence and data. It might include external entities, like trust centers, birth records, phone calls, presenting a legitimation or an official identity card, etc. Those can never be handled with the system itself, as long, as the process isn't anchored securely.

Software update is a related area. Again, either secured out-of band mechanisms must be used, or a secure anchor like trusted certificates are to be used. (cf. Section 3.7.5)

A general security requirement for all of the communication of the installed software is to secure that communication from tampering with: no part of the data transfer should be alterable by an active attacker on the line without this modification being noticed by the communicating parties.

Guaranteed connectivity can not be achieved without dedicated and specialized network hardware, that isn't present on the open network environment as assumed. An active man-in-the-middle attacker has always the option of interrupting the "cable."

For privacy reasons, the user might want, in addition to all of the communication, even the data collection and search specification, that his data can't be eavesdropped on by unauthorized third parties.

The service providers, though, possibly don't want to offer the service to just anyone. Pre-registration might be required, that can be authenticated by trusted certificates. The provider then might want to restrict access, after verifying authenticity, only to registered and authorized users.

It is assumed, that the user is using a laptop for his tasks, and can connect to the Internet by some way, including wireless. Possible options thereof include UMTS or wireless LAN connections.

2.2.1 Task to Solve

As an example, the complex planning and booking of a travel is envisaged, including information collection and aggregation. The main motivation for the journey is a business trip, but the system will add value for the spare time of the user, depending on his preferences.

2.2.2 Travel Specification

The user starts on his laptop the agent-based access software and requests the travel service. The search for applicable providers is done transparently for him by the agent system. The service provider sends a mobile agent to the platform of the user, that includes the necessary graphical user interface (GUI) for specification of his needs. The user must trust the provider to send agents to his agent platform to be executed, and it must be secured, that the agent in fact migrates from and belongs to the service provider.

It is worthwhile to note, that the mobile agent is the most current version of the appropriate execution logic and data management. Thus, the user always has access to the latest and most advanced version of the software without having to trouble himself with update cycles and incompatibilities. This is in some respect similar to the application service provisioning approach.

After successful migration from the provider domain onto the user's platform, the potentially expensive connection to the network over a wireless phone can be terminated. The next step being a time-consuming one is done locally on the laptop thus saving the user money by reducing necessary online time.

The user specifies his travel: "Tomorrow date in Hamburg, CCH, 1400-1700. The day after date in Frankfurt, airport hotel, 0900-1700. Then back." The preferences of the user specify further constraints, like using first class tickets, non-smoking seat.

2.2.3 Migration to Service Provider

A mobile agent is sent from the user platform to the provider platform. This might either be the original agent, that came from the provider, or a new one spawned by it.

Again, migration security is an issue. The user will possibly restrict the set of target agent platforms where to send agents to. The receiver would like to ensure, that the mobile agent really stems from the user's platform. To prevent hostile agents, that came to the user's platform to subvert other machines from there, it must be determined, whose agent it is. After determining the originator of the agent, the receiving platform has to confirm, that the mobile agent is allowed to migrate there.

If his preferences aren't stored locally on his laptop and given the mobile agent before migration, but instead reside in the fixed network, e.g. because he needs them at different terminals, they must be communicated between agents. The sensitivity of the data demands, that nobody can eavesdrop on them. Further, only allowed parties may access them.

2.2.4 Complex Information Retrieval

The mobile agent retrieves addresses of local service providers for his task to solve. The agent contacts the providers and negotiates terms of usage. If acceptable terms are agreed upon, the mobile agent uses the service. If the received answers satisfy the service goal, and a parent agent exists (see next Subsection), the answers are sent to the parent and the collecting agent terminates. Section 2.2.7 describes the following steps for result integration.

Again, if necessary, the communications of the agents might be sensitive enough for the user that he wants to hide them from third parties. Successful service must be accounted for securely, or possibly be paid directly with electronic money. In the case of accounting, offline processing steps will use the collected accounting data for billing.

2.2.5 Problem Partitioning

If the answers aren't satisfying, and/or no service usage terms have been negotiated, the problem is partitioned according to the principles of distributed problem solving. (cf. Section 5.4) The preferences of the user and the knowledge of the agent about the problem domain influence the partitioning.

New agents are cloned, that each have individual goals to solve. The results of their plans are to be sent back to the spawning parent agent. Each newly created agent starts its work as described in Section 2.2.3.

The new agents have to be initialized with all necessary information to fulfill their tasks. The more the initial information is distributed, though, the more potential for a leak exists. It should be taken care of, that the potential benefit of a security breach is inverse to the probability of the loss of the data.

The cloned agents must be properly authorized to use the forseeable services, i.e. eventually it has to carry several different certificates issued from different authorities. On the other hand, provisions against escrowing and unlimited service usage must be taken, if spawned agents accidently go rampage. Again, the new agents must decide, which agent place is acceptable to migrate to, and the receiving place must decide, if that agent should be received and started. This time, it is not the user's platform, where the new agents come from, though.

2.2.6 Simple Communication

For simple tasks it is sufficient for the agent to directly communicate with the service provider by speech acts, (cf. Section 5.5.1) instead of migrating to its platform. The speech act then is a detailed request containing possibly sensitive information. The answer as well has some details, which the user often doesn't want to be exposed.

The contents of the communication should be protectable according to the service used. This includes protection against modification of data, spoofing of communication partners, and preventing others from spying out the data. Some means to configure the needed security requiremts of the service use and of the provisioning is needed.

2.2.7 Result Integration

The parent agent collects the information gathered by its clones. There are two generally possible ways therefore. The information is either sent by speech acts over the network as described in the previous Section. As an alternative, the spawned agents migrate back to the parent for local communication. (cf. Section 2.2.3) The requirements for simple communication or mobile agent migration apply as discussed above. The receiving agent combines the results into an optimal solution, according to the information at hand.

2.2.8 Completing the Transactions

The agent books each of the necessary tickets for the travel, depending on the scenario and the user's requirements, automatically or after manual confirmation or adaptation.

The contract conclusion will include modality agreement for paying for and delivery of tickets. This again might require remote communication or migration for local negotiation, if not already done beforehand, as just described.

The conclusion is embedded in a respective protocol, ensuring non-repudiation and measures for fair exchange, either self-enforcing or by an arbitrator.

Payment can be done by different payment mechanisms, either debit-based or credit-based, or with any kind of cyber-coins. Later billing is possible either. Ticket delivery can either be done by traditional mail service, or per electronic document exchange to the agent.

This is potentially the most sensitive part of the transaction, because "real" value changes "hands." In a business to business setting, the amount of money might be very high. The user's information like credit card numbers or electronic coins must be protected from loss. The service provider must receive his payment for the service. On the other hand, it must be assured, that the user's agent and finally the user gets what he paid for.

2.2.9 User Notification

Finally, the user will receive the results of the agent's work. Several delivery mechanisms can be supported, like email, fax, or SMS notification. If the communication to the user is over insecure channels like email, it should be protected as well against eavesdropping or modification.

2.2.10 Sub-Summary

As can be seen from the above short scenario, there is a minimal set of features that must be supported by a secure agent platform in the telematic area. Communicating parties must be authenticated and their communication protected against eavesdropping and manipulation by third parties. Migration of software agents must be supported as well as non-repudiation of communication. The executing platforms must be protected against malicious code, vice versa the agents should be protected as much as possible against malicious executing environments. The agents on the platform must be protected against each other. Contracting and the resulting fair exchange of electronic goods should be found in a corresponding infrastructure. Service use and provisioning should be configurable with respect to the security features.

The omnipresence of the necessity and the multitude of the above-mentioned features makes it clear, that it is important to have an encompassing infrastructure supporting and providing those attributes and functionalities. It can be envisaged to have this support on different levels of the architecture-to-be-developed.

2.3 Telecommunication Applications and Services

Motivated by the previous Section 2.2 the application domain of telecommunication applications will be discussed in more detail in this Section. The consequences for a new toolkit, that fits the deduced requirements of the application domain, will be shown.

In the future, the revenue in the telecommunication sector will be gained in providing and billing for *services*. Today's application service provisioning

business models and on demand generation of content are ways to that scenario. The important features of tomorrow's services will be shown in this Section, several aspects are derived from [Alb98] and [AW99].

2.3.1 Features to the Service User

The attributes of future services are divided here in two categories. In this Subsection, features are decribed that are directly visible to the user, he has an obvious benefit from the aspects. In the following Subsection, benefits more relevant to the service provider are described.

Mobility

In the context of globalization and independence of location for new services the legal circumstances and potentially restrictions in different countries must be considered. Cryptography is still deemed dangerous in the hands of the citizens in some countries, and is accordingly supervised. Service provisioning must adapt to the legal environment. (cf. Section 2.6) On the other hand, the consumer potentially doesn't want to do business in those environments and must accordingly be enabled to restrict the service usage to minimal acceptable standards.

Common telecommunication applications today assume a fixed client-server relationship between service provider and user, where the user typically is fixed to a specific hardware device at a specific location. In the future, this will change. The user demands mobility, with different aspects as described in the following.

Device Independence The user of future services will change his terminal device even for the same service. For example, at work he might use his stationary PC, while in a train he makes use of a laptop, and in other environments a PDA or a WAP-enabled cell phone will be used as an access device, at home a set-top box might be applied. The telecommunication service of the future shall be usable on all of the potential platforms.

The service provided and the information returned shall be the same for all platforms. Only the presentation of the data will differ according to the capabilities of the specific device.

User Mobility While using the same service, the user changes its physical location. In the example of Section 2.2, part of the travel route might be replanned during the trip itself due to traffic jam conditions. Combined with device independence the user will then be able to access and continue to use a service independent of the access media and his location.

On the service side this requires continuation of a service even if the synchronous connection to the user is lost for a short interval because of location or device change. The service must provide a method of asynchronous usage even for disconnected users.

Depending on the deployment of the network infrastructure, tomorrow's services will become ubiquitous in the sense, that the user can access the network at any location.

Code Mobility The future will see an increasing amount of mobile program code to make efficient use of sparse resource like network connectivity or processing power. The code itself will wander through the network and seek the execution place best fitted to the situation at hand. If complex algorithms are to be processed, the code will migrate to a fast machine. If significant amount of memory is to be used, the code will seek a machine with a lot of ram. If specific information is to be filtered out of a database, the program will move to the database and process the data locally to reduce network load. etc.pp.

Mobile code adds to the resistance against connection failure and is a way to enable asynchronous service usage. The advantages for the user have been shown in Section 2.2.

Service Personalization

"One face to the customer" is a service provisioning doctrine implemented today. But perhaps Bill Gates and an eight year old boy have different needs and expectations upon drawing their pocket money from a bank account?

The service from tomorrow will, for the provider, still have the benefit of being the same for each customer, thus holding maintenance cost at a minimum. To the user, the service can be customized to his needs. The appearance as well as the functionality can be tailored by the user to its wants.

User Profiling

Users are bored by having to provide the same details over and over again. With the increased circulation of small devices with only limited ergonomic features like WAP-phones comes demand for requiring as little input to a service as necessary. By saving a user profile and forwarding the data to a service when needed, the user can be relieved from input tasks. Profiling is tightly intercoupled with service personalization.

This feature is under heavy controversial discussion, an exemplary forum is [Wei]. While some users want their privacy protected, others simply don't care and prefer ease of use over privacy. The debate is not an issue of this thesis, but should be remembered and closely followed in the future.

Delegation

Again, the user should be relieved of tedious and repetitive work as much as possible. This will include delegating a task to be done to autonomous entities. The user should be interacted with only at the start of service usage and only as much as needed. Other information sources will be used to derive other necessary data, like the user profile as described above.

The delegation will include processing the request in the network, even with the user not online. The results of the service usage will be forwarded to the user according to his preferences or as specified in the request. Unified messaging mechanisms will be employed for feedback to the user.

It should be kept in mind, that giving control out of hand might not be desirable to some users. Thus, the service must be at least configurable in its transparency attributes, giving feedback to the user in a verbosity of his choice.

Robustness

The service execution must be able to adapt to unforseen events. There must exist failure reaction schemes and rollback points. This increases service availability and user acceptance even under difficult conditions due to the specific usage environment.

2.3.2 Features to the Service Provider

In this Subsection, attributes of future services with pre-dominant relevance to the service provider are described. Where appropriate, the visible benefit for the user is given as well.

Networks

"People operate in two worlds today – one of computing and one of communications" said Lew Platt, chairman and chief executive of Hewlett-Packard in 1999. Up to today there exists a separation between telephony and data carrier networks. Further, the access to the carrier network is dependent on the access network and applicable devices. It is expected, that in the future the different networks converge into one single large network which carries voice and data transmissions. For this reasons, migration paths from the diversity to unification are sought, and enabling technology is developed.

Networks in the future will become ubiquitous. One isn't restricted any longer to the one data link connected to one terminal, e.g. at home. Instead, toasters and similar day-to-day devices will get networked. Further, the network will appear everywhere, e.g. by employing wireless carrier technology like Bluetooth [Met99] and service discovery software architectures like Jini. [Sun99a]

As a consequence, services will be independent of the carrier used to access them, and they must be independent of the access device. For this work it is presumed, that the global internet will be used for service provisioning. [Inf81a, DH98] On the other hand, nothing in particular precludes the application or deployment of the proposed system in another network context, be it an IP-based local area network or another transport layer used.

2.3.3 Composability

In traditional systems, each application is re-done from scratch. The services of tomorrow can easily be combined into new services. By combining simple services into complex and integrated new ones, substantial benefit can ensue and be charged as an added value. The integration might even be done on-the-fly upon request by a user.

2.3.4 Scalability

The underlying implementation architecture should support services of different magnitudes. From only a few users in a closed environment to massive access by potentially millions of users at a time in an open network should be supported. That reduces maintenance needs on the provider side due to a homogenous service provisioning environment. A benefit for the user is, that he will interact with the same environment independent of his application, because all his service uses in all contexts of use are similar.

2.3.5 Openness

The new services will be based upon standards. Proprietary solutions not only hamper interoperability, but lead to increased maintenance costs as well. Further, it is a trend for the future, that systems are increasingly interconnected. This will only be possible, if those systems are opened to a common communication infrastructure, specifically the Internet.

2.3.6 Manageability

Traditional management capabilities, like the ISO fault, configuration, and performance management are to be supported. [ITU97a] This ensures consistent management over different service domains and different technology providers. Existing tools and policies can be used easing the migration path.

2.3.7 Accounting and Billing

Revenue generation is the main reason for providing telecommunication services. Thus, a consistent accounting and billing framework and infrastructure must be found to integrate existing and new services.

2.3.8 Security

Consumer, provider, and producer of telecommunication applications are interested in trusted exchange of information and goods. The systems put into use must be reliable and robust. In the case of employing software agents, this pertains especially to supported security functionalities of the agent architecture and the security of agent places, where agents meet. (cf. Chapter 5)

Personal and private data of the user has to be secured against loss during communication. This includes direct financial data like credit card numbers, but other confidential data like personal preferences of the user must be protected, too. This could be done with smartcard systems or other locally trusted storage or processing devices. On the other hand, these aren't available to a mobile agent on a remote execution place.

Access to the data must be restricted to properly authenticated and authorized entities, and encompass only the data needed for the task at hand. The saved data must be protected against eavesdropping as well, so that other users of the same machine won't have access. The programming language and execution environment Java already provides some corresponding solutions. (cf. Chapter 4)

Because agents rely heavily on communication, (cf. Section 5.5.1) that must be secured against unnoticed modification and eavesdropping. Identification of entities must be provided for authentication and billing functionalities. Despite being on an open network, not every user should be able to access all functions unrestricted, hence authorization is desirable. For signing contracts, certificates and electronic signatures must be supported, accompanied by additional services like time-stamping and non-repudiation. Chapter 3 gives more details on the mechanisms to employ.

Another important point of a security infrastructure for agents is to secure the migration of agents between agent execution places. An agent should be protected against modification while migrating, and it should be able to

migrate only to trusted agent places. Protecting an agent against a malicious host would be desirable, but is, in the general case, not possible. (cf. Section 6.5)

In the view of the executing host, an agent has properties of a virus. Therefore, the host must be secured against harmful agents. Again, Java provides some corresponding solutions. (cf. Chapter 4)

2.4 Electronic Markets

In the economic science a market is defined as a mechanism by which buyers and sellers interact to determine the price and quantity of a good or service. [SN95] The technical advances in the area of networks enable the automatization of the relevant steps of a business process. Depending on the specific market, the possible automatization differs. The steps are summarized in the market transaction model, which is discussed in the following.

2.4.1 Information Phase

The information about the provisioning of some wanted good is essential to the potential buyer. A simple form of an electronic market is a collection of information services, that provide comprehensive data about products and their suppliers. This services might either be provided free of charge, or are to be paid by itself. Current examples are well-known WWW search engines.

Contrary to the described active or user-driven mechanisms are the passive distribution channels: Providers pro-actively give information to potential customers. This “active trading” is often implemented in the form of mailing lists.

Information technology can significantly reduce time and money needed in the information phase.

2.4.2 Agreement Phase

The buyer contacts potential suppliers. The goal is to define the product to be bought, including as much of its qualities as possible and reasonable. Further, the price and modalities of exchange will be set. The result of this phase is a contract mutually agreed upon.

The necessary negotiations can get very complicated, influenced by the complexity of the good and the pricing models of the suppliers. In the general case, this is very difficult to achieve in an automated way. In today's systems, this works best for well-defined goods with defined and simple qualities, like at stock exchanges. Stocks have well-defined qualities, only the price is negotiated.

Newer auction systems on the Internet allow for trading non-standard goods, where the quality can be negotiated as well, and/or that have no pre-defined market value. Today, the user has to evaluate the offer, while intelligent agent technology might be a way to automate these kinds of transactions.

User-oriented services are usually based on web forms. For business-to-business processes, EDI is a standard for data exchange. [oST96] Telematic services help to streamline internal and external business processes and to reduce transaction costs. The benefit raises with the volume of transactions handled.

2.4.3 Settlement Phase

After contract conclusion, money has to be paid. Money can be exchanged in numerous ways. Crediting and debiting a bank or credit account can be done by a partaking bank. More indirect ways of payment include different ecash-systems, either credit- or debit-based. Electronic money in the form of virtual coins is another way of money exchange. Money transactions can be either anonymous or the business partners know each other. Money exchange can be done very efficiently by computers, and is deployed in several variations.

2.4.4 Goods Delivery

After contract conclusion, the good and the money have to change hands. If hard goods are to be exchanged, traditional "real-world" delivery services are to be employed, which is another business transaction again. Linking the business processes of supplier and distributor in a supply chain offers great benefit in optimizing and reducing costs in the business processes for both parties. The benefit of the user lies in faster delivery and the possibility of package tracking.

In the case of virtual goods, computer based delivery is very appropriate. For example, electronic books, music, videos, and software can easily be distributed. The delivery method might vary, and requires different network infrastructure. If a high-quality video is to be streamed, the bandwidth of the network must be higher and isochronous transfer is needed, contrary to when a software package is downloaded. Establishing the right connection might again be another business process, or is part of the service provisioning supply chain.

If the communication channel between user and provider is bidirectional, interactive services can be provided. The user could modify the virtual "good" received, e.g. an electronic newspaper would contain only articles interesting for the subscribed user, or the camera angle of a sports broadcast can be selected.

2.5 Roles

Each actor in a business process is an instantiation of a role. An actor can instantiate several roles at the same time, and the same role can be filled concurrently by several actors. In this Section, the typical roles in the telematics domain will be introduced. Their specifc needs for security are emphasized.

Most of the roles can be deduced from the above scenario, (cf. Section 2.2) but some are only implicitly deducable. The roles, which will be explained in the following in more depth, are:

Consumer: User of a service.

Provider: Offers services.

Retailer: A special case of a provider, who makes available services of other providers, potentially by adding value due to intelligent combination of services.

Content Provider: A special case of a provider, who only serves content without supporting or wrapping services.

Connectivity Provider: Offers network access.

Platform Provider: Offers the execution environment, where instances of other roles meet and communicate.

Broker: The broker mediates information, often pertaining to the availability of services, between actors.

Arbitrator: In a security-aware environment, as neutral party in an ongoing communication protocol is charged with ensuring, that a contract conclusion or information exchange is done fairly.

Adjudicator: In a security-aware environment, a neutral party, that, after a dispute about an already completed transaction arises, acts as a judge to set things right.

These roles will now be more elaborated upon. Each role will be described in more depth. The requirements of the roles with respect to a security infrastructure for agent-based telecommunication applications will be identified.

2.5.1 Consumer

In the telecommunication market, everyone who uses a service is a consumer. This can be an end-user, a company, or an institution. The service can either be charged for or be given free of charge. Service usage might either be anonymous or the consumer may be identifiable. In reference to the above scenario, the consumer is the person who has his trip planned and the tickets booked.

The user wants his data to be protected against other roles, and against instances of his own role. Only the entities selected by the consumer, respectively only the entities necessary for service provisioning shall get his data. His communication shall not be successfully eavesdropped upon.

He wants a guarantee, that he will receive the goods or services, he has paid for. Nobody else should be able to impose him, and to receive the goods instead of him.

2.5.2 Provider

A provider is the general case of providing a service. This role is described in more detail in the next Subsections pertaining to the possible incarnations of a provider.

2.5.3 Retailer

A retailer offers services to consumers, he is the intermediary between consumer and content provider. A retailer can provision several client groups and offer multiple services, the services might be used for free or are charged, they might be anonymous or offered only to identified and authenticated actors.

For service provisioning, the retailer can integrate the services of another retailer, content provider, broker, or connectivity provider, or he can delegate tasks to them. He will then be himself a customer to the other party. The retailer can instantiate several roles at the same moment. He can, e.g., be his own content provider or connectivity provider.

In the context of the above scenario, a travel agency or a provider of flight tickets could be a retailer, because they employ the data basis of other retailers or content providers.

The retailer wants to ensure, that he gets paid correctly for his provided services. He wants to hide his data and possibly algorithms before other parties, not to loose his business secrets. In the case of non-anonymous services, he wants to ensure, that the remote party is properly authenticated and authorized to use the service.

2.5.4 Content Provider

The content provider offers services for access to its database. He supports retailers and other content providers in their service provisioning. A content provider has no direct business with the consumer. That doesn't preclude, that the content provider itself acts as a retailer as well. For example, an airline is a content provider upon relaying its flight information and booking capacities to travel offices. The airline might sell its tickets to the consumer as well, therupon acting as a retailer.

The data of the content provider must be protected against loss to unauthorized parties. Its protocols must be protected against impersonation and modification. Again, the services provided must be payed for.

2.5.5 Connectivity Provider

This is the manager of a transport network, be it switches, cross-connects, bridges, routers, etc. Normally positioned at the interface between content provider and retailer, they are entitled to establish connections between arbitrary nodes in the network. Because no single one connectivity provider connects all nodes of a network, this is normally done in close relationship to other connectivitiy providers connecting their respective subnets to a complete end-to-end link. The established connection often has to fulfill certain quality of service requirements. This parameter negotiation is a business process in itself. [Töb99]

Normally, one of the required quality of service parameters pertains to security. Authentication, integrity, and protection against eavesdropping are typically needed aspects. This is the domain of virtual private networks, not covered in detail in this work.

2.5.6 Platform Provider

A platform provider has the technical equipment to execute the programs of other roles. This is a set of hardware, and an execution environment.

It is mandatory, that entities executing on the platform do not impair the execution of programs belonging to other entities. No entity is allowed to have access to any other entity, neither its code nor data nor its very existence, except through well defined and mediated interfaces.

It is desirable to protect the executing entities against malicious hosts as well, but this is very hard to achieve, because the host must have access to the entity to execute it, or the host might modify or ignore the entity at will. Protecting against other platforms except the local one, running the code under discussion, is possible, though.

2.5.7 Broker

A broker informs entities about the location of and access to services. This is particularly important for the consumer role, so that the customer can find the needed services. It is important for a provider to locate the consumer, if the service provisioning is done for a mobile or roaming consumer. Again, the broker can impersonate other roles as well. A broker is in fact a specialization of a retailer, but reduced to service and entity location.

In the above scenario, a broker would mediate the access to retailers and content providers offering booking services, e.g. airlines, hotels, and booking agencies.

The information itself is the main business value of a retailer. It must be protected especially well against loss to third parties. The broker must get his money for the service provided. A broker should be unbiased in its given information, else his value to the other roles will shrink. Though, if he is bound by contract to mediate a certain provider, he must do so.

2.5.8 Arbitrator

An arbitrator is a disinterested third party trusted by the other involved parties to complete a protocol. Disinterested means that the arbitrator has no vested interest in the protocol and no particular allegiance to any of the parties involved. Trusted means that all people involved in the protocol accept what he says as true, what he does as correct, and that he will complete his part of the protocol. Arbitrators can help complete protocols between two mutually distrustful parties. [Sch96]

In the above scenario, an arbitrator could be involved to guarantee the booking, the modes of the contract, and processing of the bill.

The arbitrator is a very sensitive entitiy in the system. It is the single most point of failure in a protocol, because both parties expect it to complete its part of the process. Integrity, authentication, enciphering, and non-repudiation are strong requirements for this role. Its database is to be protected to prevent data sniffing and profiling.

2.5.9 Adjudicator

The adjudicator has the same properties as an arbitrator. Although, this role is activated not within the protocol, but after its completion, and only if one of the parties suspect illegitimate action by the other. The adjudicator then has to decide, which party of the protocol hasn't fulfilled its obligation. This decision will involve contacting the arbitrator for the correctness of the presented proof of evidence.

The security requirements are similar to the arbitrator, while its non-reachability is not critical to the already finished protocol.

2.6 Law

All cryptography-related discussions are held in the context of applicable law. The first aspect to be considered is the relationship between cryptography and ecommerce, repectively the necessary prerequisites to be established in law for providing a reliable framework for conducting electronic business.

But there is more in cryptography in this field. Algorithms, protocols, and systems are patented and thus protected by patent law issues. Because strong cryptography enables people to hide information, military and government departments have an interest against widespread use of cryptography. This finally leads to a political aspect as well.

2.6.1 Ecommerce

It is often mentioned, e.g. in [Ern01], that the upcoming ecommerce boom will only be successful, if the customer has the impression of a secured business process. This, on one hand, pertains to the technical aspects as mentioned so far in Chapter 3. On the other hand, there are important legal considerations involved. Is an electronic contract binding? How is it to be enforced? What, if the good doesn't arrive, but the money has already been paid? What to do, if the good is damaged?

In Germany, there is a strong support in politics and the responsible justice departments for ecommerce. Germany was one of the first countries to establish digital signatures (cf. Section 3.7.4) as a binding signature by law, in

the respective German law. [Bun97] Further, there is a catalogue of measures that describes, how the law is technically to be implemented. [Reg97]

Further accomodations are in work, to adapt other laws to ecommerce. The "Bürgerliches Gesetzbuch," [Rei96] (civil law) will be adapted in the near future in §126 BGB to anchor digital signatures as equally binding as written signatures currently are. [Reg00]

In Europe, national laws concerning warranty have to be significantly adapted until 2002. Up to then, upon a flaw in a good, the customer had to prove the flaw, and that he handled the object carefully, and didn't harm it. Since 2002, the seller of a good is responsible to prove, that the good had been error-free at the time of purchase. Further, the period of complaining will be significantly enhanced from six months to two years. [Möc99]

Upon the growing volume of electronically conducted commerce, tax laws are considered, so that the government will participate in the profits. There is no final decision yet, how to implement such a tax.

The legal considerations get extremly complicated if one considers the nature of the Internet as an *international* web of nodes. Then, e.g., a customer might be in Germany, the server providing the relevant information for the buy is in Korea, the server handling the money in South Africa, the clearing institution in Swiss, the content server in Russia, and the business partner has its legal residence in Brazil. How can such a buy be regulated by national law?

This is an ongoing point of discussion. Some parties assume, that there is sufficient law in place to handle such a situation, because each and every involved component has some physical aspect, and that aspect can be dealt with under conventional law. Though, there are open questions: What, if the respective countries' laws are contradictory? German law specifies, a jurisdictional controversy has to be conducted at the city of the buyer; a South African content provider (and/or the remote court) might think otherwise, or simply refrain from appearing.

One could try to avoid such a situation, but this is neither easy nor reliable. The World Wide Web gives no location information to the user. Knowing with whom one conducts business in the WWW is a "service" provided by the provider, not an information gained from the system *per se.* That provider might have an interest not to provide too much details, or is conducting fraud by making false claims.

Copying digital goods is easy. Therefore, copyright infringement is another

aspect discussed in ecommerce, though applicable only to exchanged digital goods, physical goods are not concerned. Technology for copyright enforcement is handled on the application or content layer of an exchange and is not a matter of this work.

2.6.2 Patent Law and Trademarks

Software patents are problematic in the ecommerce area. First, they might induce uncertainty in the legal status of to-be-deployed or installed systems, thus might block innovation and growth. Second, they might lead to concentration of technology in few hands, fostering monopolistic structures and hampering competition. As an example, British Telecom had patented the concept of a hyperlink in the USA in 1989. [Sar89] If this patent is enforced and licensed for significant fees, the WWW might come to a sudden end. [Die00b]

A lot of algorithms and protocols currently in use in the cryptography area are protected by patents. That patents might be local to some countries, or be applicable world-wide. For example, the IDEA symmetric encryption algorithm is only usable on a royalty-free basis for non-commercial applications. The very popular asymmetric algorithm RSA was protected in the USA, but nowhere else. That license expired on September 20, 2000.

There is currently an ongoing debate about if software can be patented "as such." In the USA and Japan, this is the case without restrictions. In Europe, under current legislation, software is not viable for a software patent. The European Patent Oranization has voted for unrestricted software patents, though. [Org99, Die00a] The European Commission has declared its intention against it. [Die00c] It should be noted with care, how the future in this area evolves.

Before using any specific algorithm, protocol, system, or implementation one has to assert, that no patents are infringed. As a result of such an audit, certain parts of the system can't be used. The conclusion is, a potential software system must be adaptable to a newly arising situation.

2.6.3 Export Law

The export of cryptography has long been restricted. Evidently, governments have wanted to avoid strong cryptography from falling into the hands of foreign powers, which would have thwarted their ability to compile intelligence. Cryptography has long been regarded as a weapon, and it has featured on lists controlling the export of munitions.

The export restriction pertain mostly to encryption and decryption technology. (cf. Chapter 3) Digital signatures, hashes, and other such algorithms are normally exempt from the regulations.

In the following, due to the more widespread implementation, *export* laws are discussed. There are some countries, though, that impose *import* restrictions on cryptography. [Koo00] Account has to be taken for that by software systems employing mobile cryptographic code, as is the case in mobile agent systems.

In July 1996, 31 countries signed the *Wassenaar Arrangement on Export Controls for Conventional Arms and Dual-Use Goods and Technologies.* The Wassenaar Arrangement controls the export of weapons and of dual-use goods, that is, goods that can be used both for a military and for a civil purpose; cryptography is such a dual-use good. The *General Software Note* excepted mass-market and public-domain crypto software from the controls.

Negotiations in 1998 resulted in restrictions on the General Software Note and in some relexations. The *status quo* is:

- free for export are all symmertic crypto products of up to 56 bits, all asymmetric crypto products of up to 512 bits, and all subgroup-based crypto products (including elliptic curve) of up to 112 bits;
- mass-market symmetric crypto software and hardware of up to 64 bits are free for export (this limit applies until December 3, 2000, by which date a new agreement should have been reached);
- the export of products that use encryption to protect intellectual property (such as DVDs) is relaxed;
- export of all other crypto still requires a license.

The United States of America has restricted cryptography export by the more severe *International Traffic in Arms Regulation* (ITAR). [otOoDTC99] ITAR

restricted export of dual-use cryptography by placing it on the Munitions List and treated it like military weapons' ammunition.

At the end of 1996, cryptography export was transferred to the *Export Administration Regulations of the Department of Commerce* (BXA). The export policy was relaxed. Even new regulations were again published on January 12, 2000. The major components of the updated policy, being the *status quo,* are the following: [Koo00]

- Any crypto of any key length can be exported under a license exception, after a technical review, to non-government end users in any country except "terrorist countries." Exports to governments can be approved under a license.
- Retail crypto (i.e., crypto which does not require substantial support and is sold in tangible form through retail outlets, or which has been specifically designed for individual consumer use) of any key length can, after a technical review, be exported to any recipient in "non-terrorist countries."
- Unrestricted crypto source code like most "open source" software and publicly available commercial source code like "community source" code can be exported to any end-user under a license exception without a technical review. BXA must be given a copy or the URL of the source code. All other source code can be exported under license exception after a technial review to any non-government end-user. One may not, however, knowingly export source code to a "terrorist country," although source code may be posted on the WWW for downloading without the poster having to check whether it is downloaded from a "terrorist country."
- Any crypto can be (re-)exported to foreign subsidiaries of US firms without a technical review. Foreign nationals working in the US no longer require an export license to work for US firms on encryption.
- The regulations implement the December 1998 Wassenaar changes to the General Software Note.
- Post-export reporting is required for exporting certain products above 64 bit to non-US entities.

2.6.4 Law Enforcement

Despite the opposing advise of cryptography experts,[AAB+98] there have been numerous attempts in the mid-90's to establish key escrow schemes by law. Key escrow refers to usage of keys as secrets for encrypting communication; government; military, and other security institutions want to gain access to encrypted data streams by having access to the secret key used for achieving confidentiality. None of the respective products have achieved any significant market share. There are several security weaknesses as well. [Sch96]

In 1997, the German Minister of the Interior Kanther made an effort to establish key escrow in Germany. Only two months later, he had changed his mind, the issues was postponed. With the new government elected shortly after, the issue was dropped. [Möl97]

Within the same year, the European Commission has chosen a direction away from key recovery. The publicized document aims at creating a reliable European framework for digital signatures. It also addresses policy regarding confidentiality. It stresses the economic and social importance of cryptography. The Commission is concerned that restrictions on encryption affect the right to privacy, its effective exercise and the harmonisation of data protection laws in the Internal Market. Also, "divergence between regulatory schemes might result in obstacles to the functioning of the Internal Market." [Com97]

After numerous outcries and campaigns of public organizations, the key escrow efforts "look like to die a slow and quiet death, but standards have a way of creeping up on you." [Sch96]

As an example for this "creeping up," in Germany §88 und §90 of the law regulating telecommunication [Bun96] in conjunction with the respective executive order [Bun95] define, that it is required for the provider of a telecommunication system to offer the technical means for surveillance of phone to the appointed governmental authorities as well as he has to provide customer's data in an automated way to said authorities.

In other countries, things are different, though. There still exist restrictions on the domestic use of cryptography in several countries. [Koo00] A software system must be designed to take that restrictions into account.

2.7 Summary

This Chapter has named the requirements of future telematic services. One possibility to fulfill these requirements is the introduction of intelligent agent technology into the telematics domain.

Now follows Part II that gives an introduction of the technology as will be used for the framework to be developed. Chapter 3 will delve into the domain of security and clear up the security terms already introduced in this Chapter. Algorithms and protocols are shown, and security and attack categorizations are given that today's systems have to withstand.

The then following Chapter 4 on the programming language and execution environment Java will discuss the programming context to be used in the finished system, highlighting its use in security-related programming.

Software agents are expected to solve several problems in the domain of distributed problem solving based on open networks, as it is the needed case in the presented typical telematic scenario here. Chapter 5 will introduce software agents and shows software agents as a possible way for fulfilling the demands of the problem domain.

Part II

Technology

Chapter 3
Security Technology

"Trust me,
I know what I'm doing.
bang *ouch*"

Sledge Hammer

3.1 Synopsis

This Chapter gives an introduction into the current state of the art in the area of security technology. The contents is often based on [Sch96].

The text is structured by the level of complexity of the respective topic. The terminology of the area of expertise is introduced first. Before then going into the technical details, some general concepts of the area of expertise are introduced, and some common misunderstandings will be cleared up. Subsequently, the technical discours starts with basic techniques. Due to the complexity of the nature, they are followed by separate Sections dealing with symmetric encryption and asymmetric encryption. Thereupon builds the discussion of security systems by combining the basic techniques. Afterwards, the SSL protocol as an implementation of those systems and as the standard for secured data exchange is discussed.

The Chapter concludes with a Section on system security based on the Rainbow Book Series and the description and categorization of several forms of attacks a system is exposed to.

The mathematical and theoretical computer science background, playing an important role in the workings of cryptography, are left out due to space constraints. The reader should be aware of the foundations of complexity theory and underlying hardware operations.

3.2 Terminology

Cryptography is the science of keeping messages secure. People who work on that field are called *cryptographers*. Trying to break the security of a message is called *cryptanalysis*, and people who do so are *cryptanalysts*. The field of cryptography and cryptanalysis together form the expertise of *cryptology*, being done by *cryptologists*. Although [Sch96] and [otA90] restrict the meaning of these words to encryption and decryption (see below), within this paper the meaning is broadened to include all aspects of securing data storage and communication.

Saying to keep a message "secure" depends on the circumstances and the respective requirements. Within the context of this work, it pertains to make sure, that the wanted attributes for a communication as given in this Chapter are fulfilled.

Encrypting a communicated message hides its contents, so that third parties, that don't have access to a certain secret, i.e. the key, are not able to semantically understand the contents. The reverse operation of restoring the original contents is called *decryption*. The less common, but in [Int89] "officially standardized" names for these operations are *enciphering* and *deciphering*. These terms are synonymous. The class of functions for en- and/or decrypting is called a *cipher*.

Steganography is the science of hiding the very existence of data. This includes hiding large amounts of stored or communicated data in other data. Another aspect of steganography is the concealment of performed communication, or the hiding of communicating partners. Steganography is not further elaborated on in this paper.

Upon communication, data is sent from a *sender* to a *receiver*. Sometimes, more parties are involved in a communication. To distinguish between parties and for easy handling, they are named with "real life names" like "Alice," who is normaly the initiator,"Bob," the responder, "Carol," and "Dave" for more participants in arbitrary roles. "Eve" is commonly used for an eavesdropper,

"Malice" is a malicious active attacker. "Trent" is the common name for a trusted arbitrator, "Peggy" tries to prove something, and "Victor" verifies that.

Within more mathematical expressions, M normally denotes a message, P is a plaintext, also knows as cleartext, and C the ciphertext, i.e. encrypted data. E is an encryption function, D the corresponding decrypting function; S is a signatory function and V the corresponding verification function. An index K denotes an indexed function, this is usually the application of a key to an en- or decrypting function. Sometimes, the key is given as a party, which is to be understood as the applicable key of that party, for this function.

E.g., reading from the inner expression outwards, $V_A(D_B(E_B(S_A(M))))$ means, Alice signs a message M with her *private* key A, then encrypts the signed data with the *public* key B of Bob, who afterwards decrypts this data with his *private* key B, and then verifies the message with the *public* key A of Alice.

The use of "$\oplus$" denotes the mathematical exclusive-or operation.

3.2.1 Security in German

In the English language, the semantic concept of computer and software security is intuitively clear, and pertains to the protection of a system against willfull attacks of intruders.

Upon using the German language, the meaning is easily confused, because of several semantics of the common translation "Datensicherheit," translated to English as an aspect of "safety." There are instances, though, where safety leads to security. (cf. Section 4.2)

Within this work, "security" is limited to the above strict english meaning. It does not cover strategies against accidental data loss, like backups or CRC codes, correctly translated as "Datensicherung." Neither does the notion of "security" in this document cover the aspect of uncertainty in knowledge. [Sch97]

3.3 General Concepts

This section introduces some general ideas and widely adopted good practices in the field of computer security. By doing so, common pitfalls are shown and alleviated.

3.3.1 Code and Cipher

The distinction between *Code* and *Cipher* is often blurred, but has to be cleared up. A *code* refers to a cryptosystem that deals with linguistic units: words, phrases, sentences. [Sch96] *Ciphers* deal with the characters of a message, at the syntactical instead of the semantical level.

This reduces usage of codes to the use in environments, where for every semantic item to be communicated there exists a referring code-word. The most prominent and almost exclusive use of codes is in the military sector, where e.g. "Barbarossa" in 2nd world was the German's code-word for the military operation of invading Russia. Today, the military shifts to ciphers as well, because flexibility is gained, but nothing lost: codes can be encrypted with ciphers as well.

In an open environment like the Internet, where it is essential to communicate about almost everything and especially about non-predetermined contents without changing the underlying technical system, encryption using codes is not reasonable. Therefore, within the open Internet community, only ciphers are used, as will be in this paper.

3.3.2 Security by Obscurity

A lot of security technology vendors rely on not opening their algorithms or devices to the public, neither the specification nor the source code base. This is commonly called "Security by Obscurity."

The "a closed system is more secure"-view is rather naïve: In the open market, where products are handed over to the customer, nothing can be hidden from a dedicated attacker. The story of computer security shows a lot of failed attempts on securing products by not revealing their secrets.

Either tamper-resistant devices or physical barriers would be needed to reduce the risk of these attacks. First, these approaches are not feasible on

the consumer mass-market. Products must be widely available, early on the market, and cheap, if they are to be successful. They must be handed over to the customer.

Second, even then, attacks are very possible and have often occured: break-ins in secured building complexes are commonplace. Attacks on non-disclosed material can be done by observation, both of the expected and wanted behavior, and of side-effects not covered by the specification of the system, and hence they are neglected by the implementation. The differential power analysis [KJJ99] and timing attacks [Koc95] on SmartCards are examples of successful attacks by the second type of observation.

The common denominator of the security professionals therefore is to have as much qualified peer review and analysis of their cryptosystems as possible. Cracking contests are sometimes used to attract the attention of cryptographers, but are normally ignored for more interesting tasks. Thus, their results can not be used as a "proof" of the security or quality of a system. Not finding an error does not mean, there is none. Even worse, the contests themselves often rely on the "Security by Obscurity" principle, important information is non-disclosed, making the attempt to hack the system more difficult, but leading to a false sense of security. [Sch98]

Futile are "security concepts," that rely on the good will of the rest of the world to function. As Bruce Schneier said: "its just a matter of time before someone with the right combination of resources and ethics comes along." [Sch96]

Another problem with systems not open to public scrutiny is the very real existence of backdoors. As for example can be seen in [Yah01], in "well-known" software even of large companies code is found, that opens up unwanted access to the customers' machines, be it by malicious individuals, by company policy, or by other unnamed influence groups. As a general rule, the "a little paranoia never hurts" [Pro01] paradigm sounds odd, but has its applicability.

3.3.3 Old means Trusted

The cryptology community has a somewhat conservative attitude. New algorithms, protocols, and cryptosystems aren't embraced lightly. [Sch99a] explains by similarity and by example, why this is the case.

The bottom line is, one should prefer well-proven systems over newly-invented ones. Nevertheless, *perfect* security is not possible. One can only try to reduce the risk to a minimum, but a residual risk will always remain.

3.3.4 Snake Oil

Obviously, in choosing a security product a user has to consider its security. The security of a product depends on several factors. The algorithm used is normally of the least concern, because there is a number of well-established mechanisms available, all well-proven as discussed above.

The pitfalls normally lie in the implemention of the system, which is very often extremly fragile with respect to various implementation details, e.g. the correct mode of an algorithm used. (cf. Section 3.5.3) Things get even worse, if the vendor implements its own algorithms or protocols.

In the commercial world, it is rather uncommon for a vendor to put its product under an open source license. Thus, the potential buyer and user of the software has no opportunity to look at the software first-hand, or to have it examined by experts. Rather, his decision on trusting the software has to rely on the information he gets from the supplier.

The advertisements of vendors often help to distinguish between good products and bad products, because there exist obvious warning signs for software to avoid. This is exemplified and described for encryption algorithms in the "Snake Oil FAQ." [Cur98] They are summarized in [Sch99b] as follows:

Pseudo-mathematical gobbledygook: (also known as technobabble) If the contents of a system's description appears to be nonsense, it often is.

New mathematics: If an algorithm is newly invented, there often are new security holes as well, which are discovered only shortly later.

Proprietary cryptography: If the company refuses to open their technology to public scrutiny, they are very suspect of having something to hide. The security of an encryption algorithm, for example, lies in the non-disclosure of the keys used, and not in the knowledge about the algorithm itself. The distributed code, and hence the algorithm it implements, is subject to reverse engineering anyway.

Extreme cluelessness: If it appears that a company doesn't know, what its talking about, it is very likely that it doesn't. This point is closely related to technobabble, as described above.

Ridiculous key length: If a vendor is proud of "overkill" attributes of their product, it is a hint that something other more important is missing. This point might be an expression of the previous warning sign.

Unsubstantiated claims: If the vendor makes claims of world-wide acceptance of the security or technology of their product, than there normally isn't.

Security proofs: These warning signs are divided in two categories. Either the system is "proven" by explaining some well-known knowledge, and then acclaiming that the product inhibts these properties, or there is a real mathematical proof, which isn't directly related to the product.

Cracking contests: Often, vendors encourage the public to try to break an algorithm. Out of the fact that no-one succeeded they infer that their algorithm is secure. This is more closely analysed in [Sch98] and [Spa95].

[Sch99b] concludes with:

> These snake-oil warning signs are neither necessary nor sufficient criteria for separating the good cryptography from the snake oil. Just as there could be insecure products that don't trigger any of these [...] warning signs, there could be secure products that look very much like snake oil. But most people don't have the time, patience, or expertise to perform the kind of analysis necessary to make an educated determination. [...] the only thing a reasonable person can do is to use warning signs like these as guides.

This effectively means, that one can not be sure of a product used. Instead, one has to build up some trust in the product by relaying to the expertise of others whom one trusts. It is therefore not possible to attain perfect security with a proprietary product.

3.4 Basic Techniques

This section covers some basic technical aspects of computer security. It covers methods used as building blocks in the other algorithms and protocols as described in further Sections.

3.4.1 Random Numbers

Random numbers form the basis for a lot of cryptographic methods. The unpredictability of e.g. key material used in encryption is fundamental. Random numbers are used as secrets between two parties for denying a third party access to the information communicated. Therefore, the random numbers used for cryptography must be "good."

There exist several categories of random numbers: The first category is called *pseudo random numbers.* They appear to be random, and pass all statistical tests of randomness, like the χ^2-test. Unfortunately, they have a period and thus are predictable after a certain amount of randomness they generate. A discernable period, though, makes applying the data to cryptography worthless, because then, statistical attacks on the material are viable. Further, on any computer under the same circumstances, the generated sequence is the same. This is not sufficiently random.

Cryptographically secure pseudo-random numbers are, in addition to the above properties, non-predictable, even if full knowledge of the algorithm and any sequence-related data is given. This is a demand on the random number generating algorithm, which must exhibit a sufficient amount of entropy. Still, there is a problem with cycles, but good algorithms have a period of 2^{256} and more. [Sch96] Hashes (cf. Section 3.4.3) and encryption functions, (cf. Section 3.5) due to their chaotic behavior and large periods, are often employed as random number generators.

A cryptographically secure pseudo-random number generator still produces the same sequence of numbers, if it is being fed the same initial state. *Real random numbers* have the property of being non-reproducable. For a real random number generator, the same initial input, if at all possible to provide, yields different resulting random number streams. For these purposes, specialized hardware is utilized. Commonly, radioactive decay is measured, filtered, and converted to a random number stream. The radioactive decay is, for human means and technology, neither reproducable nor predictable.

[ECS94] gives in-depth thoughts about the generation of randomness on computer systems.

3.4.2 Keys

The security of a message being worked on with a proven cryptographic function relies entirely on the secrecy and quality of the key material used.

Key Length

The keyspace should be large enough to ward off brute-force attacks, (cf. Section 3.10.2) but longer keys lead to slower processing. The DES algorithm [oST99] with its design restriction to 56 bit can not be considered secure any more, even less the variant of varying only 40 bit thereof. It is vulnerable to brute force attacks. Today's standard is to have 128 bit of symmetric key length. (cf. Section 3.5) For RSA keys, 1024 bit are defined as to be used in several signature standards, [Reg97, oST00] but seriously security concerned experts recommend 2048 bit. [Sch96] Some programms allow to use 4096 or even 8192 bit as key length.

Randomness

The keys must be as random as possible, (cf. Section 3.4.1) so that no algorithm or statistical analysis might be able to predict them.

The probability of an adversary succeeding at guessing a random key must be made acceptably low, depending on the particular application. The size of the space the adversary must search is related to the amount of information present in the key. "Information" here pertains to the information theoretic sense as defined in [Sha63]. This *entropy* of a message depends on the number of different secret values possible and the specific probability of each value, and is measured in bit.

Flat Key Space

A *weak key* of a symmetric algorithm leads back to the plaintext, if applied twice: $E_K(E_K(M)) = M$. *Semiweak keys* are pairs of two keys K_1, K_2, where $D_{K_2}(E_{K_1}(M)) = M$. *Possibly weak keys* restrict the way an algorithm works on the keys, and theoretically lead to easier breakage, but it is unknown how to exploit this vulnerability in reality.

Since there are so few weak keys in proportion to the keyspace, there exist only an infinitesimal small chance of picking one at random. Within the

cryptographer's community it is debated, if it is worth the effort to test for them upon key generation. [RSA00] On the other hand, the test itself isn't much effort either, so for serious applications or a paranoid user the test should be performed. [Sch96]

It is the design goal of a cryptographer, to build algorithms with as few weak keys as possible. The DES algorithm [oST99] has only four weak and twelve semiweak keys. It has 64 possibly weak keys, out of its keyspace of 2^{56} keys. [Sch96]

If all keys are of the same strength for the algorithm, the key space is said to be *linear* or *flat*.

Total Keyspace

Not only should each generated key have the same probability to be generated than each other generated key, but the *whole* keyspace should be used with equal probability. This seemingly simple requirement is important for two often encountered issues.

Implementation errors often lead to a non-exhaustive key usage, where whole classes of keys will never be generated. This weakens the resulting system, because it reduces the effort for a analysis and brute force attacks.

Second, due to export regulations of some countries require a maximum key length to be used for encryption, so that their executive institutions will be able to decrypt secret messages. (cf. Section 2.6) Therefore, in some products part of the keys will be set to predefined values, for example setting 40 bit of a "128 bit" key to a true random value, the remaining 88 bit to all zeroes. This reduces the effective keyspace to be searched in a brute force attack from 2^{128} to 2^{40} and makes the attack feasible.

Key Storage

Having a good key is not enough. It is required, that the key does not fall into wrong hands. This is a task harder than it sounds. Even on a single computer system, the key will end somewhere on the hard disk. There it might be accessed by other processes, possibly by other users, and might even get send over a network accidentally. A comprehensive security system must provide means to protect keys against getting lost. This includes never saving keys willfully on a disk, using SmartCards for non-retrieval key storage, using

operating system means to prevent memory areas holding keys being swapped to disk, encrypting keys before being saved, etc.

The standards [RSA93d], [RSA93f], and [RSA97] together define a way, how to store a secret locally in an cryptographically secured form on disks. The method is strong enough for transporting the data store over a network to a remote machine.

For secret key sharing systems, as a requirement for the system to work, at least two parties have to have access to shared and secret key material. This explodes the chances for mishaps and keys gotten lost. A system, if possible, should be designed not to require shared long-term secrets.

Duration of Use

The longer a key is used, the more amount of data can be collected to be used for key retrieval. For high-traffic communication, symmetric keys should be changed more often than for low-traffic data exchange. Further, the more valuable an information is, the more often the key should be changed. The disadvantage is, each new key exchange adds latency time to the communication.

Using ephemeral keys for just the session at hand reduces the valid lifetime and therefore the damage possibly incurred with it. Wherever possible, temporary keys should be used in exchange for long-term keys. Long-term keys must never leave the system they are on.

Key Compromise

In practise, keys get lost, broken, or expired for different reasons. Therefore, the revocation of keys must be provided for. This is no problem for ephemeral keys, because the connection they pertain to can simply be closed. If the protected connection provides for *perfect forward secrecy,* this does not have any security implications for other secured communications. For long-time keys, there must be a corresponding mechanism and infrastructure installed. (cf. Section 3.7.5)

3.4.3 Hashes

Be there a function H for a message M that results in a hash value h of constant length: $h = H(M)$. H is called a hash function, if and only if

- for a given M, h is easily calculated, and
- for a given h, it is a "hard" problem to find an M with the property $h = H(M)$, and
- for a given M, it is a "hard" problem to find an M' with the property $H(M) = H(M')$.

Further, it is very desirable for H to be collision-free. That means, it is a "hard" problem to find two random messages M and M', so that $H(M) = H(M')$. The term "hard" here and in the above list refers to its complexity class. This property is directly targeted against the birthday paradox problem, i.e. that it is significantly easier to find two random pairs of the same number, than it is to find the corresponding number for a given one. There is often a viable form of attack against a cryptosystem.

Unfortunately, collisions for widely-adopted and well-known hash functions were discovered: RIPE-MD [DBP96] was discussed in [Dob97], MD4 in [Gol96]. MD5 [Riv92] was shown to have collisions in [dB94]. For SHA-1 [oST95] there are currently no collisions known, it appears to be the favorite of a today's hash function usage evaluation although it is encumbered wih a patent and a corresponding licensing scheme. [KBC97]

Hashes have chaotic properties, they inhibit an avalanche effect. That means, one bit in a data collection, e.g. a key or a plaintext block, effects several bit in a further operation. Hashes are used primarily for integrity checking. (cf. Section 3.7.2)

Other names of hash functions are *compression function, message digest, fingerprint, cryptographic checksum,* or *message integrity check.* Due to its nature, the result of a hash incurs information loss. Tagging a hash function as a "compression function" is insofar ambigous, as it contradicts with the compression functions of the next Subsection.

3.4.4 Compressions

A "true" compression function analyses its input data for redundancy and tries to eliminate that in contrast to a hash by exchanging the redundant data with a *recoverable* substitute. The inverse function recreates the original input stream.

Compression functions find their use in cryptography for reducing the input data volume thus speeding up the time-consuming cryptographic functions. Additionally, a compressed data stream has a significantly reduced redundancy and non-trivially predictable cleartext, which hampers attacks on the stream.

3.5 Symmetric Ciphers

Symmetric ciphers have the property, that the same key is used for en- and decrypting a message, i.e. $D_K(E_K(M)) = M$. Another class of symmetric ciphers use the same function for both en- and decrypting, but different keys, that are computable from each other: $C_{K^{-1}}(C_K(M)) = M$, where K^{-1} is the inverse of K, and C is the cipher. Figure 3.1 [Cer97] visualizes symmetric encryption, where the encrypting and the decrypting key are the same. Because the key(s) have to remain secret between the communicating parties to secure the data, this type of ciphers is also named *secret-key cryptography.*

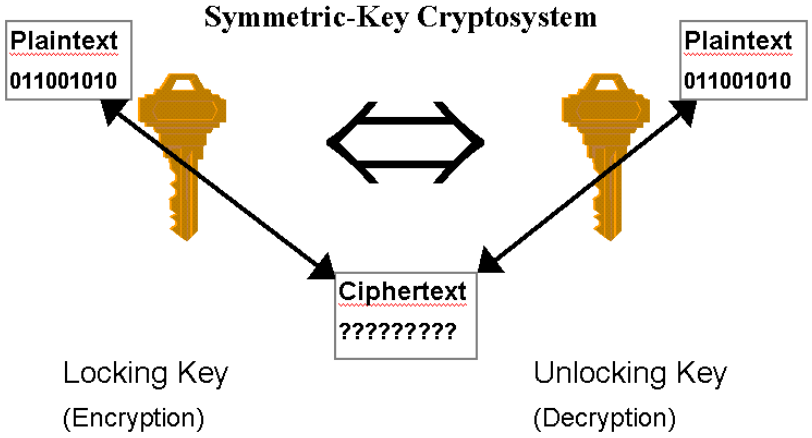

Figure 3.1: Symmetric Encryption

There is an important distinction on what data fragment a cipher operates.

This consideration has significant security implications, and the applicability of the algorithms are different, as is shown in the following.

3.5.1 Stream Ciphers

A *stream cipher* transforms one bit of a cleartext to one bit of ciphertext per operation. The bitstream of plaintext is $\oplus$-ed with a bitstream of keystream bits. For decryption, the resulting ciphertext bitstream is again $\oplus$-ed with the same key bitstream and according to the properties of an $\oplus$-operation yields the plaintext again.

The more random the generated keystream is, the better is the security of the ciphertext. The period of the stream must be, as always in cryptography, as large as possible, to thwart statistical analysis. On the other hand, the keystream must be reproducable, else decryption would not be possible.

The keystream generator must not produce the same keystream every time it is used. Instead, a key is used to initialize the keystream generator, so that only if one knows the initially used key, one is able to reproduce the same keystream. If the key is not known, the keystream is non-predictable, hence the ciphertext can't be decrypted. Thus, the security of a stream cipher lies in the size and randomness of the key as well.

3.5.2 Block Ciphers

A *block cipher* works on a fixed length of bits at a time. A typical number of bits is 32 or 64, the number often results from optimization of the algorithm for a specifc hardware architecture. The ability to make advantage of the inherent word size of a processor leads to high performance, normally faster than stream ciphers.

As a further difference to stream ciphers, a block cipher operates on several bits a time. This allows to use forward references to bits in the block within the algorithm. Avalanche effects can be implemented with less operations, due to the availability of more bit material. The result is enhanced security due to increased entropy.

Because block ciphers need to have a complete block of input before they generate a ciphertext block, they are not suitable for serial communication and other instances, where encryption bit-by-bit is needed. Where there is a

bulk of data ready at hand, a block cipher is the preferred encryption algorithm. Block ciphers are more flexible upon usage of the algorithm mode. (cf. Section 3.5.3)

The inner working of a block cipher depends heavily on the design, so there is no general image to show. Block ciphers employ a subset of the available basic techniques as shown in Section 3.4 and 3.5.

Padding

Because block ciphers require a fixed amount of data, special considerations are to be taken for the last amount of a bulk of input data. The last few bytes usually do not conform to the block boundary as imposed by the algorithm used. The short block is padded at the end with an arbitrary pattern.

If the padding is to be recognized as such after deciphering, some padding indicator is to be used. This is normally implemented by appending the amount of padding into the last few bit (usually the last byte) of the padded plaintext. Upon deciphering, the last byte is interpreted as the number of bits or bytes to eliminate from the decoded data. This semantics requires that *every* last block is to be padded, even if it is already conforming to the block length, to prevent accidental data loss due to interpretation of the counter where it shouldn't have been. As a consequence, the ciphertext will be longer than the plaintext, up to one additional block.

3.5.3 Algorithm Modes

An algorithmic mode is the combination of a cipher and some *feedback,* also named *chaining,* mechanisms applied to the data.

Further, the mode in stream ciphers is often implemented in the keystream generator and is not only hidden from the application programmer's interface, but often defined at design time and can't be changed afterwards. Because of these reasons, only block ciphers are assumed for use of modes. In principle, though, all statements are valid for stream ciphers as well.

The naïve way of applying a cipher to a sequence of blocks of data is to use the cipher for each block irrespective of any previous or following ones. This mode is called *Electronic Code Book* (ECB), because each plaintext block encrypts to the same ciphertext block, provided the same key is used. Thus one could, for each key, build a complete book showing for each input block

the resulting cipherblock. Upon getting hold of one encrypted block, the corresponding cleartext could easily be attained by mapping it back with the help of the previously generated codebook. This attack is even more feasible, because especially the start and end of messages tend to repeat: e.g., mail headers or signatures are stereotypical.

To circumvent the security problems of arbitrarily modified, inserted, and repeated blocks in ECB mode, the ciphertext blocks will be connected and depend on each other if the cipher is used in *Cipher Block Chaining* (CBC) mode, where each resulting cipher block is $\oplus$-ed with the next plaintext block before encryption.

Different bulks of data would, using the same key, encrypt to the same ciphertext up to the first difference. To circumvent this problem, and to be able to use the same encryption routine even for the first block, where there is no previous ciphertext block available, an *initialization vector* (IV) is used for $\oplus$-ing the first plaintext block with. This is a strengthening of the cipher's application, because with a sufficiently large IV, a codebook attack on a stereotypical first data block would become infeasible. Hence, an unpredictably random initialization vector should be used, but it need not be secret.

Using a block cipher in *Cipher Feedback* (CFB) mode allows to send along single bytes before the whole block is computed. For generating the next output chunk, the previous ciphertext chunk is encrypted and then $\oplus$-ed with the plaintext chunk. If only processing one bit at a time, this mode becomes undistinguishable from a stream cipher.

The *Output Feedback Mode* (OFB), also called *internal feedback,* is very similar to CFB mode. Instead, the keystream is independent of the plaintext and the ciphertext. Large amounts of key material can be pre-generated, even before the plaintext arrives, which helps securing high-speed data communication.

In some instances, it is desirable to combine the qualities of several algorithms into a composite one. For example, DES [oST99] has very good resistance properties against linear and differential cryptanlaysis. (cf. Section 3.10) Unfortunately, the key length of 56 bit makes a brute force attack very practical with today's hardware. To circumvent this, the variant Triple-DES [oST99] (also called 3DES, or TDES) has been introduced.

For encrypting a plaintext block, it is encrypted with the first key, then decrypted with the second, following encrypted again with the third key. The three keys are independent of each other and randomly determined. For

Triple-DES this results in an effective key length of $3 \cdot 56 = 168$ bit, enough to withstand brute force attacks in the forseeable future. The three rounds of encryption are combined in outer-CBC mode. Using the same algorithm repeatedly instead of combining different algorithms has the advantage of exploiting existing speedup hardware devices.

Which mode to chose depends on several factors, primarily security, speed, and access considerations. ECB mode is the weakest, but is very well suited for encrypting non-message oriented and short data, like ephemeral keys. CBC mode is the most secure mode and applicable to large bulks of data with a low chance for bit errors, as in encrypting data files on disk. CFB is adequate for character-oriented streams of data like terminals of a host system. OFB is suitable for high-speed synchronous data communcication applications.

3.5.4 One-Time Pad

The only unbreakable encryption known today is the one-time pad. It comprises of a simple $\oplus$-operation, with a completly random and non-repeating key. Then, if given a ciphertext and testing a key, every resulting plaintext is equally probable, and the original plaintext can't be identified by any statictical means.

Unfortunately, the key length has to be the same as the message length. This makes distributing the key to the communication partner as problematic as sending the message itself, thus renders the one-time pad as unpractical in an open and high-bandwith environment between identities unknown before commencing the communication.

3.6 Asymmetric Ciphers

This kind of encryption functions use two different keys for encryption and decryption, the two keys are interdependent. For an attacker it is not feasible to compute one key from the other, because all *asymmetric ciphers* are based on "hard" problems according to complexity theory. Figure 3.2 [Cer97] visualizes asymmetric encryption.

Because usually one of the keys is made public for purposes of either encryption or signature checking, the resulting system is often called *public-key*

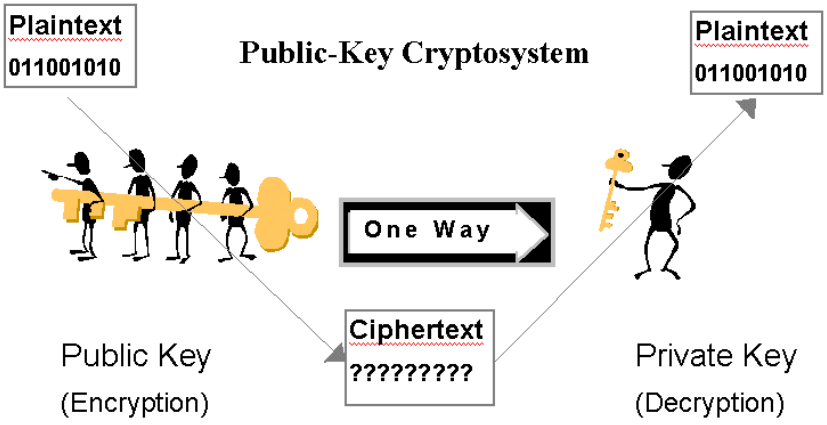

Figure 3.2: Asymmetric Encryption

system.

For encryption applications, the public key is used to encrypt a message, which can only be decrypted using the private key of the recipient. For authentication purposes, the secret key is used to sign a message; the public key is then used to ensure, that the signature has been made with the secret key.

Due to the fact, that an attacker has access to the public key of the recipient, he is able to pre-generate a codebook $E_K(P)$ for chosen P's. This is a practical attack, if P is either short or well-known, and faster to resolve than an attack on the key itself. Therefore, public key algorithms should be avoided for classes of applications with a short plaintext, or messages should be padded with random data to at least a secure length.

The factorization of large numbers into their prime components is one of the "hard" problems made use of in cryptography. The most known representative is the *RSA* cipher, betitled after the names of its inventors, and it is currently the best analysed asymmetric cipher. Because there hasn't been any serious security issues discovered since its inception, it is regarded as the most secure asymmetric cipher as well. Further, it is easy to implement, resulting in RSA being the most popular asymmetric cipher today.

Neither the insecurity nor the security of RSA has been *proven.* It is only *assumed*, that breaking the RSA keys is as difficult as factoring the numbers used in the key generation. There might well be a shortcut to factoring being undiscovered yet, putting the security of RSA at stake.

ElGamal was the first who realized using the difficulty of calculating *dis-*

crete logarithms in a finite field for encrypting and signing, [EG85] hence this mechanisms is known under his name as well, variants have been proposed by Schnorr.

The resulting ciphertext is twice the size of the plaintext, which renders it unattractive for purposes of encryption. The ElGamal scheme is well established in the area of digital signatures. (cf. Section 3.7.4)

Elliptic curves are mathematical constructions from number theory and algebraic geometry, which in recent years have found numerous applications in cryptography. An elliptic curve can be defined over any field (for example, real, rational, complex), though elliptic curves used in cryptography are mainly defined over finite fields.

ECC has several advantages over RSA. For similar security, data requirements are reduced, as is shown in [Cer99] and in [Cer00]. This makes ECC particularly attractive for data environments with strict hardware limitations, like smartcards. Further, implementations are faster for ECC than for RSA, e.g. signing up $10-40$ times. [Sch96] But, ECC cryptography is a relatively new branch of cryptology, hence the community is somewhat reluctant to embrace it. (cf. Section 3.3.3)

3.6.1 Hybrid Encryption

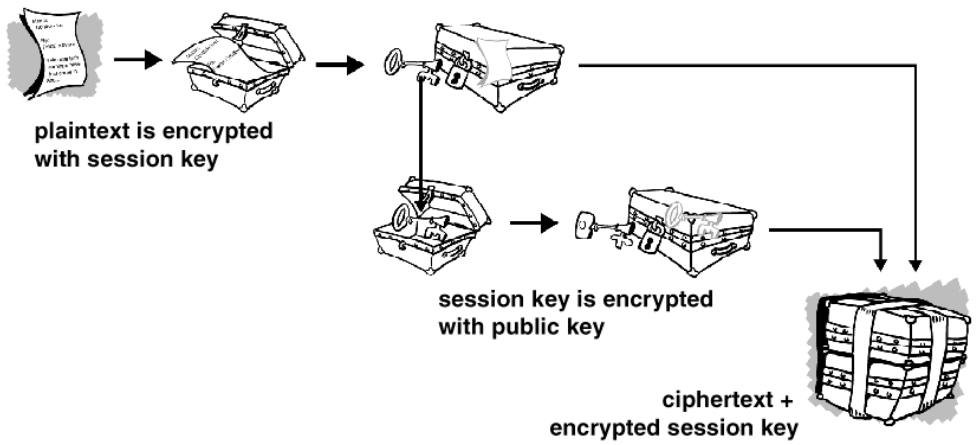

Figure 3.3: Hybrid Encryption

Asymmetric encryption is significantly slower than symmetric algorithms. Further, keys for the former are about 10 times longer for comparable security. For these reasons, one combines both encryption types in a hybrid

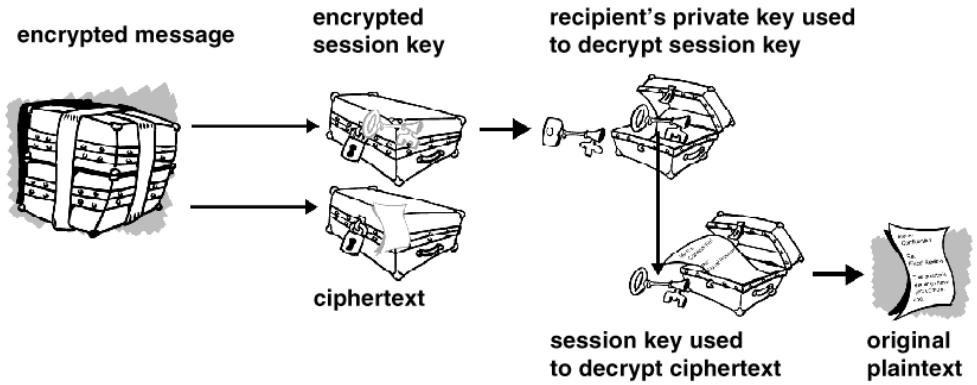

Figure 3.4: Hybrid Decryption

scheme: Beforehand, both communicating parties generate asymmetric keys and publish their public part. For a communication session to initiate, a random symmetric session key is generated. The session key is then encrypted asymmetrically and sent to the partner. From here on, the session key is used to symmetrically encrypt the communication. (cf. Figure 3.3 [Net99] and Figure 3.4 [Net99])

3.7 Security Systems

In the former Sections, several algorithmic building-blocks for cryptography applications have been introduced. To fulfill a specific purpose, though, these algorithms must be combined into a security system. The system combines algorithms into a complex synthesis to achieve a specific property. *Protocols* are used for communication of data, where a protocol is defined as a series of steps, involving two or more parties. [Sch96]

3.7.1 Key Agreement

For securing a communication against eavesdropping, the involved parties have to agree upon a shared secret key to use for encryption, if they want the benefits of symmetric ciphers as shown in Section 3.5. In the general case, and according to good practices of key use and storage as described in Section 3.4.2, an ephemeral key for the communication session at hand has to be established and exchanged, before encrypted communication can

commence. The tricky part is, that neither of the party yet has a shared secret to use for initialization, but even more a third party should not get hold of the secret-to-be-established.

Upon encrypting a communication and thus deprieving third parties of the possibility to eavesdrop on the exchanged data gives the property *confidentiality*, sometimes also called *privcay*, to the communication.

Authentication and encryption should use different asymmetric key pairs. Authentication is an orthogonal feature to confidentiality, so this is, in the first place, a cleaner system architecture. There is a stronger reason for two key pairs. If one key pair is used and it gets compromized, both encryption and authentication falls down. If two key pairs had been used, either authentication or encryption could still hold an attacker at bay.

For key generation, the currently most often employed mechanism is very similar to the ElGamal encryption scheme as introduced in Section 3.6. This scheme is named after its two inventors *Diffie and Hellman*. It has the very interesting property, that there is no other key material involved in the process, i.e. not even a public key of the other party is needed to establish the shared secret. Because each key exchange is independent of all others, the compromise of one key does not lead to compromise of other communications secured by different keys, this property is called *forward secrecy*. [RSA93c] defines technical details, how to implement the algorithm, and defines object identifiers.

If there exists a public key of the party one would like to establish a shared secret key with, that key can be used to encrypt a randomly generated key and then sent to the other party. No one else except the holder of the corresponding secret key can then decrypt the generated ephemeral key.

One must take care of not falling victim to a man-in-the-middle attack, were an attacker spoofs the identity of the recipient and issues the public key to a secret key he possesses instead of the correct one. Certificates help to thwart these attacks. (cf. Section 3.7.5) Either an infrastructure to retrieve public keys or another means of key distribution must exist to get hold of the public key.

This method of generating the shared secret on one side of the communication is faster and the corresponding communication protocol for key exchange is simpler than keys generated with the Diffie-Hellman method. In addition to being prone to a man-in-the-middle attack, it is less secure because the key is based on input of only one party. This makes bad random number generators on one side having significant impact.

3.7.2 Message Authentication Code

A special application of hashes is their use as a *message authentication code* (denoted MAC), upon using it as a MAC it is also tagged *keyed hash function*. (cf. Section 3.4.3)

To compute a MAC, a shared secret key between the communicating parties is needed, for example to be established by a Diffie-Hellman key exchange protocol completion. The key is mathematically used as an index to the hash function's application, in practise this is realized as concatenating the key with the message to be checked, and computing the hash value over the concatenated data.

This intuitive approach of generating a hash is insecure, because an attacker could add data blocks to the message, and be able to calculate and insert a valid hash for that. Therefore, the strengthened scheme of a HMAC has been introduced, where the data to be hashed is pre-processed, then hashed, then alternatively pre-processed and hashed again. If, in the following, a MAC is referenced, a HMAC is included in the semantics as well.

It is an ongoing debate in the cryptographer's community, if truncating the HMAC output is a strengthening or weakening of the mechanism. Truncation has the advantage, that an attacker has less information about the output to use for compromising the communication. On the other hand, requiring less data for integrity checking makes forging the checksum easier.

Employing a MAC in a communication enables to ensure the *integrity* of a communication. No one is able to modify the communicated data block without it being noticed. Although, this only ensures, that each specific block has not been tampered with. If the sequence of data flowing in a connection should be protected (partial sequence integrity), additional measures must be taken, as described in Section 3.7.3

Figure 3.5 [Cer97] visualizes the same process upon using public key cryptography: The computed hash is decrypted by Bob with his secret key and then sent along with the message to Alice. She checks the hash value by encrypting it with Bob's public key. The result must match with her own calculation of the hash, else the message has been tampered with. This also is an application of authentication (cf. Section 3.7.6) and digital signing. (cf. Section 3.7.4)

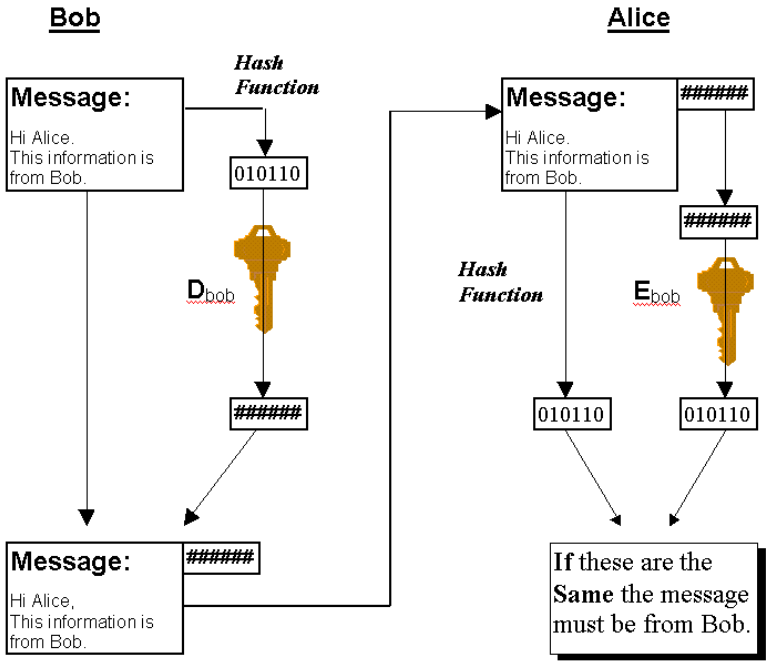

Figure 3.5: Message Integrity Check

3.7.3 Block Counter

Within each message exchanged between two parties, a message counter is inserted. The counter is incremented strictly monotonic, i.e. for each block. It is included in MAC computation to prevent it from being tampered with. A packet is only valid, if its counter is one larger than the last received. This strict form of sequence integrity is called *connection-oriented integrity*, all blocks must be received strictly in order. Lost blocks are detected (except the last, special measures upon closing the connection must be conceived there), if the increment is by a known fixed amount.

Overflow of the counter should be detected and a mutual re-initialization has to take place for continued communication; choosing the counter large enough for any reasonable data exchange can accomodate for this.

Adding a block counter to communicated data blocks ensures *replay protection*: No data blocks can be re-inserted into the communicated data stream without being noticed. Also, it is impossible to re-order data blocks.

The situation gets complicated, if data exchanged is not strictly sequential, as in the situation of receiving internet protocol datagrams out-of-order. A replay protection window can be used. A datagram is additionally still

considered valid, if its counter is not larger than the last received, and its sequence number is not older than a predefined window size, and has not been received before. [KA98] This is also known as *partial sequence integrity*.

Block counters do not protect against replay of the entire communication, including the first block. Therefore, a *nonce* is used by both parties as part of establishing the security session. A nonce is a random piece of data to be included in MAC computation or into key generation. Introduction of a nonce guarantees *liveness* of the communication. [MSST98]

3.7.4 Digital Signatures

Signatures in the "real world" have been used to express commitment to the contents of a document. This only works, because a signature has several properties:

- The signature can't be forged. No one than the signer can produce the same signature.
- The signature can't be re-used. The same signature can only be used for the signed document, it can't be attached to another without being noticed.
- The signed document can't be altered. It is not possible to later change a signed document without noticing the mismatch with the attached signature.
- The signature can't be repudiated. The signer can't claim later, he didn't sign the document.

In the "real world," none of the properties are absolute. But the scheme is practical, because there exist norms and regulations for proper signature use, and punishment for fraud.

Within the "electronic world," it is tried to re-model these properties to exchanged data using digital signatures. A digital signature is used to achieve similar properties.

Unforgeability is attained by requiring secret material by the signer. It is assumed, that the signer takes care keeping the needed material confidential. Then, only the keeper of the secret material is able to provide the signature.

The same digital signature can't be reused for another document, because it depends on the contents of the document, and signing different documents yields different electronic signatures. The inverse is, if the signed document is altered after signing, the signature no longer conforms to the document: the alteration is evident.

Non-repudiation, meaning the impossibility for the signer to later deny, that he signed the document, is harder to achieve. The signer can always claim later, his key got lost or compromised, and deny having signed a message. To achieve a limited non-repudiation of older material, a trusted arbitrator Trent is included: upon Alice signing a message M and sending the signature and the message to the recipient Bob, she instead sends $(M, S_A(M))$ to Trent. Trent affixes a timestamp t to the message, and signs the combination himself, forwarding $((M, S_A(M)), t, S_T((M, S_A(M), t)))$ to Bob. Bob can check, that Trent signed the message and timestamp, and he can check Alice's signature. If Alice later claims, her key got lost, still all timestamped communications before that event will be regarded to be valid and signed by Alice, not an attacker. Having a timestamp in the communication has the additional advantage, that all recipients can recognize the same document being sent either again, or newly sent and timestamped: for example, this makes repeated presentation of the same electronic check to a bank impossible.

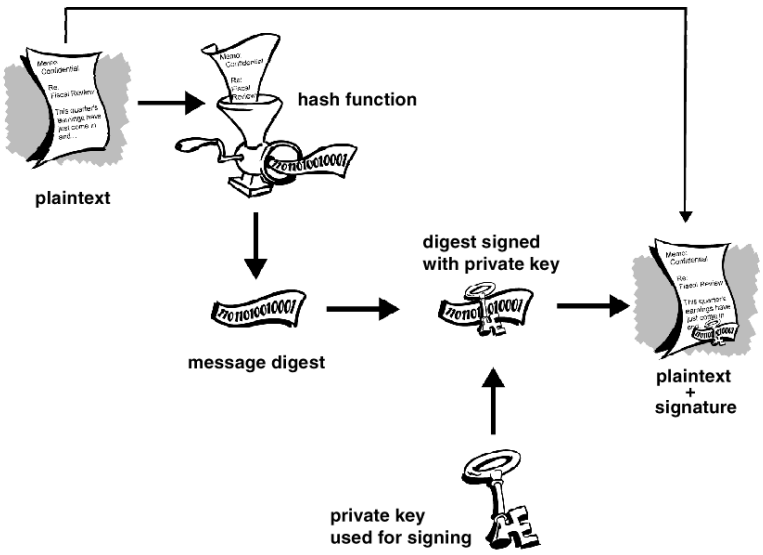

Figure 3.6: Digital Signature

Signing a document using a public key pair is the inverted application of encryption. (cf. Section 3.6) A message is first hashed and then encrypted by the sending Alice with her private key and decrypted by the recipient with Alice's public key. (cf. Figure 3.6) [Net99] If the decryption works, it is the proof, that the encryption has been performed by Alice with her secret key. In fact, calling the signing process "encryption" and the verifying process "decryption" is only valid, if RSA is used as the algorithm. (cf. Section 3.6) [RSA93a] defines, how to do that, and defines common object identifiers.

The *Digital Signature Standard* (DSS) is a worldwide adopted standard for digital signatures. [oST00] It provides two functions, one for signing and one for verifying the signature. The corresponding Digital Signature Algorithm (DSA) is based on the ElGamal scheme. (cf. Section 3.6)

[oST00] allows to use RSA or Elliptic Curves as alternatives for signing and verification algorithms. Signature verification using DSA takes about half as long as signing. DSA signing is up to 40 times slower than with RSA. If used with a 1,204 bit key, it is assumed to be strong enough for long-term security. [Sch96] A second keypair and a different algorithm is used for encryption purposes.

Because the DSA has been developed by the National Security Agency (NSA) of the USA, and weaknesses in the form of subliminal channels that can be used to leak information about the secret key have been detected, DSA is not applicable for deeply suspicious people.

3.7.5 Certificates

In all public-key systems, the danger of a man-in-the-middle attack lingers: an attacker could substitute a public key chosen by him, of which he has the corresponding secret key, and provide this to the sender to be used for encryption.

To circumvent this issue, public-key certificates (only "certificates" for short) have been introduced. A certificate (more precise: a *user-certificate*) is the association between an *identity* and its public key, affirmed (i.e. signed) by a trusted party. The identity of an entity is a recognizable and unique *name* for the entity. There must be a syntax to express an identity.

A very common representation for identities is defined in the ISO standard X.501 [ITU93] called a *distinguished name* (DN) and is effectively a sequence of pairs of an attribute and a value. Each of these pairs is called a *relative distinguished names* (RDN), the structure is shown in Figure 3.7. A

DN uniquely identifies the corresponding entity. To exemplify the structure imposed, and the conceptual integration of RDN and DN, Table 3.1 gives an example for the construction of the DN out of several RDNs. The certificate itself is defined in the accompanying document X.509 [ITU97b] and constructed as shown in Figure 3.8. The data type `Name` is a DN as above.

```
1 RelativeDistinguishedName ::=
2         SET SIZE (1..MAX) OF AttributeTypeAndValue
3
4 AttributeTypeAndValue   ::= SEQUENCE
5         type    ATTRIBUTE.&id ({SupportedAttributes}),
6         value   ({ATTRIBUTE.&Type ({SupportedAttributes}{@type})}
```

Figure 3.7: Relative Distinguished Name

Hierarchical Level	RDN	DN
Root		{}
Countries	C = GB	{C=GB}
Organizations	O = Telecom	{C=GB, O=Telecom}
Organizational Units	(OU = Sales, L=Ipswich)	{C=GB, O=Telecom, (OU = Sales, L=Ipswich)}
People	CN = Smith	{C=GB, O=Telecom, (OU = Sales, L=Ipswich), CN=Smith}

Table 3.1: Determination of Distinguished Names

Attribute extensions as defined in Figure 3.9 can be used, to add other desirable information to the certificate. If the criticality flag is set to `TRUE`, and the values can not be recognized upon parsing, the whole certificate shall be deemed invalid. Any organization may define attribute extensions as needed by using the extended key usage extension. [ITU97b]

As an example, the standardized extension in Figure 3.10 shows, how the use of a certificate can be restricted by defining extension properties. Each allowed use of a certificate has a specific property to be set, if that specific certificate is allowed to be used for that purpose. The `extKeyUsage` type allows to define own specific key usages not covered by [ITU97b].

Attribute certificates [ITU97b] are handled just as the above-mentioned user-certificates. The only difference is, that an attribute certificate contains more information about the user, for example date of birth, social security number, etc. There is no conceptual difference to a user-certificate.

```
Certificate ::= SIGNED { SEQUENCE {
     version [0]          Version DEFAULT v1,
     serialNumber         CertificateSerialNumber,
     signature            AlgorithmIdentifier,
     issuer               Name,
     validity             Validity,
     subject              Name,
     subjectPublicKeyInfo SubjectPublicKeyInfo,
     issuerUniqueIdentifier [1] IMPLICIT
          UniqueIdentifier OPTIONAL,
          -- if present, version must be v2 or v3
     subjectUniqueIdentifier [2] IMPLICIT
          UniqueIdentifier OPTIONAL
          -- if present, version must be v2 or v3
     extensions [3] Extensions OPTIONAL
     -- If present, version must be v3
} }
```

Figure 3.8: X.509 Certificate

```
EXTENSION       ::= CLASS {
     &id OBJECT     IDENTIFIER UNIQUE,
     &ExtnType
} WITH SYNTAX {
     SYNTAX &ExtnType
     IDENTIFIED BY &id
}
Extensions      ::= SEQUENCE OF Extension
Extension       ::= SEQUENCE {
     extnId     EXTENSION.&id ({ExtensionSet}),
     critical   BOOLEAN DEFAULT FALSE,
     extnValue  OCTET STRING
     -- contains a DER encoding of a value
     -- of type &ExtnType for the extension
     -- object identified by extnId
}
ExtensionSet EXTENSION ::= { ... }
```

Figure 3.9: X.509 Certificate Extensions

```
keyUsage EXTENSION      ::= {
     SYNTAX             KeyUsage
     IDENTIFIED BY      id-ce-keyUsage
}
KeyUsage                ::= BIT STRING {
     digitalSignature   (0),
     nonRepudiation     (1),
     keyEncipherment    (2),
     dataEncipherment   (3),
     keyAgreement       (4),
     keyCertSign        (5),
     cRLSign            (6),
     encipherOnly       (7),
     decipherOnly       (8)
}
```

Figure 3.10: KeyUsage Certificate Extensions

It is important to notice, that the signer of a certificate can, in principle, be anyone with a distinguished name. That effectively means, that it is possible to create chains of certificates, where the signer is to be verified by another certificate, and so on.

In a centralized certification scheme, for all participating users there is exactly one trusted third party, the *Certification Authority* (CA). The CA has the task to issue certificates, to prove, that a certain public key is bound to a certain identity. All players in the community trust the CA, that it is capable of doing so, and does it reliably.

The CA does this by signing the association between the identity and the public key with its own secret key. To test the signature of the CA on user's certificates, one has to have the public key of the CA. That public key is distributed in the CA certificate, which is the association of the CA's public key to its identity, signed by itself, called the *root certificate.*

To get hold of a certificate, an entity generates a *Certification Request*, [RSA93b] and sends this to the CA for further processing. The CA, after successfully generating the certificate, propagates the certificate via key distribution centers. (see below)

In centralized certificate scheme, all chains of certificates must ultimatively be traced back to the root certificate. Each player must have access to the root certificate and all intermediate certificates to verify the last in a

chain of certificates, the *leaf*. For better scaling of the installed architecture, intermediate CAs might be installed, all tracing their certificate back to the root certificate.

It is necessary, that the verifier has a reasonable level of confidence in the correctness of the root certificate. This normally involves testing the root certificate by some off-line means, like getting it from a CD-ROM, comparing it with a print media, or checking it vis-à-vis with the issuing organization.

An advantage of centralized certificate creation is the ability to accompany it with a centralized certificate distribution mechanism, called *key distribution center* (KDC). Every user, who needs a certificate to verify a chain, may contact the KDC to retrieve the needed key. The KDC's database is filled by the CA, that simply pushes all issued certificates.

To reduce the risk of secret key compromise, certificates are only valid in a pre-defined interval. A certificate is automatically invalidated outside this intervall. This is a particularly tricky and not yet resolved issue for the root certificate itself.

```
1  certificateRevocationList ATTRIBUTE ::= {
2       WITH SYNTAX CertificateList
3       ID id-at-certificateRevocationList }
4  CertificateList ::= SIGNED { SEQUENCE {
5       signature AlgorithmIdentifier,
6       issuer Name,
7       thisUpdate UTCTime,
8       nextUpdate UTCTime OPTIONAL,
9       revokedCertificates SEQUENCE OF SEQUENCE {
10      userCertificate CertificateSerialNumber,
11      revocationDate UTCTime } OPTIONAL }}
```

Figure 3.11: Certificate Revocation List

Keys still can get compromised within the valid time period of its corresponding certificate. If such a situation is detected, the user sends a *certificate revocation request* to the CA. The CA then retracts the certificate. To enable others to know of this invalidation, the CA provides a *certificate revocation list* (CRL), which contains all revoked certificates. [ITU97b] (cf. Figure 3.11) While checking any certificate, an entity has to check it against the CRL.

For different applications or communities different requirements for the quality of the certificate issuing process is required. The quality is defined by the

policy of the issuing organization, where each organization might issue several different kinds of certificates. There exist low-quality certificates, that bind a public key to an email-address without further identity checking, like Class 1 certificates by Verisign, [Ver97] as well as certificates that are only issued after presenting one's government-issued identification documents in person to a CA's representative. [Hei99] Certificates to be used for legally binding statements require a high quality, and the issuing CA has to take a lot of technical and organizational measures and has to undergo a rigorous process of getting certified therefore. [Reg97]

For different communities or application environments, different CAs might be used. Each tree of issued certificates of one CA is disjunct to all certificate trees of all other CAs. For interoperability, a policy mapping is desirable: when a CA in one domain certifies a CA in another domain, a particular certificate policy in the second domain may be considered to be equivalent (but not necessarily identical in all respects) to a particular certificate policy in the first domain by the authority of the first domain. [ITU97b]

There are two common interpretations of a CA: in the "loose" sense, any entity creating a certificate is called a CA. More "strict," only those entities undergoing the abovementioned rigorous accreditation process and whose certificates have a strong legal foundation are a CA.

3.7.6 Authentication

Authentication is the process of ensuring, that in a communication protocol a presented *identity* is impersonated rightly by the presenting entity. Further, an authenticated communication includes ensuring integrity of the data exchanged.

[ITU97b] specifies, in addition to the certificate format as presented in Section 3.7.5, a framework for mutual strong authentication based on cryptographic techniques. Successful completion of the three-way authentication protocol ensures, that both parties are asserted of the identity of the other side. Further, the communication (so far) is known to be unalterated. Including a nonce assures liveness and prevents a replay attack.

3.7.7 Authorization

Granting authorization is to allow an acting subject an operation on an object. The subject is some kind of active entity, usually a program acting on behalf of an user. The operation depends on the object, in the context of data files this might be read, write, execute, delete, change, and similar operations. The object is normally some kind of passive entity, e.g. a data file or process being acted on.

In the context of service-based software agents as described in Chapter 2, the operation is the call of a service, which is the only possible operation on a service, and the object is the service itself.

3.8 Secure Socket Layer

The primary goal of the SSL protocol (SSL) [FKK96] is to provide privacy and authenticity between two communicating applications. The protocol is composed of two layers. At the lowest level, layered on top of some reliable transport protocol (e.g., TCP), is the SSL Record Protocol. The SSL Record Protocol is used for encapsulation of various higher level protocols. One such encapsulated protocol, the SSL Handshake Protocol, allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data. One advantage of SSL is that it is independent of the transported application protocol. A higher level protocol can layer on top of the SSL protocol transparently. The SSL protocol provides connection security that has three basic properties:

- The connection is confidential. Encryption is used after an initial handshake to define a secret key. Symmetric cryptography is used for data encryption.
- The peer's identity can be authenticated using asymmetric, or public key, cryptography.
- The connection is reliable. Message transport includes a message integrity check using a keyed MAC.

SSL is a stateful protocol. All records are protected using the encryption and MAC algorithms defined in the negotiated cipher specification. There

is always an active cipher specification, however initially it is a no-op, which does not provide any security. Once the handshake is complete, the two parties have shared secrets which are used to encrypt records and compute keyed message authentication codes (MACs) on their contents. The techniques used to perform the encryption and MAC operations are defined by the cipher specification. Transmissions also include a sequence number so that missing, altered, or extra messages are detectable.

The cryptographic parameters of the session state are produced by the SSL Handshake Protocol, which operates on top of the SSL Record Layer. When a SSL client and server first start communicating, they agree on a protocol version, select cryptographic algorithms, optionally authenticate each other, and use public-key encryption techniques to generate shared secrets. These processes are performed in the handshake protocol, which is shown in Figure 3.12. [FKK96]

Figure 3.12: SSL Handshake Protocol

The client sends a client hello message to which the server must respond with a server hello message, or else a fatal error will occur and the connection will fail. The client hello and server hello are used to establish security enhancement capabilities between client and server. The client hello and server hello establish the following attributes: protocol version, session id,

cipher suite, and compression method. Additionally, two random values are generated and exchanged.

Following the hello messages, the server will send its certificate, if it is to be authenticated. Additionally, a server key exchange message may be sent, if it is required (e.g. if their server has no certificate, or if its certificate is for signing only). If the server is authenticated, it may request a certificate from the client, if that is appropriate to the cipher suite selected. Now the server will send the server hello done message, indicating that the hello-message phase of the handshake is complete. The server will then wait for a client response. If the server has sent a certificate request message, the client must send either the certificate message or an alert. The client key exchange message is now sent, and the content of that message will depend on the public key algorithm selected between the client hello and the server hello. If the client has sent a certificate with signing ability, a digitally-signed certificate verify message is sent to explicitly verify the certificate.

At this point, a change cipher spec message is sent by the client. The client then immediately sends the finished message under the new algorithms, keys, and secrets. In response, the server will send its own change cipher spec message, and sends its finished message under the new Cipher Spec. The handshake is now complete and the client and server may begin to exchange application layer data.

Application data messages are carried by the Record Layer and are fragmented, compressed and encrypted based on the current connection state. The messages are treated as transparent data to the record layer.

The security depends on the properties of the generated randomness for keying material. (cf. Section 3.4.1) As the key is usually generated by the client, a common PC, the quality of the keys will be disputable.

SSL itself does not cope with the validity of certificates. The target applications have to provide their own means of checking. This might involve using a host library or, if certificate checking is skipped altogether, leads to non-authenticated communication.

SSL is currently the standard for encrypted and authenticated communication on the connection layer.

3.9 System Security Requirements

The most fundamental requirement standard for computer security is defined in the "Orange Book" [TCS85] of the "Rainbow Book" [NCS] Series. The requirements for the writing were motivated by the need of the military and government agencies to determine the level of trust to be put into computer systems. Features of systems should be made measurable, acquisitions should be better described, and manufacturers given a categorization of features. The evaluation criteria, in the spirit of the time of its writing, pertained primarily to host systems, networks are only implicitly covered and are not explicitly dealt with. It defines four requirements:

Security Policy: The security policy of the system must be well-defined and enforced by the system.

Marking: Objects must have access control labels, that describe the allowed operations.

Identification: Subject accessing data must be identifiable, and the access to the data is granted based on that information.

Accountability: The system must selectively collect security auditing data, and protect that information against tampering.

The document defines criteria against that systems are to be measured for fulfilling the above security requirements. It defines the four divisions A through D, with some classes as lower structural levels in divisions B and C. Division D provides only "minimal protection," according to [TCS85], but that name is misleading, because division D pertains to completly unsecured systems as well, as long as they have been assessed against the standard.

Classes in division C provide for discretionary (need-to-know) protection and, through the inclusion of audit capabilities, for accountability of subjects and the actions they initiate. Systems in class C1 provide separation of users and data. It incorporates some form of credible controls capable of enforcing access limitations on an individual basis, it allows users to be able to protect information and to keep other users from reading or destroying their data.

Allowed object sharing of class C1 is more restricted in class C2: the system shall provide controls to limit propagation of access rights. The discretionary

access control mechanism shall provide that objects are protected from unauthorized access. These access controls shall be capable of including or excluding access to the granularity of a single user. Access permission to an object by users not already possessing access permission shall only be assigned by authorized users.

Division B is a jump up in security requirements by demanding mandatory protection: the notion of a Trusted Computing Base (TCB) that preserves the integrity of sensitivity labels and uses them to enforce a set of mandatory access control rules is a major requirement in this division. Systems in this division must carry the sensitivity labels with major data structures in the system. The system developer also provides the security policy model on which the TCB is based and furnishes a specification of the TCB. Evidence must be provided to demonstrate that the reference monitor concept has been implemented. The reference monitor is the part of the TCB, that enforces the authorized access relationships between subjects and objects of a system, and must fulfill all of these requirements: [And72]

- The reference validation mechanism must be tamper proof.
- The reference validation mechanism must always be invoked.
- The reference validation mechanism must be small enough to be subject to analysis and tests, the completeness of which can be assured.

Class B1 systems are said to have labeled security protection and require all the features required for class C2. In addition, an informal statement of the security policy model, data labeling, and mandatory access control over named subjects and objects must be present. The capability must exist for accurately labeling exported information. The TCB shall enforce a mandatory access control policy over all subjects and storage objects under its control. These subjects and objects shall be assigned sensitivity labels that are a combination of hierarchical classification levels and non-hierarchical categories, and the labels shall be used as the basis for mandatory access control decisions. The TCB shall be able to support two or more such security levels.

In class B2 systems, the TCB is based on a clearly defined and documented formal security policy model that requires the discretionary and mandatory access control enforcement found in class B1 systems be extended to all subjects and objects in the system. In addition, covert channels are addressed. The TCB must be carefully structured into protection-critical and non-protection-critical elements. The TCB interface is well-defined and the

TCB design and implementation enable it to be subjected to more thorough testing and more complete review. Authentication mechanisms are strengthened, trusted facility management is provided in the form of support for system administrator and operator functions, and stringent configuration management controls are imposed.

The class B3 TCB must satisfy the reference monitor requirements that it mediates all accesses of subjects to objects, be tamperproof, and be small enough to be subjected to analysis and tests. To this end, the TCB is structured to exclude code not essential to security policy enforcement, with significant system engineering during TCB design and implementation directed toward minimizing its complexity. A security administrator is supported by introducing security domains, audit mechanisms are expanded to signal security-relevant events, and system recovery procedures are required.

Division A is characterized by the use of formal security verification methods to assure that the mandatory and discretionary security controls employed in the system can effectively protect classified or other sensitive information stored or processed by the system. Extensive documentation is required to demonstrate that the TCB meets the security requirements in all aspects of design, development and implementation.

Systems in class A1 are functionally equivalent to those in class B3 in that no additional architectural features or policy requirements are added. The distinguishing feature of systems in this class is the analysis derived from formal design specification and verification techniques and the resulting high degree of assurance that the TCB is correctly implemented.

The Orange Book allows for classes better than A1 but states, that it is beyond current technology. There are guidelines though, in which directions that classes might delve. At the time of writing this thesis, there are no operating systems listed as being of class A1 and only one, in several versions, is classified as B3. [NCS00]

3.10 Attacks

There is a multitude of possibilites to undermine the security of a system. Unfortunately, there can be no comprehensive list of possible attacks, because every day a new one can be discovered or implemented. Research never stops, ingenuity can't be predicted.

It is possible, though, to identify areas in which attacks can be classified, and where standard approaches are known. A system should, as far as a technical system can cope with some of the more obscure or solely social issues, at least try to minimize the risks the attacks described in the following pose.

3.10.1 Attacks on Algorithms

The most fundamental attack form is to try to break the cryptographic algorithm used. The categorization of the attacks is based on the available information, an attacker has, and the possibilities for modification of data. [Sch96]

In a *ciphtertext-only attack* the attacker has the ciphertext of several messages all of which have been encrypted using the same encryption algorithm. The attacker wants to recover the plaintext of as many messages as possible. Better yet, he tries to find out the key used to encrypt the messages.

In a *known-plaintext attack,* in addition to the ciphertext, the attacker has the plaintext messages thereof as well. A successful attack is conducted, if it is possible to deduce the key used, or to be able to recover other plaintexts encrypted with the same key, be it either known or not.

In a *chosen-plaintext attack,* in addition to the initial information as given in the previous attack, an attacker has the possibility to inject plaintext chosen by him and can get hold of the resulting ciphertext. Again, the keys are sought, or an algorithm to recover plaintext from ciphertext.

In an *adaptive-chosen-plaintext attack* an attacker has the same information and possibilities as in the previous attack. In addition, feedback of insights achieved from injecting own plaintext can be used for further plaintext input. This is the case, e.g., in an ongoing communication, where answer messages can be observed and new injected messages can be constructed in reaction to the received answers.

As a specific variant, using *differential cryptanalysis* the adversary tries to exploit differences in the ciphertext as generated by slightly different plaintexts. The resulting ciphertext differences are used to conclude about weaknesses of the algorithm by then being able to deduce about different probabilities of keys. This reduces the key space to search for the correct key. (cf. Section 3.4.2)

In a *chosen-ciphertext attack* an attacker is able to let the system decrypt ciphertexts chosen by him, and evaluates the decryption output. This is a "promising" attack against public key algorithms. [Sch96]

A *chosen-text attack* is a combination of a chosen-plaintext attack with a chosen-ciphertext attack: the attacker has both possibilities.

A *chosen-key attack* is a very sophisticated form of attack, where relationships between different keys is alleviated, similar to differential cryptanalysis for plaintexts. Another name of it is *related-key* attack.

In the known-plaintext version of a chosen-key attack, the attacker knows the plaintext and the ciphertext and tries to deduce the keys used. In the chosen-plaintext version, the attacker might inject plaintexts of his choosing.

In *linear cryptanalysis*, the inner working of a block cipher is approximated by a linear equation. This gives probabilities for the correctness of parts of the key used. The more material to test the resulting hypothesis on, the more probable the success of the attack is.

3.10.2 Brute Force Attacks

A very simple form of attack is just to try out different keys. This is an implementation of a ciphertext-only attack: the attacker tries each key of the keyspace, decrypts the ciphertext, and tests the result for being a reasonable plaintext. This attack is very effective in real-world situation due to the following reasons.

Users often choose *bad passwords* used as keys or for key generation. They use words easily found in a dictionary, even worse, the passwords might have a personal connection to the user. This enables the attacker, after trying obvious socially related passwords like the spouse's name or the birthday of the kid, to run a dictionary attack. All words of a language dictionary, with variations, are tried, until the password or key is recovered.

A keyspace should be large enough, that only a very minor fraction of it can be tested against in reasonable time. (cf. Section 3.4.2) Some algorithms only use 40 or 56 bit, which is not enough for today's standards. Thus, a *small keyspace of the algorithm* leads to insecurity.

The system in use, like the login prompt of a computer system, restricts the keyspace even further. Although an algorithm might allow a range of 256 values for each key byte, there are often restrictions on the values actually possible due to the *design of the cryptosystem:* entering all 256 values at a login prompt is normally not allowed or possible.

Increasing computing power make the computers of today more powerful than designers might have expected upon inception of their algorithms. Dis-

tributing the necessary computation over lots of nodes in a computer network mobilize additional resources not available before. Brute force attacks become feasible where they haven't been some years ago.

It is easier to automatically *recognize a recovered plaintext* than one might think. First, most computer data follows a specific format, e.g. a bank transaction might always comprise of two account numbers, two bank numbers, an amount, etc., in a fixed order. Natural language is easy to discern as well, because each language has a specific index of coincidence of all letters and letter combinations. [otA90] gives a detailed account how to recognize the English language algorithmically.

3.10.3 Attacks on Protocols

Another general form of attack is not to undermine the algorithm but instead the protocol the algorithm is used in. These attacks can be categorized as follows.

In a *passive attack,* the adversary has no possibility to manipulate the protocol exchange. Instead, he is only able to eavesdrop on the communication. It is a protocol-related form of a ciphertext-only attack. Protocols can neither prevent nor detect this form of attack, but have to provide for not falling victim of it.

In an *active attack,* the adversary has the possibility to manipulate the exchanged data stream. This can take several forms that can be combined: being a man in the middle, manipulating bits in a stream, replaying old communication, inserting own data, substituting or reordering messages, cutting off communication as wanted, etc.

Protocols can be disrupted on the semantical level by *cheating attacks.* An otherwise legitimate participant of a protocol might give false data, withhold important data, or disregard the protocol's sequence. The latter is called active cheating in contrast to the aforementioned types of passive cheating.

This kind of attack is very difficult to circumvent on the technical level, because it involves the semantics of the data exchanged, which can't be evaluated for the general case. Self-enforcing protocols, arbitrated, and adjucated protocols might be used to prevent cheating.

3.10.4 Attacks on Systems

Using a "secure" protocol only featuring well-accepted algorithms by itself does not guarantee a secure system: "details matter." [Sch96]

For example, it doesn't do any good in SSL (cf. Section 3.8) to exchange a strong symmetric cipher specification and associated key material if one renounces authentication beforehand, because then any man in the middle could act as the communication partner and be part of the key exchange himself without being noticed as such. This is not the fault of SSL, because strong authentication is supported. It is the fault of the system in use, that doesn't employ the features provided.

Even if each secure subpart of a system is used correctly doesn't mean, the resulting system is secure. Very popular design errors do use, e.g., SSL in a secure manner, and the data afterwards is processed behind good firewalls. Unfortunately, the interim data between receipt via SSL and further processing is stored unprotected on the web server, making it available to any one in the world, ironically protected by SSL as well (see e.g. the list at [Sec00]).

Connection high-jacking is another form of attacks on cryptosystems. Insecure lower-layer protocols are used to subvert an established connection between two peers by substituting for one of them. Because of the inherent insecurity of TCP connections, this attack must be countered on the security system level.

3.10.5 Attacks on Implementations

Implementation bugs of systems can be exploited by attackers. This is one of the most practical attacks today. Faults in web browsers, specifically in the subsystem of securing active content execution, are the prevalent cause of danger for the client side. [Sec00] Server software is prone to common programming errors, [Var00] gives a frightening list of vulnerabilities.

Because making errors is human, programs will have bugs. As discussed in Section 3.3.2, security can be gained by peer review of the program's source code and reducing those bugs.

3.10.6 The Human Factor

Another very practical way to attack a system involving people is to exploit the human factor: "That sack of organs sitting in front of the monitor has more security holes than all the Microsoft products put together, and its much easier to write malicious scripts to exploit them." [Hig00]

People can be deluded by misinformation in numerous ways. They might be faked about the machine, they are "talking" to, they might be tricked to enter sensitive information to an untrusted program, they can be deceived by phone to give their passwords to the "administrator," etc.

Installation of unverified software or from untrusted sources is another source of compromize. The installed software might be a backdoor to the computer or spying and sending information by various ways back to the attacker or whatever else a local program might do on the machine.

Another factor to recon with is missing skills of the people using a cryptosystem. They might not know, how to use it properly, or might be lacking consciousness about their doing and the requirements thereof. Teaching might help, but it requires capability and willfullness of the people. Being forced by higher authorities in a company to do something *very important now* while often being ignorant about the security issues very quickly leads to bypassing or ignoring security concerns as well.

Carelessness of people using the system is another path to insecurity. Passwords are written on paper and stored under the keyboard, plain text files contain sensitive information like keys, and so on. A security policy for the system that defines the requirements of use and proper handling should be in place to reduce the human factor of insecurity.

3.11 Summary

For realizing a secure system, there are a lot of algorithms, protocols, and crypto systems to choose from. Because their applicability pertain to different cicumstances, there is no single choice to be made available. Therefore, the whole suite of solutions must be provided for a system.

This condition is even worsened for interoperability reasons. Because in an open environment it can not be determined beforehand, or a global standard be established, which mechanisms to use for securing a system. For

maximum interoperability, flexibility, and applicability a vast range of supported mechanisms and parametrization must be supported. In addition, depending on the circumstances, the mechanisms to use and to enforce must be selectable and configurable. It is mandatory though, not to invent own solutions but instead employ proven and well-analysed mechanisms must be employed.

A comprehensive security solution must therefore provide for flexibility and adaptivity to the local circumstances. Components and intelligent self-inspection of the software must be supported as well.

The next Chapter will give a technical overview of Java and its security-related features, followed by a Chapter that delves into the agent domain.

Chapter 4
Java

"Drink your coffee.
Maybe it will warm your heart."
Miss Brown, Birds of Passage

4.1 Synopsis

Since the inception of Java technology there has been strong and growing interest around the security of the Java platform as well as new security issues raised by the deployment of Java technology. Java is a registered trademark of Sun Microsystems, Inc.

From its early days, Java was designed to be a framework "native" to the network. There should be no conceptual difference between a "local" and "remote" program, both should be executed likewise. The only difference, as will be discussed below, is the difference in trust that is put into the remote code. "Write once, run everywhere" is the related and widely known slogan. [Sun99b]

Further, Java was designed to enable application execution at the client side of a network connection, especially if transmitted over Web pages. For these reasons, in addition to stand-alone applications the applet concept had been introduced and implemented. An applet is a piece of code that can be transmitted from a web server to a client web browser and be executed in the browser, the execution is controlled by the browser.

From the beginning of promoting the Java technology, Sun Microsystems has released tools to develop programs written in Java. The fundamental toolkit is the Java Development Kit (JDK). Because the JDK Version 1.2 [Sun00a] introduced several new and powerfull security features, as will be discussed in this Chapter, that version of the JDK was chosen to be the basis of the concept and implementation as presented in Part III.

Java security includes the following aspects, that will be elaborated in the remainder of this Chapter:

- A language concept with strong data typing and features for error-free programming.
- The Java platform as a secure platform on which to run Java applications in a secure fashion.
- The standard API as a class library supporting security.
- Security tools and services implemented in the Java programming language that enable security-sensitive applications.

4.2 Programming Language

The Java programming language (abbreviated in the following as only "Java") is a general-purpose concurrent class-based object-oriented programming language, specifically designed to have as few implementation dependencies as possible. It allows application developers to write a program once and then be able to run it everywhere on the Internet. [GBS00]

The evolution of Java started with its identity as a subset of C++ with slightly different syntax. This subset was designed to eliminate features of the C++ language that were deemed to be error-prone or unsafe. In this sense the term *code safety* refers to the property that Java allows to generate bug-free code that doesn't crash the program or the machine. This is an aspect of robustness, because Java helps preventing writing programs that are accidentally harmful. [Ins98]

Reducing the possibility for programming errors reduces the probability of errorneous programs, that show unwanted and probably unsecure behavior. Thus, Java as a programming language helps to write correct programs and

to provide reliable systems. This safety adds to the security of a system as discussed in Section 3.10.5.

4.2.1 Invalid Memory Access

Java does not enable pointer arithmetic. This is perhaps the most important language feature that contributes to the Java language's safety, since it is pointer arithmetic that leads to accessing inappropriate memory areas, which leads to runtime crashes or access to otherwise protected or sensitive data.

In addition to removing pointer arithmetic the Java language specification defines behavior of uninitialized variables. All heap-based memory is automatically initialized, all stack-based memory isn't. All class and instance variables are always set to defined values, and all local variables must be initialized before use or the source compiler generates an error.

4.2.2 Garbage Collection

Another feature of Java that contributes to safe code is automatic garbage collection. The runtime environment is able to automatically release memory no longer referenced.

Java also has a stack concept for memory allocation. There can be multiple stacks within a program, one for each executing thread. The only memory allocated on the stack is for local variables of a method. When the method ends, the variable space is relinquished. Due to the Java garbage collector concept, a programmer no longer needs to determine when to release memory. The garbage collector will only release memory when it is safe and no longer referenced.

4.2.3 Other Safety Features

Strong compile-time type-checking disallows illegal casts. It ensures, that a programmer can't interpret a block of memory as anything other than what it is.

Java's access modifiers `public`, `package`, `protected`, and `private`, contribute to robustness by controlling visibility of class members. One can restrict access to data instances, methods, and inner classes.

The Java programming language also provides the `final` modifier, which disallows subclassing when applied to class definitions and disallows overriding when applied to method definitions. This makes it impossible to overwrite security-sensitive class members.

4.3 Execution Environment

Java is a byte-compiled language. This means, Java source code is compiled by the source level compiler, after checking the correctness of the code against the language definition, [GBS00] into an interim coding, the byte code. That byte code is not directly executed on a target machine as it is the case with other compiled languages.

Instead, the byte code is interpreted as the instruction set for a Java virtual machine (JVM) as defined in the Java VM specification. [LY99] For each target platform that shall run Java Code there must exist a native implementation of the JVM, the Java Runtime Environment (JRE). That JRE then interprets and executes the byte code. Because interpreting the byte code is a performance bottleneck, a runtime compiler for on-demand generation of native code is included in a standard Java distribution today.

4.3.1 Byte Code Verification

Before a class is loaded into the JVM, it verifies, that the byte code conforms to the Java language specification. [GBS00] This prevents executing maliciously modified or unintentionally errorneous code before it gets into the JVM.

4.3.2 Class Loading

Dynamic class loading is an important feature of the Java Virtual Machine (JVM), because it provides the Java platform with the ability to install software components at runtime. It has a number of unique characteristics.

Lazy loading means that classes are loaded on demand and at the last moment possible. Dynamic class loading maintains the type safety of the JVM by adding linktime checks, which are performed only once. Programmers

can define their own class loaders that, for example, specify the remote location from which certain classes are loaded, or assign appropriate security attributes to them. This way, programs to be loaded are not restricted to be located on the local file system, but might equally reside on the network. Class loaders can be used to provide separate name spaces for various software components. (cf. Section 4.3.5)

The class loading mechanism is central to the dynamic nature of the Java programming language. It also plays a critical role in providing security, because the class loader is responsible for locating and fetching the class file, consulting the security policy, and defining the class object with the appropriate permissions.

4.3.3 Runtime Checking

After classes are loaded, the JVM's next level of security performed is its runtime checking. Because of the late binding provided by the JVM, additional late type checking is done of assignments and array bounds at runtime. While certain type-checking can be done at compile time, there are cases where the specific type of an object is not determinable until runtime. In these cases, the JVM ensures, that only properly assignable operations are performed.

While removal of pointer arithmetic is a compile-type operation, array bounds checking is a runtime check. If an invalid array position is referenced, the JVM throws an exception.

4.3.4 Sandbox

The original security model provided by the Java platform and implemented in the JDK 1.0 is known as the sandbox model, which implemented a very restricted environment in which to run untrusted code obtained from the network. The essence of the sandbox model was, that local code is trusted to have full access to vital system resources while downloaded remote code is not trusted and can access only the limited resources provided inside the sandbox. This sandbox model is illustrated in Figure 4.1. [Gon98]

JDK 1.1 introduced the concept of a signed applet, as illustrated by Figure 4.2. [Gon98] In that Java release, a correctly digitally signed applet is treated as if it were trusted local code, if the signature key is recognized as trusted by the end system that receives the applet. Signed applets, together

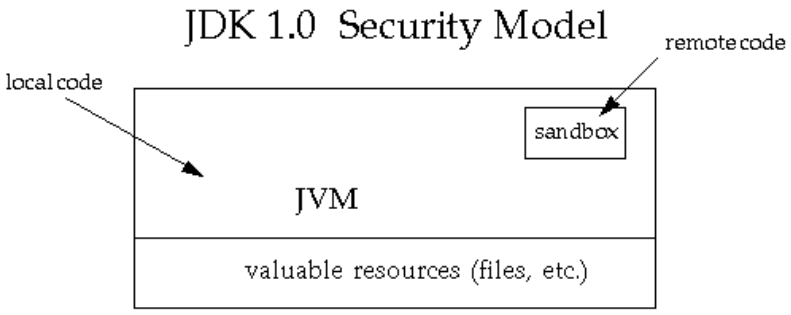

Figure 4.1: Java 1.0 Sandbox

with their signatures, are delivered in the JAR (Java Archive) format. In JDK 1.1, unsigned applets still run in the sandbox.

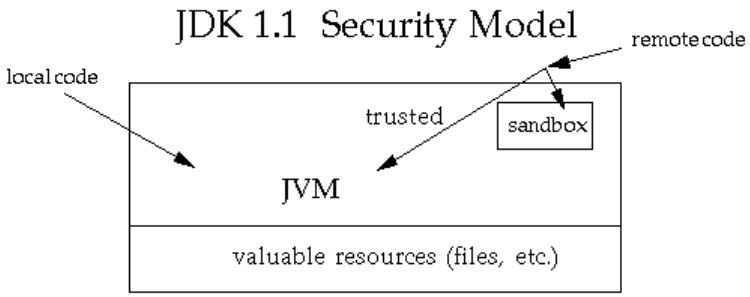

Figure 4.2: Java 1.1 Sandbox

The security architecture in JDK 1.2, illustrated in Figure 4.3, [Gon98] introduces several new features to the Java runtime environment, as will be discussed in the following.

Fine-grained Access Control

This capability existed in the JDK from the beginning, but to use it, the application writer had to do substantial programming. Such programming is security-sensitive and requires sophisticated skills and in-depth knowledge of computer security. The JDK 1.2 architecture makes this simpler and safer.

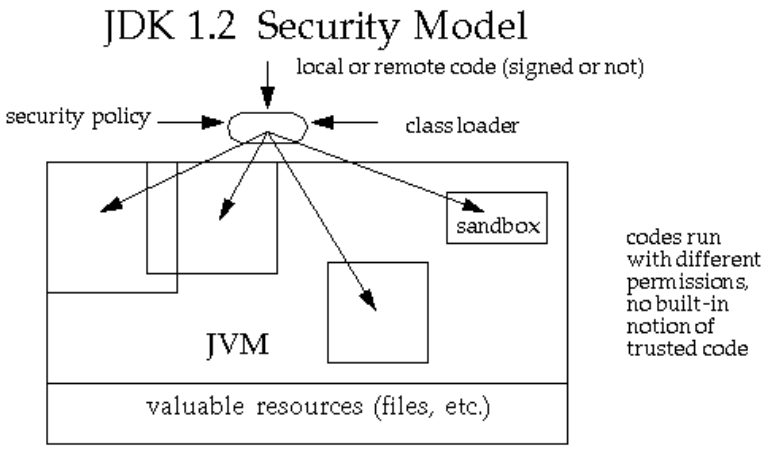

Figure 4.3: Java 1.2 Sandbox

Configurable Security Policy

Again, this capability existed previously in the JDK, but was not easy to use. Moreover, writing security code is not straightforward, so it is desirable to allow application builders and users to configure security policies without having to program.

Extensible Access Control Structure

Up to JDK 1.1, in order to create a new access permission, one had to add a new check method to the `SecurityManager` class. The new architecture allows typed permissions, each representing an access to a system resource, and automatic handling of all permissions, including yet-to-be-defined permissions, of the correct type. No new method in the `SecurityManager` class needs to be created in most cases.

Extension of security checks

In JDK 1.2, there is no longer a built-in concept, that all local code is trusted. Instead, local code is subjected to the same security control as remote code, although it is possible, if desired, to declare that the policy on local or remote code be more liberal, thus enabling such code to effectively run as totally trusted. The same principle applies to signed applets and any Java application.

4.3.5 Protection Domains of Java 1.2

The fundamental concept of the Java execution environment and its important building block of system security is the protection domain. (cf. Section 3.9) [SS75] A domain can be scoped by the set of objects that are currently directly accessible by a principal, where a principal is an entity in the computer system to which permissions and, as a result, accountability are granted. (cf. Section 3.9) The sandbox utilized in JDK 1.0 is one example of a protection domain with a fixed boundary.

The protection domain concept serves as a mechanism for grouping and isolation between units of protection. It is possible to separate protection domains from interacting with each other, so that any permitted interaction must be either through trusted system code or explicitly allowed by the domains concerned.

Protection domains generally fall into two distinct categories: system domain and application domain. It is important that all protected external resources, such as the file system, the networking facility, and the screen and keyboard, be accessible only via system domains.

A domain conceptually encloses a set of classes, whose instances are granted the same set of permissions. Classes signed by the same keys and from the same URL are placed in the same domain. A domain also encompasses the permissions granted to code in the domain, as determined by the security policy currently in effect. Classes that have the same permissions but are from different code sources belong to different domains. A class belongs to one and only one protection domain. In JDK 1.2 protection domains are created on demand as a result of class loading. (cf. Section 4.3.2)

Each domain may also implement additional protection of its internal resources within its own domain boundary. For example, a banking application may need to support and protect internal concepts such as checking accounts,

deposits, and withdrawals. Because the semantics of such protection is unlikely to be predictable or enforceable by the JDK, the protection system at this level is best left to the system or application developers. The JDK provides utility classes to this effect. (cf. Section 4.4)

A thread of execution may occur completely within a single protection domain or may involve an application domain and also the system domain. For example, an application that prints a message out will have to interact with the system domain that is the only access point to an output stream. In this case, it is crucial that at any time the application domain does not gain additional permissions by calling the system domain. Otherwise, there can be serious security implications.

In the reverse situation where a system domain invokes a method from an application domain it is again crucial that at any time the effective access rights are the same as the current rights enabled in the application domain. Thus, a less powerful domain cannot gain additional permissions as a result of calling or being called by a more powerful domain.

4.3.6 Access Checking

Sensitive JDK system code invokes methods of the class `SecurityManager` to check the policy currently in effect and perform access control checks. There is typically a security manager installed whenever an applet is running. A security manager is not automatically installed, when an application is running.

Any code that controls access to system resources should invoke methods of the class `AccessController`, if it wishes to use the specific security model and access control algorithm utilized by these methods. If the application wishes to defer the security model to that of the `SecurityManager` installed at runtime, then it should instead invoke corresponding methods in the `SecurityManager` class.

The `SecurityManager` represents the concept of a central point of access control, while an `AccessController` implements a particular access control algorithm with special features.

4.4 Standard Class Library

Some classes that very strongly interact with the runtime environment have already been mentioned. There are some more security related mechanisms distributed with every standard Java distribution.

The JDK Security API is a core API of the Java programming language. This API is designed to allow developers to incorporate both low-level and high-level security functionality into their programs.

The Java Cryptography Architecture (JCA) refers to a framework for accessing and developing cryptographic functionality for the Java platform. It includes APIs for digital signatures and message digests. The certificate management infrastructure supports X.509v3 certificates (cf. Section 3.7.5) and realizes a Java Security Architecture for fine-grained, configurable, flexible, and extensible access control. [Gon98] The JCA allows for multiple and interoperable cryptography implementations.

The Java Cryptography Extension (JCE) [Sun00c] extends the JCA API to include APIs for encryption, key exchange, and message authentication codes. (cf. Chapter 3) Together, the JCE and the cryptography aspects of the JDK provide the cryptography API. The JCE is released separately as an extension to the JDK, in accordance with US export control regulations at the time of its inception. (cf. Section 2.6)

4.5 Java Tools

Within the JDK, several security related tools are provided. These tools realize only very basic functionalities.

4.5.1 Keytool

The keytool is a key and certificate management utility. It enables users to administer their own public/private key pairs and associated certificates. The authentication information includes both a sequence of X.509 certificates [ITU97b] and an associated private key. This tool also manages certificates that are trusted by the user, which are stored in the same database as the authentication information.

The keytool stores the keys and certificates in a keystore. The default keystore implementation implements the keystore as a file. It protects private keys with a password. Each private key in the keystore can be protected using its own individual password.

4.5.2 JAR Signing Tool

The jar signing and verification tool is used to digitally sign Java archives, that are held in jar files, and to verify such signatures. This tool depends on the keystore that is managed by the keytool.

4.5.3 Policy Tool

The policy tool is a small program with a graphical user interface that assists an user in managing a security policy.

4.6 Summary

The Java programming language and runtime environment provides several valuable aspects to use it for the realization of security-demanding applications. The language enforces robust and reliable programs, adding to security. Libraries are readily provided that add important functionality in the security domain. There exists a framework for the security of the runtime-system. Even mobile code finds support in the Java system. Hence, Java is chosen as the basis of the framework and implementation as discussed in this work.

Especially due to the jurisdictional situation in the United States of America, (cf. Section 2.6) the provided mechanisms of Java are not fully distributed. Additional libraries have to be employed to fill the functionality not provided with the Java system as distributed.

Chapter 5
Agents

"Never send a human
to do a machine's job."
Agent Smith, The Matrix

5.1 Synopsis

The notion of an agent has many roots, and its use is accordingly very diverse. Within this Chapter, more light shall be shed on "Agents." The most widespread notions of the agent term and definitions of what an agent is are given.

Examination of the concepts of a single agent follows. Multiple agents can be combined to form an agent community as discussed in the next Section.

A description of agent architectures defining the framework in which to realize a concrete agent system follows. Platforms and toolkits are used to form the model of an agent architecture into an executing software system. This Chapter is closed with a discussion of common misunderstandings and overexcitement in the software agent domain.

5.2 Notions of an Agent

"Although the term [Agent] is widely used by many people working in closely related areas, it defies attempts to produce a single universally accepted definition." [WJ95]

The diversity of the term "Agent" is due to its many ancestors. One important influence of Agent-oriented Technology (AOT) is the rise of Distributed Artifical Intelligence (DAI) at the beginning of the 1980's. The DAI branch deals with emergent high-level properties of complex distributed systems. Interaction and cooperation are important research topics in these "intelligent" systems. [BG88] Another root of AOT lies in parallel object-oriented programming (OOP), combining conventional OO-techniques with distributed systems. [Hew77]

There are a lot of other disciplines that influence AOT as well, due to the wide range of anticipated or realized applications. Organizational research, decision theory, psychology, theories of cognition, philosophy, and others all play a role for the agent domain.

The main branches of the concept of an agent are:

The agent as an object with competence: Agents are seen as an extension to the OOP paradigm. Agents are recognized as complex objects with a higher degree of autonomy and flexible interactions. [Sho93a]

The agent as a part of a multi-agent system: The interaktion between agents are the focus of this conceptualization. Each agent is a specialized problem-solver, that communicates and cooperates with other agents to solve a common and complex problem. [GK94]

The agent as an autonomous actor: An agent has a view of its environment. It decides for itself about the goals to pursue and actions to be taken. Interactions with other agents are possible. [RN95, FG96]

The agent as a mental system: Following this idea, an agent is a logical system for the description of a "mental state" including beliefs, capablities, obligations, etc. This is problematic insofar, as those attributes sometimes differ considerably from the common understanding of these terms. [Sho93b, WJ95]

The agent as personal presentation: The main motivation is to have the agent acting on behalf of its user without continous control or interference of the user. Further, the agent shall adapt to his preferences. [Gil96]

All of these views of an agent have some common aspects, but all are different in some respect. There is no agreed-upon concise notion of an agent. Though, for the remainder of this work it is sufficient to grasp the common intent of the above notions of an agent.

Agent-Oriented Techniques (AOT) promise several advantages over traditional approaches, especially in the area of complex and highly distributed computing. Using agents as basic building blocks for modeling and realization is intuitive and helps grasping the tasks-to-be-solved easier.

As being highly parallel and multi-threaded entities, more efficient computing can be achieved. Robustness and reliability are further factors that can be achieved due to distribution and redundancy.

Despite their relatively strict closure against its environment, agents are inherently communication-oriented. In the normal case as implied in this work, agents are composed of modules. These properties yield increased flexibility and dynamics to the resulting overall system. Adding and removing agents and/or their modules is possible without much hassle if provided for by the realizing agent architecture.

5.3 Definitions of Agents

There are a lot of definitions of agents, differing in the underlying notion of an agent resp. which implementation of an agent system is described. (cf. Section 5.2) Some definitions are vague and reference intuitive associations. Often, they described goals to be reached rather than possible realizations.

In the following, some important definitions are given. The result will be a definition of an agent to be used in the remainder of the work and in the realized architecture. A commented overview of definitions is given in [FG96].

In the definition of agents it is often mentioned, that they can be described by mental attributes like “knowledge” and “goals.” [WJ95] Sometimes, this is used as the sole criterion: “An agent is an entity whose state is viewed

as consisting of mental components such as beliefs, capabilities, choices, and commitments. [Sho93a]

Though, the question, if an entity is an agent or not, can't be solved objectively: "Agenthood is in the mind of the programmer." [Sho93a] There is no widely adopted theory about how to attribute mental states to programs, which attributes to use, and what the relevant parameters are. Those attributions today are based on the intuition and wishful thinking of the agent system designer, or the user. A common understanding about an agent is not present.

Another well-ambitioned approach to define an agent is based on the concept of an autonomously acting entity: "An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future." [FG96]

Or a similar approach: "An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors . . . A rational agent is one that does the right thing . . . the right action is the one that will cause the agent to be most successful." [RN95]

Unfortunately, as soon as perception is interpreted as data input, acting is interpreted as data output, and the environment is seen as the execution environment, this "definitions" transform every program into an agent. Even adding attributes of goal orientation and rationality ("doing the right thing") doesn't help much in this dilemma: each program serves a specific purpose, and is intended to fulfill this as best as it can, namely by programming it so. Intuition of the recipient rules the definition again.

The possibly most widely adopted definition of an agent tries to define an agent by giving specific properties of a computer program to fulfill. According to [WJ95], an agent is constituted by a program with the following attributes:

autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state; [Cas95]

social ability: agents interact with other agents (and possibly humans) via some kind of agent communication language; [GK94]

reactivity: agents perceive their environment, which may be the physical world, a user operating a graphical user interface, a collection of other

agents, the Internet, or perhaps all of these combined, and respond in a timely fashion to changes that occur in it;

pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative.

[WJ95] gives other important attributes, that are often present, but optional:

mobility is the ability of an agent to move around in an electronic network; [J.E94]

veracity is the assumption that an agent will not knowingly communicate false information; [Gal88]

benevolence is the assumption that agents do not have conflicting goals and that every agent will therefore always try to do what is asked of it; [RG85] and

rationality is the assumption that an agent will act in order to achieve its goals and will not act in such a way as to prevent its goals being achieved — at least insofar as its beliefs permit. [Gal88]

These definitions, too, are intuitive and allow for ample interpretation space. Other attributes like coordination, cooperation, adaptivity, planning ability, being able to reflect its own behavior are often mentioned in addition. All have in common, that there are neither widely agreed upon, nor are they in themselves well-defined terms.

In recent years, the Foundation for Intelligent Physical Agents (FIPA), an organization pursuing "the promotion of technologies and interoperability specifications that facilitate the end-to-end interworking of intelligent agent systems in modern commercial and industrial settings," [Fou00] has become an influencial part of the agent community. Their definition of an agent is more practically oriented: "An Agent is the fundamental actor in a domain. It combines one or more service capabilities into a unified and integrated execu-tion model which can include access to external software, human users and communication facilities." [Fou97] Here, the agent is seen as a closed entity providing services. Problematic is the differentiation from object oriented programming.

The following definition of an agent, as it shall be applied for the remainder of this work, is more oriented at practical concerns and feasibility: "An agent is an autonomous specialist that, within a system of multiple agents, provides a

specific functionality or works on a specific task. Therefore, it communicates with other agents by an agent communication language."

This definition restricts agents to software agents, limiting the environment to other agents. Interaction is limited to employ a specialized language, the *agent communication language* (ACL). Each agent hides its functionality or task from the environment by only defining his relevance from its role. The autonomy refers to the agent's ability in the sense, that it is not directly dependent on other agents or human users for providing its service or fulfilling its task. It may, and normally does make use of the service provisioning of other agents for his own working.

Single-agent systems, like expert shells or buying assistants, are excluded by this definition from being an agent, because those agents don't communicate as extensively as is required.

On designing and analysing multi-agent systems there are two important levels to discern: the single agent level (micro level) and the agent community (macro level) as discuessed now.

5.4 The Single Agent

The modeling of a single agent is defined by an architecture. That architecture equips an agent with basic features and the necessary integration mechanisms. The most important functionalities are artifical intelligence (AI) capabilities to endow the agent with autonomy. Other basic functionality according to the above agent definition is capability for interactions with other agents.

5.4.1 Knowledge Representation and Reasoning

A lot of agent systems are knowledge based systems that operate on symbolic representations. Those systems have a long tradition in the area of AI, they are well-researched and widely employed. [Bib93]

The environment is modeled by symbolic formalisms, often based on predicate logics. Knowledge is notated as objects, classes, attributes, relations, and so on. The advantages of symbolic representation lies in the precise syntax and semantics, and in the enabled reasoning mechanisms.

5.4.2 Planning and Deciding

This heading subsumes all processes that select actions for execution based on the available knowledge. The most simple form is a pure reactive approach. If defined patterns become true, a specific action is triggered. [Bro86] Normally, mechanisms are present that coordinate multiple possible actions, that sequentialize or parallelize actions, and those that can abort actions.

Good efficiency and high speeds are advantages of this approach, as well as flexibility upon changes in the environment. These makes the approach suitable for mechanical robot systems. On the bad side, a pure reactive approach isn't very well suited for complex or pro-active actions, because all possible actions are pre-programmed

The opposite of the pure reactive architecture is the deliberative planning approach for creation of sequences of actions. [FN71] There is ongoing research in this AI-domain, and new practical approaches are being persued. [RN95]

Complex tasks can be solved with deliberative planning, consisting of large amounts of single interdependent steps. For each specific problem the steps can be combined anew. However, those plans don't adapt easily to changes in the environment, new plans have to be generated often, and the ongoing execution of a plan has to be modified.

Hybrid architectures merge both approaches to get the advantages of both. There is a deliberative planning for plan generation, but rules enable dynamic reactions to a changing environment. For this, the deliberative and reactive modules of an agent act in parallel, but there emerge new problems dealing with the coordination of that modules and their results.

5.4.3 BDI

A widely-known method for modeling a single agent is the belief-desire-intention theory (BDI). That three attributes describe the state of a single agent. The beliefs include the agent's knowledge about its environment, goals are general top-level and abstract items, whereas intentions are short-term action triggers. Possible actions are structured in plans that consist of simple actions, sequences, or whole trees of actions, sub-intentions, or sub-plans. Figure 5.1 shows a representation of the inner workings of a typical BDI agent.

This modeling approach is attractive due to the formalism of the relations between the attributes. The rationality of the intentions can be formally

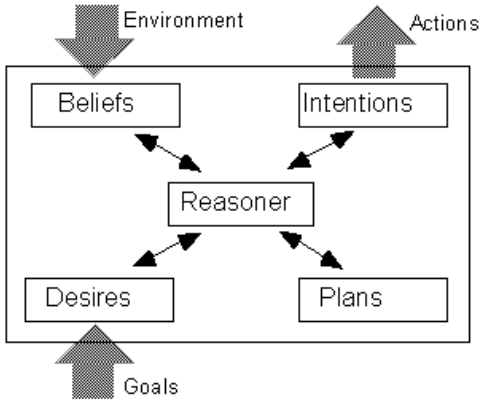

Figure 5.1: BDI Agent

deduced from the given beliefs and desires. [CL90] There is a close relationship to classical AI terms of knowledge, goals, plans, etc.

If, in the following, an agent is termed to "know," to "want," to "decide," to "delegate," i.e. it is described as an active entity with cognitive capabilities, that attribution is done according to the principles layed out in this Section.

5.4.4 Learning and Adaptivity

For optimization of an agent, it is desirable to have learning algorithms within an agent. This can be as simple as accumulating knowledge about its environment. That accumulation can even be persued actively by the agent. This knowledge is normally restricted to "pure" facts, and excludes complex items like rules or intentions or desires of other agents. More subtle methods, e.g. to deduce categories or heuristics, can be imagined but are not well elaborated yet.

The agent could optimize its own actions. This depends heavily on the internal structure and its AI-methods. For example, the agent could reuse plans, change rules, or optimize its search algorithms.

Especially agents representing a user should adapt to the personal preferences, interests, and habits of its owner. This could be done either by spying on the user's action, or by interaction with him by questioning and rating of responses. [ELG99]

5.4.5 Mobility

Agents are per definition distributed systems, and it is sometimes desirable for an agent to change its physical position in the network. Such an agent can optimize its communication with an other agent by migrating to the vicinity of that other agent, which might reduce costs and speed up interaction. [HCK95]

5.5 Agent Community

Whereas techniques of computer science and artifical intelligence are used for dealing with single agents, the organizations of multiple agents as a whole system is coped with by applying social science theories. The areas of communication, coordination, and cooperation are looked into more detail now.

5.5.1 Communication and Speech Act Theory

Communication is the fundamental, and normally sole, mechanism for interaction between agents in a multi-agent system. In contrast to traditional data exchanges, software agents employ a specific language, the "Agent Communication Language" (ACL). That language abstracts from data types, and shall enable a communication between different systems easily. ACLs are based on speech act theory. [Sea69]

According to the speech act theory, there lies a fundamental dualism in human communication: Communication is exchange of information, as well as each communication has an underlying intention of issuing it. That dualism manifests itself in speech acts, which has three aspects: [Aus62]

Locutionary act: The act of speaking itself, including articulation and grammar.

Illocutionary act: The communicative function, e.g. if it is a question, a proposition, etc. It encompasses the propositional content.

Perlocutionary act: The intended effect on the recipient.

As an example: "I want water" has the locutionary act of uttering the syllables. The illocutionary act depends on the context: in a bar, it is the

request to the bar keeper for a drink; starving in a desert it is a wish for heavenly intervention. The perlocutionary act in the bar is to effect the bar keeper in such a way, that he will serve a glass of water; in the desert, the perlocutionary act is void.

There are two important implementations of speech act theory in the agent domain, KQML and FIPA, which will be described in the following.

KQML

The "Knowledge Query and Manipulation Language" (KQML) is part of the "Knowledge Sharing Effort" (KSE) project of DARPA. It defines syntax and semantics of speech acts for inter-agent communication. [DAR, LF97] The KQML-specification only covers the speech acts and their communicative role, but doesn't define the contents.

A KQML-speech act consists of a performative describing the illocutionary act, and a list of parameters being a set of 2-tuples (tag, value). Those parameters contain the sender, the receiver, the content language used and the ontology (see below) needed for interpretation, and the propositional content.

A set of pre-defined performatives exist for the areas of information exchange, conversation, and networks. Extending the set of performatives is possible. Examples are `TELL` for sending facts to the receiver, `ASK-ONE` for an information request, or `REPLY` as an answer to a request.

The meaning of the performatives is defined by a virtual knowledge base (VKB). Each agent acts as if his communication references an internal VKB of facts and goals that can be requested, validated, or modified. A formal approach for a definition of semantic based on knowledge and goals of an agent exists. [LF94]

FIPA ACL

The FIPA ACL is Part 2 of [Fou97]. Similar to KQML it does not define the content aspect of the speech act. The syntactical structure consists of a speech act type and is similar to the KQML performative. A list of parameters follows, the important parameters are the same as in KQML: sender, receiver, content language, content, ontology.

The set of speech act types has similar functionality to KQML, though the syntax more closely resembles actions than information exchange as in KQML. Examples are `INFORM` for sending facts, `QUERY-REF` is a question for information, `REQUEST` is a demand for action.

The semantics of FIPA speech act types is defined by a formal logic called "Semantic Language" (SL). It is a multi-modal logic, based on first order predicate logic and has modal operators for beliefs, uncertainty, and intentions, extended for actions. The representation of that states within the agent is not specified.

KIF

Because both, KQML and FIPA ACL, don't specify a formalism for the content of a speech act, a corresponding language is needed. The most widely known is the "Knowledge Interchange Format" (KIF), stemming from the KSE project as well. [GF]

KIF is specifically designed for communication between agents, and not suitable for internal representation or interaction with the user. Instead, it is very encompassing, so that as much knowledge representations can be transformed into and from KIF.

The syntax is like LISP, the semantics is derived from predicate logic. KIF specifies four types of expressions: terms, sentences, rules, and definitions. Terms declare objects of the world and are composed of constants, variables, functions, and relations. Sentences are logic formulas expresing facts of the world. Rules define inferences to be used on sentences. Definitions give semantics to constants, partial definitions are allowed.

Ontologies

Ontologies are used to define a common vocabulary. An ontology is a set of definitions for basic expressions in a domain. It comprises of categories, objects, attributes, relations, and constraints.

5.5.2 Coordination

Coordination is the effective organization and control of activities of a group of agents. It is important for resource allocation or synchronization, it reduces redundancies and overload situations.

Coordination can be achieved on different levels and with different mechanisms. The mechanisms are standardization for pre-defining patterns of interactions, centralization of control, where one agent supervises the rest of the multi-agent community, and mutual agreement of agents in a flat organization. The different levels of coordination are described in the following.

Organizational Structures

One needs organizational structures that constitutes the multi-agent system. They define the comprising agents and the communication channels. This requires services for agent registration and deregistration, a defined address space, and location services.

The FIPA defines three agent management services needed for an agent platform (AP): [Fou97]

Directory Facilitator (DF): The DF is an information agent sharing information about registered services.

Agent Communication Channel (ACC): It is the standard communication channel between agents on an AP and between APs.

Agent Management System (AMS): The AMS is the actively managing entity on an AP. It controls the life cycle of agents and resource usage, including the ACC.

On a higher abstraction level, agents can be grouped according to their role in the team-based solving of a specific problem. (cf. Section 5.5.3)

Communication Protocols

Communication is more than a chaotic sequence of speech acts. Instead, it is structured according to the task to be solved. The communication process is regulated in a communication protocol that governs the sequence of steps.

A typical representation of a communication protocol is depicted graphically as state diagrams or sequence diagrams. Logically it can be notated as a set of rules that govern the message exchange.

A simple protocol is a question, responded to with an answer. It can be seen in this simple example that the initiative in an ongoing conversation changes between the partners. Further, speech acts refer to other speech acts.

An important factor in the conversation in a multi-agent system is task delegation, if an agent is not capable of solving the problem at hand itself, or doesn't want to.

The traditional master-slave, resp. client-server communication is a hierarchical conversation pattern. The master has a task to solve, and delegates the task to the slave to solve. There is no initiative for the slave. This protocol is efficient, but doesn't adapt well to a changing environment, because the slave entities must be known beforehand, and solving the task relies on them.

The "Contract Net Protocol" (CNP) solves the distribution of roles in the process of solving a task dynamically. [Smi80] There is no need for definition or configuration of delegations beforehand. It is an important protocol within the agent domain. It defines the solving process in four phases:

Task Announcement: The managing agent, that wants a task to be solved, broadcasts its need for solving a specific sub-task to all known and relevant other agents.

Bidding: Each agent that wants to take part in the problem solving process responds with a bidding message. It includes the part of the problem that can be solved by the respective bidder. Furthermore, it contains information for the managing agent to select from the set of bidders.

Awarding: The managing agent selects from the bidding agents and awards the task to the selected one by a respective message. The target agent responds with a contract establishment notification and works on the awarded task. This might include sub-delegation with an own iteration of the CNP.

Controlling: The managing agent supervises its contracted agents, including receiving result messages, or aborting the sub-task.

There exist several variants of the CNP. For example, if the capabilities and availability of agents are known, one might drop the announcement and bidding phases and instead ask the relevant agents directly. That approach adds to speed and efficiency in the simple case, but reduces adaptivity or, if the standard CNP is used as a fallback mechanism, incurs a communication overhead on the cooperation.

Multi-agent Planning

One way of coordination of multi agents is the generation of plans encompassing multiple agents. The available algorithms differ in how the planning process itself is distributed:

Central Planning: One agent generates the plan for all involved agents.

Joint Planning: Several agents work together on a single plan, often on a shared communication media, typically a blackboard.

Decentralized Planning: Each agent generates its own plan, and then integrates it with all other.

Distributed planning reduces complexity and probabilities of errors for each of the participants, but leads to increased communication overhead and potential for conflicts. With decentralized planning, the local environment can be regarded more dynamically, but centralized planning gives better control of the process. The choice depends on the specific case.

Coordination without Communication

There are concepts that allow for coordination without communication:

Conventions, Rules, Laws: These can govern standard situations for resource allocation, although deadlocks can't be prevented.

Roles, Power: According to human organization structures, the same can be implemented for agents, e.g. authority due to a dedicated role.

Deductions about Rationality: Game theory gives the theoretical fundament for coping with situations, where there is no communication between parties. [vNM44] An agent can reason about the expected behavior of another agent and act accordingly.

5.5.3 Cooperation

Cooperation is the shared effort of agents for solving a common goal. There are several classes of cooperation tasks. Some of the tasks are shown now.

Handling of Conflicts

Conflicts can emerge in several variants: conflicting goals, conflicting plans, conflicting actions, resource allocation, deadlocks, etc.

One instance of a conflict handling protocol is the "Speech Act based Negotiation Protocol" (SANP). [CW92] Two agents have different opinions and try to convince the other party by arguments. The protocol has six phases for thesis proposal, iterative argumentation, compromise finding, and acceptance of an agreement resp. call of an arbitor.

Avoidance of Redundancy

The "Partial Global Planning" (PGP, not to be confused with Pretty Good Privacy) algorithm is used for avoiding redundancy in problem solving. [DL91] Agents generate, manage, and communicate generated plans describing their views and reactions to their specific and local environment. These plans reflect only a partial view on the environment due to the restricted perception capabilities of each agent, the view might even be incorrect. The goal is to construct an integrating global plan, that establishes a consistent and correct view on the environment.

Therefore, the agents communicate their partial plans to agents in the neighborhood. While integrating, inconsistencies and redundancies are discovered and synergies are made use of. The successor of PGP, the "Generalized PGP," makes explicit use of generic relations in coordinated plans, specifically subsumptions, causality, temporal constraints, and blocking. [DL91]

Other Methods of Cooperation

Other cooperations between agents exist as well, demonstrating the interdisciplinary character of AOT.

There are a lot more cooperation protocols for resolving conflicts between agents. Those protocols are based on the exchange of hypotheses or proposals and subsequently discovering conflicts. [CML86, DM90, BF95] Weighting and balancing information, finding compromises, and inclusion of arbiters in multiple iterations is sometimes covered as well. [Syc88]

If conflicts arise about resources that can be quantified, general market mechanisms can be employed. A "price" can be found based on available and requested resources. Decision theory or game theory often apply as well. Cooperation protocols exist that are modelled after auction procedures, traditional and dutch auctions being prevalent. [Fou97]

5.6 Agent Architectures

An agent architecture defines the principal construction of an agent and agent communities in several dimensions. The description and construction of an agent within an architecture is either done by its functional components, or by the involved layers of information processing.

5.6.1 Component View

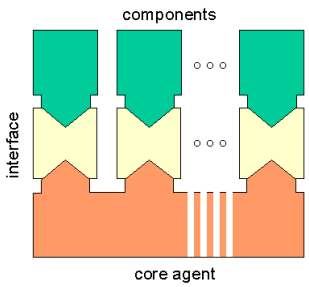

Figure 5.2: Agent Component Plugging

On the structural level, an architecture defines the generic building blocks of an agent. The magnitude of operations of an agent can be implemented by

dedicated modules that fit together by role-specific interfaces, as is depicted in Figure 5.2. [Ses02]

Agents have an aspect of information processing. This includes information filtering, storage, and processing. The components of an architecture defines how this is done within an agent.

The architecture defines the Application Programmers Interface (API), i.e. how the functionality of an agent can be employed by a developer. An agent architecture should provide a rich set of functionalities in libraries or components, all of them being reusable, for easy agent creation. Applications can then be constructed simply by appropriately combining agent funcitonalities and, if necessary, adding application specific components individually. One must take care, though, that enriching the architecture with context-dependent information might overload it and either adds unnecessary burden or restricts the architecture to a specific problem domain, making it less useful in the general case.

Agents are widely classified according to their information processing model:

Reactive Architecture: Signals of the environment are directly put into action, without any cognitive ability. No explicit representation of the environment is necessary here. These architectures are well suited for real-time applications, as is necessary in robotics. Actions, though, are simple and there are no long-term plans.

Cognitive Architecture: Those enable an agent to learn, plan, reason, etc. Local intelligence is an aspect of these agents. Because of the complexity of processing response-time is problematic.

Hybrid Architecture: These combine reactive and cognitive aspects into one architecture. They enable fast responses and allow for complex planning. They are prevalent in the DAI domain.

BDI Architectures

BDI architectures, i.e. architectures, where the single agent is modelled after the BDI model, (cf. Section 5.4.3) are subject to a puzzling contradiction. In theory, those architectures rely heavily on cognitive abilities, thus the resulting architecture should be termed a cognitive one. The reality is different, though. The normal BDI architecture today is rather rule-based, and the way from "perception" to action is very short. Hence, they can be considered as a reactive architecture.

There exist several instantiations of BDI architectures. They differ in several aspects, like the mode of communication, being simple data exchange up to complex speech acts. Management of their belief databases ranges from simple addition of new facts over consistency ensuring mechanisms to complex learning. The structure of plans is another distinguishing factor, they can be simple action secquences, or they might support sub-goals, or even "active" communication between plans. The logic of knowledge processing, i.e. the inference and reasoning mechanisms, aren't fixed.

5.6.2 Layered View

As an alternative to the modul-oriented view of an agent as described above, an agent can be described by its layered hierarchies. An agent then composes of several layers. On each layer there are mechanisms for processing the knowledge on that layer. The structure of an agent is less component-oriented as is for those architectures, but instead a conceptual differentiation is done.

With increasing abstractions, the mechanisms get more and more powerful, but more complex as well. On the lowest layer information is restricted to pure sensoric in- and output, hence is fast. This enables such an agent to react quickly to well-known situations. On a higher level, knowledge is represented and processed. Long-term goals are persued by creating plans and following intentions. On an even more complex layer, global activities take place: A comprehensive world view is persued, actions and plans of other agents are derived, starting of and participation in cooperation is realized.

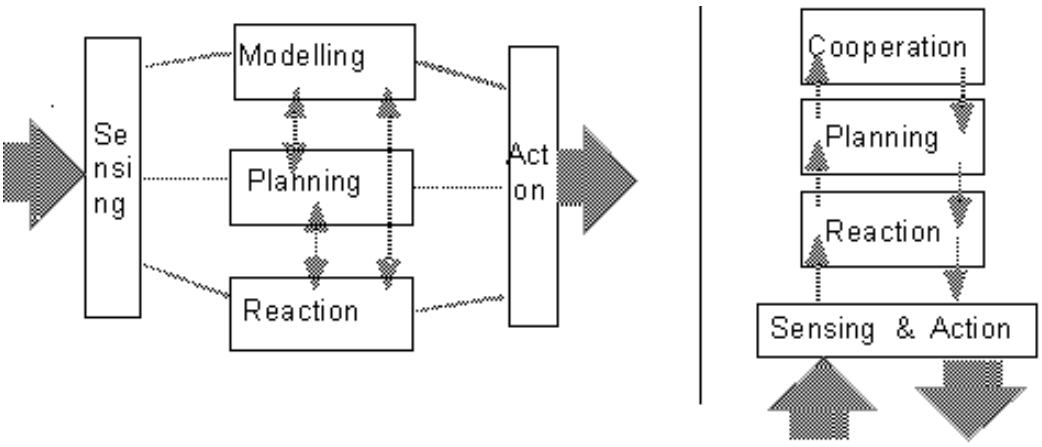

Figure 5.3: Layered Agent Architecture View

Figure 5.3 show two different approaches to the concept of layered agents. On

the left side, each layer has direct access to the sensors and actors of the agent. The layers work in parallel, control is achieved by agent-global control rules. In the depiction to the right, information is transported vertically up between the layers, each with a layer-local knowledge base and specific mechanisms thereon. Control is exercised in the downward direction to implement the consequences of the actions of each layer by using the functionality of the lower layer.

Layered architectures have the advantage of a comprehensive approach for solving a wide range of tasks, ranging from pure reflex-like reactivity to cooperative problem-solving in a complex agent community.

5.7 Agent Platforms and Toolkits

An agent platform is an implementation of an agent architecture, which is merily a concept. A platform comprises of all necessary components or libraries to build a simple agent, mechanisms to add application specific funtionality, and an execution environment, often called agent place.

If a platform is enriched with supporting tools that enable easy and rapid development as well as sophisticated and interactive management capabilities, one speaks of an agent toolkit.

Specific platforms and toolkits are more closely dealed with in Section 6.2, where they are analysed according to their security functionality.

5.8 Pitfalls of Agents

The mentioned notions of agents are often based on the specific interests and intentions of the declaring party. Today, the stated attributes are rather reasearch goals than results.

Sometimes, reading statements about agents today sound overwhelming: “Intelligent agents represent the next generation beyond object oriented software: objects that think. Intelligent agents are task-oriented software components that have the ability to act intelligently.” [GW94]

Due to those exorbitant claims and anthromorphisms, the term "agent" is often reduced to marketing hype. The tagging of agents as having intelligence, emotions, and/or social competence are effectively used to further the sale of a product.

It is important to keep a clear head about agents in the face of euphoric statements. In [WJ98], the authors give common pitfalls for employing multi-agent based systems. That article can be used as a checklist, when and if to employ agents for solving the problem at hand. It can also be used to analyse a presented agent-based system for its reasonability, similar to the Snake Oil FAQ [Cur98] in security issues. (cf. Section 3.3.4)

5.9 Summary

Though there is no common notion of the term "agent" in general, there should be an informal agreement about the underlying and represented idea. Agents have several properties that seem to make them a valuable addition to the service provisioning infrastructure of the future.

As a single agent, they feature autonomy, can be used to create open systems, allow mobile code natively, amonst others advantages. The power of a single agent can be multiplied by the creation of agencies containing places where several agents are located that solve problems together, making modeling and realizing solutions for several real-world problems easier.

It should be kept in mind, that agents doesn't solve everything just because there are agents. Specifically, the introduction of autonomy and mobile code, opens up challenges in the security domain that must be provided for, before systems that will be trusted by users can be created. This challenge will be tackled in the following Part of this thesis.

Part III

Security Infrastructure for Agents

Chapter 6
Analysis and Design

"The situation is extremly hostile!"
Carmen Ibanez, Starship Troopers

6.1 Synopsis

Within this Part of the work an analysis of existing approaches and technical demands is given. In several Chapters an encompassing security infrastructure will be developed and described. The Part closes with conclusions about the presented ideas and an outlook into further work.

This particular Chapter will look closer into other concepts and products. Different aspects of the architecture will be laid out to be refined in later Chapters of this Part.

6.2 Existing Agent Architectures

In this Section, several agent architectures will be analysed to find, where deficiencies exist and where existing concepts can be leveraged. For analysis of agent architectures, several approaches can be taken: one can focus on the architecture as providing basic services; another approach is to view on the intended application domain of the architecture; the methodology how

to reach an application can be analysed; or the supporting tools can be considered. For this work though, the focus lies on the security aspects of the platforms, which mechanisms they provide.

Not considered are one-shot applications focused on and written for a singular task and never re-used again. Simulation testbeds aren't analysed either, for they are no real agent-system but instead simulations thereof. Cognitive architectures, comparable to expert system, often are limited to being one-agent systems and lack communication features, they are left out of this work as well. Finally, robotic systems and other sub-symbolic architectures are conceived for signal processing and machine control, they are left out, too.

This work focusses on multi-agent architectures, that are re-usable for several applications, and make heavy use of communication. For the proposed infrastructure it is important, that the agent architecture supports components in agents. For communication, it is assumed that the global internet infrastructure is used, based on TCP/IP communication. [Inf81a, Inf81b]

6.2.1 FIPA-OS

The FIPA-OS agent platform [Nor00] is a reference implementation of the FIPA agent standardization effort, originally from Nortel Networks and available for free. [Fou97] FIPA's agent reference model provides the normative framework that governs FIPA-OS agents. Combined with the agent life cycle it establishes the logical and temporal contexts for the creation, operation, and retirement of agents.

The implementation of the FIPA reference model includes a Directory Facilitator (DF), an Agent Management System (AMS), and an Agent Communication Channel (ACC). An Internal Message Transport System (MTS) is used to pass messages within the platform.

A FIPA-OS agent consists of an Agent Shell, a Task Manager, a Conversation Manager, and a Message Transport Service as mandatory components. The Message Transport Protocols and Planner Scheduler components can be substituted. Interfaces to databases are offered as well, as is the rule engine JESS [San01] as an optional component.

With respect to security, FIPA-OS optionally supports use of the Java Secure Socket Extension, [Sun01] which has to be downloaded separatly. It can then transparently be used to provide Remote Method Invocation over an

SSL connection, the certificates are stored in Java keystores. The passwords needed for accessing the keystores are set globally within the Java VM, so every process and agent might access them.

All agents of the agent place share the same SSL external Message Transport Protocol (MTP) resource, managed by the ACC, hence strong authentication within a Java VM between agents is not possible. Further, this requires the existence of an appropriate SecurityManager object to prevent e.g. mobile agents migrating to the platform from reading private data including the private keys stored in the keystores, that is, if the FIPA-OS platform would have supported mobile agents at all.

6.2.2 Grasshopper

That platform emerged from a research project of GMD to a freely available product by VKI++. It is written in Java, and several extensions to the core platform for compliance to FIPA or OMG MASIF standards are available. There is no provision for any reasoning or planning capability, the platform is a pure runtime environment supporting the life-cycle, communication, and mobility of agents.

In Grasshopper, security is differentiated between internal and external security. External security here means supporting RMI and plain socket communication over SSL with the usual properties. Internal security protects agency resources from unauthorized access and agents against each other by user authentication and access control, based on Java mechanisms.

Confidentiality is provided by using an encrypted link, no agent-internal encryption is used. As in FIPA-OS, one certificate and private key is shared per agency, hence all agents are authenticated as belonging to the same user with the same privileges.

Requirements for using SSL is configured globally in the communication preferences, it is not possible to discriminate between secure and insecure services, or different levels of security. It isn't possible either to allow resp. require authenticated communication for some peers, and not for others. A considerable problem is, the agent platforms authenticate each other by the certificates of their owners. Hence, if an Agent A from Alice migrates to Bob's platform, at Bob's platform Agent A would be authenticated as stemming from Bob! "Different agents from different owners and creators can travel through the same authenticated secure socket. However, in typical applications this can be considered as a minor drawback." [IKV] In the en-

visaged typical applications as layed out in Chapter 2, this can be considered as a *major* drawback. There is no support for certificate revocation.

For internal security, the AccessController object of Java is used. Permissions for method calls are based on the code-base of the program, which is mapped to the subject's identity and checked against permissions. The concept of Java's protection domains and class loaders is employed. (cf. Chapter 4)

6.2.3 Voyager

Voyager from ObjectSpace Inc. used to be called an agent platform but has been rebranded as a product suite comprising of an application server with Enterprise Java Beans, an Object Request Broker, and value extensions. Such an extension is SSL for secured communication. It is only available as a paid-for commercial product.

Internally, Voyager extends the Java core security functionality of class loading and security domains by offering privilege delegation. Further, it supports multiple policies to be existent at the same time for one Java virtual machine. [Obj00] Access control lists for internal operations, based on simple username/password authentication, are stored at and retrieved from a remote directory server. Policies can be changed during runtime.

Voyager doesn't qualify for an agent system as outlined in Chapter 5. It merely provides for a communication and runtime framework for objects, incidentally allowing remote execution without true migration. There is no expressed life-cycle support or any conformance to agent standards. Merely none of the typical agent's attributes are fulfilled. It is discussed here for its enhanced Java-based security features and due to it being well-known.

6.2.4 Ajanta

Ajanta is a Java-based agent architecture. Its main focus is on security aspects. It offers security for agent migration, RMI, and TCP communication. The Java Security Manager objet has been extended. They have read-only containers and append-only objects for logging purposes. Resources are protected by proxy objects. [Kar]

With respect to the formerly mentioned architectures, Ajanta puts the most emphasis on security. Unfortunately, it lacks in other aspects like an extensive supporting class library or tools for development and platform management.

6.2.5 CASA

The agent architecture CASA uses a complex agent model and integrates agent mobility. [Ses02] Agent gather on "electronic market places," not to be confused with the common recognition in the area of electronic commerce, (cf. Section 2.4) but rather modelling FIPA agent places. [Fou97] These places hold service provider agents and potential service user agents migrating thereto. Dedicated agents provided by the architecture handle agent place management and assist in communication.

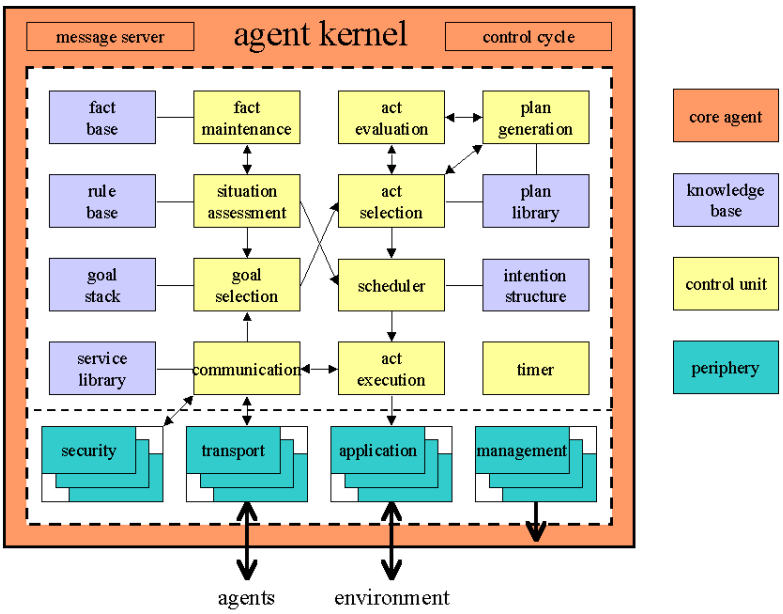

Figure 6.1: CASA Agent Kernel Default Architecture

The CASA architecture, [Ses02] depicted in Figure 6.1, is a modular one; the components are realized as Java Beans. Components can be exchanged, added, or removed at runtime. This is achieved by internal message brokering based on roles fulfilled by the components. That concepts easily allows to add new roles and components to the basic architecture and makes it very extensible, adaptable, and scalable. Capabilities of agents are represented in scripts, with pre- and postconditions allowing for basic reasoning and planning mechanisms. Agents offer services to other agents. Their knowledge and capabilities is represented in ontologies by the descriptive language "CAL." A brief introduction to CAL is given in Section A.1.

CASA is a reactive and BDI architecture in one, as can be seen in Figure 6.2. [Ses02] It reacts on incoming speech acts, but can have long-term

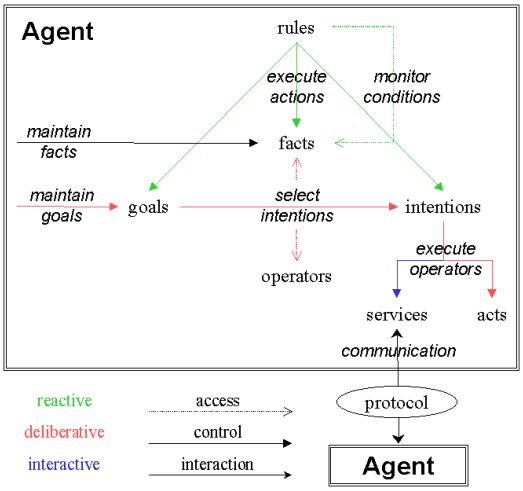

Figure 6.2: CASA Knowledge-based Behaviour

goals worked on by a planner component as well. An implementation of CASA is the JIAC IV agent system. [FBK+01]

6.2.6 Others

There are other agent architectures spuriously heard of. Telescript from General Magic is one popular example and has been, at the time of the inception of Java, a serious competitor to it for remote code execution. [Jos95] Today, it doesn't play any role anymore and even can't be found at General Magic's Web page, neither can its successor Odyssey. [Mag]

Other once-important and often mentioned architectures in the literature, though without putting much emphasis on security, have been discontinued from any serious further development efforts and aren't discussed here. This pertains particularly to Aglets from IBM, [Tok00] Zeus from British Telecom, [plc99] D'Agents (formerly known as AgentTcl) from the Dartmouth College, [Geo00] and Mole from Universität Stuttgart.

6.3 Certificates

Central to most of the security mechanisms proposed in the following is the handling of certitficates. They will be used for message authentication, for authorization, for class loading, for signing contracts, accountability, non-repudiation, etc.

The security infrastructure therefore will have to have mechanisms for dealing with certificates. Each agent will have a component able to deal with them. There must be an infrastructure that supports creation, distribution, validation, and revocation of certificates. Based on the current state of technology, X.509 certificates will be used.

The functionality will be realized by a dedicated certificate management component. It is responsible for storing, receiving, sending, validating, and checking certificates and certificate revocation lists.

6.4 Communication

Communication between agents and agent platforms should allow for all current security attributes. This is in particular authentication, confidentiality, and integrity including protection against packet re-insertion. (cf. Section 3.8) Figure 6.3 gives an overall view of the communication security features of the proposed infrastructure, showing the way an incoming message has to pass before its agent-internal processing commences.

Hence, the security architecture must provide for appropriate means. This will be done on several levels. On the ISO/OSI communication layer 4, standard mechansisms will be employed, namely a Secure Socket Layer (SSL) [FKK96] library will be provided. That standardized protocol allows for authentication, confidentiality, and integrity. It is restricted to peer-to-peer communication.

To allow for advanced message processing of encrypted packets between agent platforms on the ISO/OSI application layer SSL is not usable because it encrypts everything in the packet making processing even by platform agents impossible. Instead, a specfic agent component will be provided that enables an agent to employ security mechanisms on the speech act layer. Leaving

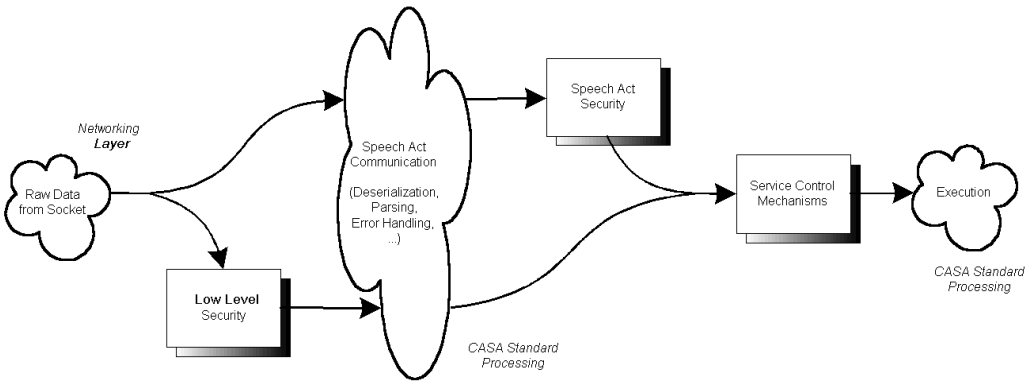

Figure 6.3: Communication Security

information like the sender and receiver of the speech act visible allows the AMS and ACC agents to process the speech acts according to the specific needs at that time without opening the content.

The "interface" of an agent to other agents is the set of services it provides. Therefore, all use of services must be restrictable to only authorized entities. (cf. Section 8.6) SSL alone doesn't give means for authorization. The new agent security infrastructure therefore will provide a specific agent component for access control to services.

6.5 Mobile Agents

Special considerations have to be taken for supporting mobile agents. Upon there migration to remote agent places, it must be cared, that they don't leak information, neither on the communication path nor upon remote execution.

Securing the migration of mobile agents can be provided by means already described: it is treated as a service, hence all mechanisms shown for securing services can be applied to roaming agents. (cf. Section 6.4)

It is very hard though to secure the mobile agent against malicious executing host systems. There are currently no known ways of totally securing a software against its executing environment. The best results in this topic today enable to securely compute a polynom. [ST98] That expressive power is not sufficient for the application domain discussed here.

Rather, the proposed architecture will enable agents to use trust relation-

ships between agent platforms. Further, selected knowledge of agents will be protected against misuse.

The notion of short-time certificates will be introduced to allow for only a short interval for compromize of the respective private key. By using extensions of the X.509v3 certificate format, the length of a certificate chain will be restricted to reduce the amount of possible damage that can be done with a hijacked certificate respectively its private key; this guards against mobile agents running havoc as well. Further, each agent will be enabled to cryptographically sign and encrypt selected knowledge elements.

6.6 Intra-Agent Security

Agents are recognized and modelled as one homogenous entity. There is no competition or maliciousness within an agent against itself. Therefore, the proposed infrastructure will not explicitly provide mechanisms for securing parts of the agent against other parts of itself. The platform as will be chosen below for the design and realization of the system, though, supports screening the components within an agent against each other. There is even a rudimentary access control for internal message passing, but that is rather motivated by programmatic reasons than for security.

This especially means that from the security point of view the knowledge base of an agent is one non-partitioned data pool. Further, each component within an agent has equal rights, there are all in the same security domain. (cf. Section 3.9)

An agent must have means to determine, which security components are available, further which algorithms and parameters are supported, and which configuration to use for a specific communication. Mechanisms will be provided by the proposed security infrastructure.

6.7 Platform Security

As has been defined in Chapter 1, this work does not deal with securing the executing host itself, except for threads by agents. There are two different environment situations though, that must be brought to awareness as a basis for the further discussion. (cf. Chapter 2)

In the first environment, the execution platform is based on a host system at a service provider, retailer, etc. It is assumed, that the agent platform there is a part of the complex information technology infrastructure of that party. Therefore, the platform is governed by the usual policies of that institutions and is assumed to be running in a secured environment against low-layered attacks like network stack bug exploitation, tampering of local code, etc.

The second scenario which is to be mentioned is the use of the agent platform at an end user's machine, either on a desktop, a laptop, or an arbitrary mobile execution platform. There can be no preassumption about the security of that systems, which are commonly regarded as completly insecure.

Due to these two circumstances, it is valid to neglect some security mechanisms like protecting the communication of components of the agent platform against each other. In the case of the provider-based execution host, the machine is assumed to be secure by corporate means. In the case of the user's machine, security mechanisms wouldn't be any good due to the inherent insecurity of the rest of the machine.

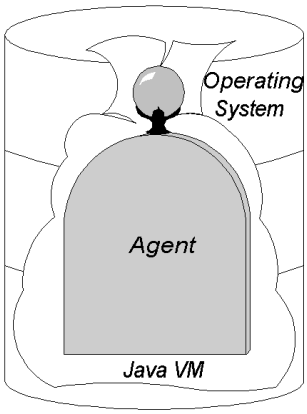

Figure 6.4: Protection Layers

On the other hand, there must be protection mechanisms for the agents

against each other running on the same platform. Further, the executing host itself must be guarded against the agents. All this is provided by a Java execution environment and specific extensions to it as will be developed. (cf. Chapter 4) The layered protection approach is depicted in Figure 6.4.

6.8 Implementation Specifics

For the prototypical implementation of the proposed security infrastructure, some practical aspects have to be regarded.

One must choose a programming language and execution environment. As can be deduced from Chapter 4, the Java environment will be used. This is for several reasons. The programming language itself already has some features to make the code execution safe, if not secure. The library is powerful and provides for all necessary means. Having an already existing cryptographic library is very helpful as well. The execution environment provides for good ways of protecting the host against the executing programs. The security domain execution model helps as well protecting the agents within a Java machine against each other. Java code as being portable over different machine architectures enables implementing agent mobility easily.

To have a complete Java runtime and development environment though, an implementation of the JCE is needed. (cf. Section 4.4) Due to its completeness and availability without charge for educational purposes, the IAIK library has been chosen. [Ins99a] It is a complete re-implementation of the Java JCE API, available outside the United States of America, and has some more functionality: the library offers X.509v3 certificates and extensions, certificate revocation lists, ASN.1 parsing, PKCS data types, several message digest, encryption, and signature algorithms.

Another product put into use for a prototypical implementation of the CASA architecture as enhanced by the security mechanisms of this work by the same provider is the ISASILK library. [Ins99b] It is a Java implementation of the SSL protocol, which will be used for secured communication on the transport layer.

6.9 Summary

As an implicit reference model, the FIPA notion of an agent system is used. [Fou97] For the platform to enhance, the CASA system is chosen due to its flexibility and possibilities. [Ses02] Code excerpts and implementation details are given for enhancing the implementation of CASA called JIAC IV. [FBK+01] The specification language on the agent level is CAL, the "CASA Agent Description Language." In the following ontologies are given in that description language.

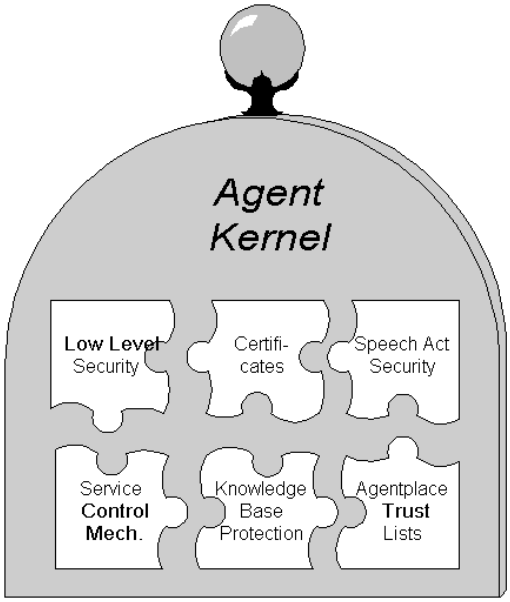

Figure 6.5: Security Components

The next Chapters will, in a bottom-up manner, build up a comprehensive security architecture for agents in the telematic domain. The proposed security infrastructure will consist of security mechanisms on several layers. The lowest layer is the Java implementation language, where thread and execution security will be provided. Agents will be enriched by specific components as shown in Figure 6.5, they can be found within the agent as was depicted in Figure 6.1. Those components help to ensure the security of communication on different abstraction layers and authorization will be provided. The Manager Agent will have trust mechanisms to help in keeping the agent place secure against threats due to agent migration. A dedicated Security Agent

will be given, that provides services to assist the agents on the agent place or in the domain. Value-added services in a dedicated agent will be realized as an exemplary implementation of e-commerce supporting functionality. A certificate authority will be shown, that supports the advanced certificate features which are used in this work.

Chapter 7
Basic Security

"How can an eight year-old boy
who can barely multiply
be a threat to national security?
And people call me paranoid!"
Fox Mulder, The X-Files

7.1 Synopsis

This Chapter deals with very basic mechanisms to be employed. The first is the security of the executing host platform itself. It is shown, how the system can be protected against malicious code executing on the platform, based on the design decision to use Java as the execution environment.

A second important point for all agent systems is communication. This Chapter names a basic way to provide all agents on an agent place with a secured point-to-point communication channel on the transport layer (ISO/OSI layer 4), based on the standard SSL as was introduced in Section 3.8.

Central to all questions pertaining to authentication are certificates. Appropriate mechanisms for handling of them are shown.

Components are used to separate between the functionalities. They are provided for SSL communication and certificate handling. (cf. Chapter 6)

This Chapter does not deal with agents in particular but shows mechanisms

common to all respective systems dealing with mobile code, communication, and securing those.

7.2 Platform Security

The platform in the architecture under discussion is the Java runtime environment (JRE). The system executing the Java runtime environment can be protected against malicious code in the virtual machines as has been introduced in Chapter 4. The exact mechanisms according to this work are described in the following.

The first step towards a secured Java execution environment is to instantiate an object of the class `SecurityManager`. That object is responsible for checking compliance of running code against a pre-defined security policy. The definition of the permissions granted to code is based on either the code base, where the program was loaded from, or by whom the code was signed, or by both attributes.

To easily distinguish between the Manager Agent (MA) and other agents put onto the agent place, the MA should be loaded from its own code base. This eases configuration of the policies and has another advantage: if the deployed base software package has potentially malicious or errorneous classes in components used in the manager agent or other system-related code, and that components are re-used in other user-land agents, than the latter can do no harm despite the faulty code, if they are loaded from a different code base that isn't given the permissions the Manager Agent code base has.

7.2.1 Local Permissions

The class files of the CASA architecture, extended by the ASITA framework, implemented in the JIAC IV toolkit, are to be signed by the issuing entity, which is assumed in the following. In general, the agent platform doesn't have to have any special privileges on the local platform.

The list of permissions in the JDK 1.2 that are checked by the security manager object is given in [Sun98a]. Which of these permissions are granted to what classes in the ASITA framework is given in Table 7.1. A more detailed differentiation according to the target names of the permissions is given in Table 7.2 and Table 7.3 for permissions of the Java type `RuntimePermission`

Permission Type	Grant
`AWTPermission`	this is only needed for local classes, that use a GUI
`FilePermission`	only needed for storage-related classes, e.g. for persistence, logging, configuration retrieval
`NetPermission`	should not be set
`PropertyPermission`	read and write for the Manager Agent, other components only for properties restricted to their own name and signed by a trustworthy software supplier
`ReflectPermission`	isn't needed
`RuntimePermission`	for individual target names see Table 7.2
`SecurityPermission`	for individual target names see Table 7.3
`SerializablePermission`	only needed, if in a particular implementation of the ASITA framework the features available through subclassing the `Object*Stream` is used
`SocketPermission`	needed to allow for autonomous communication between agents

Table 7.1: Granted Permissions

Target Name	Grant
`createClassLoader`	the Manager Agent needs this
`getClassLoader`	the Manager Agent needs this
`setContextClassLoader`	the Manager Agent needs this
`setSecurityManager`	is not needed by any agent, because the `SecurityManager` is set before the Manager Agent starts to put the latter under the control of the former
`createSecurityManager`	dto.
`exitVM`	needed only for the Manger Agent to gracefully shutdown the agent place
`setFactory`	not needed for any component including MA
`setIO`	reasonable for the MA
`*Thread*`	not needed (cf. Section 7.2.2)
`getProtectionDomain`	needed for the MA
`*FileDescriptor`	needed for the MA
`queuePrintJob`	possibly needed by any logging facility

Table 7.2: Granted Runtime Permissions

Target Name	Grant
`*Provider`	useful for dynamic aspects of Security Agent (introduced in Chapter 9); potentially harmful due to introduction of insecure (cf. Section 3.10) or unlawful algorithms (cf. Section 2.6)
`*Identity*`	useful for MA to support inherent Java functionality, but that is not used in the ASITA framework

Table 7.3: Granted Security Permissions

and `SecurityPermission`, respectively. No permissions are needed for the system for all Java permission targets not shown.

With the current implementations of the JVM the execution environment loads the policy definition files only once. Afterwards, the policy definition can not be changed. To be more precise, the policy for instantiated objects can not be changed. Hence it is not possible to exchange the policy during runtime of the agent place. This would either call for a completely separate implementation of the security mechanisms for the agent system implementation and then having to attend both security architectures in parallel, leading to an overly complex and very hard to maintain system.

Alternatively, one could modify the Java Virtual Machine implementation, but that would make the security system incompatible to the rest of the JVMs in the world, rendering the overall goal of communication of open systems based on the all-available lowest common denominator pointless. This deficit has to be overcome in future iterations of the ASITA infrastructure, or the Java security architecture. [Gon98]

7.2.2 Thread Group Separation

For securing the host system against the agent system, and the Manager Agent of the CASA architecture against the running agents, the Java mechanism of thread groups is employed. Each object runs in a thread, where threads can be grouped together in a thread group. By enforcing relationships between threads they can be secured against tampering between each other.

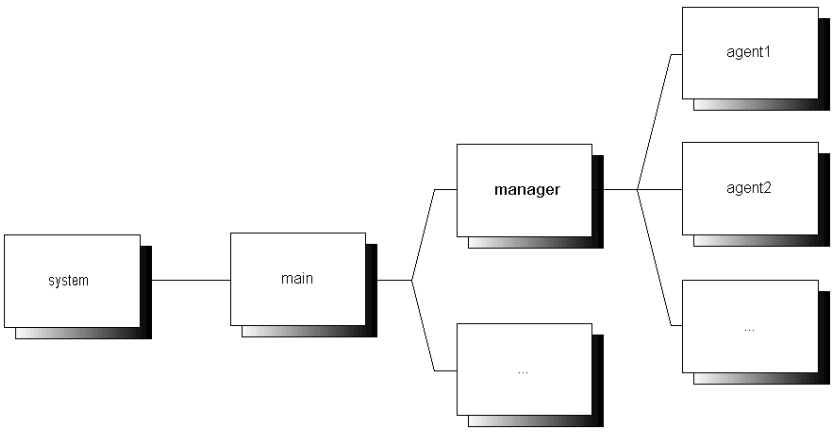

Figure 7.1: Thread Group Hierarchy

Specifically, upon creation of the JVM, the thread group `system` is initialized. That thread group creates as a decendent the thread group `main`, in which the `main` method of an application is run. That method will then create the thread group `manager`, that runs all threads of the Manager Agent. Each agent newly instantiated by the MA will be put into its own thread group decending from the `manager` thread group. (cf. Figure 7.1)

Putting the MA into an own group below the `main` group allows for restricting the CASA marketplace to less power on the local machine than the main application itself. This is of no concern at the moment, where one JVM runs exactly one MA with exactly one agent place, but is a provision for future scenarios or extensions to the CASA framework resp. the JIAC implementation.

Unfortunately, in the default `SecurityManager` class as distributed with the JVM, the access check between threads is very weak. An executed object is allowed to modify any thread group not being the `system` thread group, and any thread that is not in that group. This includes the `main` thread!

Instead, each executing thread should be bound by the limits of the thread group it is running in. No object should be able to modify any thread or thread group above its own in the hierarchy. The corresponding code excerpt from a derived `SecurityManager` class called `AgentSecurityManager` is given in Figure 7.2. Using that code, each object has only access to all objects running in its own thread group or below, but not upwards in the tree as seen in Figure 7.1.

For very special cases not envisaged at the moment, and for being compliant to the Java Security Manager guide, [Sun98b] objects holding the runtime permissions `modifyThread` or `modifyThreadGroup` are allowed to modify any thread or thread group, respectively. For the usual case of a CASA system, neither of these permissions have to be granted. Handling of the case of an object holding the omnipotent `AllPermission` is performed in the `checkPermission` call.

The `setMaxPriority` method of the class `ThreadGroup` does not allow to set a thread group priority higher than it currently is, it can only be lowered. Setting a thread priority via call of `setPriority` is upper bound to the thread priority of the thread group it belongs to. Hence a newly started agent is not able to raise its priority to a higher value than set to by the manager or other classes in the upstream thread group tree. (cf. Figure 7.1) Because the methods are both declared `final` by the standard Java Runtime Environment it is ensured, that no class can override this behaviour. The default thread

```
1  public class AgentSecurityManager
2       extends SecurityManager {
3
4   public void checkAccess(ThreadGroup tg) {
5          // allow parent access
6          if (Thread.currentThread()
7             .getThreadGroup().parentOf(tg))
8                    return;
9          // otherwise require permission
10         checkPermission(new
11            RuntimePermission("modifyThreadGroup"));
12  }
13
14  public void checkAccess(Thread t) {
15         // allow parent access
16         if (Thread.currentThread().getThreadGroup()
17            .parentOf(t.getThreadGroup()))
18                   return;
19         // otherwise require permission
20         checkPermission(new
21            RuntimePermission("modifyThread"));
22  }
23 }
```

Figure 7.2: Security Manager Code for Checking Thread Access

group, and therefore thread priority, is set to the maximum. The Manager Agent should set it to appropriately lowered values upon creating new thread groups for instantiated agents to run in.

This work does not provide for securing the components of an agent against each other, though this could have been achieved with the thread model as well. It is assumed, that the agent is a homogenous entity with respect to the mutual trust of its inner state. This design decision is backed by the strong migration concept of CASA, where complete agents with all state information and classes are transported, instead of only parts of an agent from different sources.

7.2.3 Class Loading

Class loading in Java is handled by subclasses of the `ClassLoader` class. The `SecureClassLoader` as provided by the standard Java Runtime Environment already enforces security as described for threads above: modifications and creation of a class loader is only permitted by classes "up" the tree hierarchy of loaded classes.

Created classes and their instance objects are by default separated into protection domains. A protection domain is all code loaded from the same code source, which encompasses the location of the classes and the associated permissions. The permissions are defined in the system policy as seen above. The authentication of the code is based on certificates that were used to sign the respective classes and objects in the Jar Archive File from which the objects were loaded.

The class loading mechanism of Java is extended by providing for integration of the certificate mechansims as described in Section 7.4 and Section 9.3. Certificates are now more thoroughly checked than in the basic Java Runtime Environment. Specifically, they are checked against CRLs, if being without its validity interval, if the certificate has critical extensions and if they are all known and valid. Classes are only instantiated if the corresponding class object was successfully verified.

7.3 Transport Layer Communication Security

As has been motivated in Section 6.4, allowing for secured communication with non-agent systems is rather straightforward. It is necessary to provide an SSL communication component. These components are readily available on the software market and can be integrated. For the prototypical implementation of the framework the ISASILK library has been chosen. [Ins99b]

The library can easily be added to the runtime platform by adding its JAR file to the `CLASSPATH` of the Java machine. Its use in a Java program is documented on the web page of the provider and is common practise and shall not be described here.

By requesting the specific features, authentication, confidentiality, and integrity will be provided. The integration of SSL on the agent level instead on the Java programming level is described in more depth in Section 8.4.

7.4 Certificates

Certificates are an essential basic security mechanism. They are used to attribute an identifier to a public key, commonly used for authentication. (cf. Section 3.7.5)

If an agent is to be designed as autonomous as possible, the functionalities as described in the following Subsections are to be fulfilled by the agent itself. As an alternative to gain a smaller agent and possible speed advantages due to caching, the complex process of certificate checking can be delegated to a dedicated agent. In the ASITA framework this is the security agent present at agent places. (cf. Section 9.3) If the communication to that security agent can not be trusted, at least the minimal knowledge base information for securing the socket to the security agent must be present.

7.4.1 X.509 Certificates and Extensions

In the proposed ASITA security infrastructure, the commonly used and standardized X.509 certificates are used. [ITU97b, HFPS99] For the enhanced features that will be described shortly, the extensions of version three certificates are needed.

```
// X509v3 certificate
(cat X509Certificate (ext Certificate)
     // X.509 version according to standard
     (version int fixed)
     // id per issuer
     (serialNumber CertificateSerialNumber fixed)
     // algorithms and parameters for this
     // certificate
     (signature AlgorithmIdentifier fixed)
     // issuer of this certificate
     (issuer DistinguishedName fixed)
     // contains notBefore and notAfter
     // category objects
     (validity Validity fixed)
     // identity of entity to whom the certificate
     // was issued according to X.500
     (subject DistinguishedName fixed)
     // certified algorithm identifier and public key
     (subjectPublicKeyInfo SubjectPublicKeyInfo fixed)
     // for directory access control out of X.500 namespace
     (issuerUniqueIdentifier UniqueIdentifier) //version=2|3
     // for directory access control out of X.500 namespace
     (subjectUniqueIdentifier UniqueIdentifier) //version=2|3
     // X.509v3 extensions, described later
     (extensions X509Extension[]) //version=3
     // the DER encoding of the certificate object
     (encoded byte[]) )
```

Figure 7.3: Certificate Category

The mapping of an X.509v3 certificate onto the CASA ontology language is straightforward and given in Figure 7.3. The extensions to add more semantics as introduced in version three of the relevant standard [ITU97b] are given as an CAL ontology in Figure 7.4. (see Section A.1 for a brief overview of CAL) To make use of the extensions, specific certificate extension

```
1 // X509v3 Extensions
2 (cat X509Extension
3      // which extension
4      (extnId x500.ObjectId needed)
5      // invalid if unrecognized
6      (critical bool (default false))
7      // DER encoded value
8      (extnValue byte[])) )
```

Figure 7.4: Certificate Extension Category

fields have to be defined. In CAL this can be expressed as an inheritance relationship. The relevant extension fields will be described in the following Subsections.

The validity period of certificates is of special importance for mobile agents. As a measure against malicious hosts subverting a received agent, the agent will carry only *short term certificates.* These certificates will be created by the spawning static agent with a very limited lifetime. The period of the certificate must be sufficiently large enough to allow the mobile agent to complete its task, but not longer. This duration to decide beforehand is an unsolved problem yet. In the ASITA framework it is encouraged to have this value configurable, either by general user preferences or application-specific ones. A reasonable amount seems to be at around a few minutes to perhaps ten to twenty. If this time is not sufficient, the mobile agent has to signal back this exceptional case, probably with the information gathered so far.

It is strongly discouraged to then automatically generate a new agent with the same task and a longer lasting certificate, because the agent might have been subverted and be tricked into sending this exception message to get hold of a new certificate. Instead, this situation should be brought to the attention of the user to decide what to do. If this situation persists in several domains, the timeout period should be carefully raised. If only the current problem solving application is persistently affected, out-of-band communication means should be employed to check for the correctness of the situation.

Exactly *which* of the potentially multiple certificates an agent holds is to be presented or used in a given situation is not pre-defined. Instead, the agent must make an educated guess on using the right one. The `keyUsage` extension can be employed for this guess, (see below) as well as the security requirements which are defined for the prospected service usage. (cf. Section 8.2.2) A try-and-error approach is feasible as well in this context. Previous expe-

rience of the agent with the same communication partner or situation might help as well. For a long-term history covering several instances of a specific agent this requires some persistency mechanisms of the agent architecture.

For verifying a certificate in a chain several certificates must be checked. If one is not present in the konwledge base of the agent, it must be retrieved from a Key Distribution Center. (cf. Section 3.7.5) In the ASITA infrastructure, this role is fulfilled by the security agent, present at the agent community level, and is described in Section 9.3.1.

The security agent will usually provide only those certificates created and distributed by a CA. Agent-related and short-term certificates will normally not be available at the KDC, because the management and communication overhead would be too large. Therefore, if any certificate that is not available at the security agent is needed, the agent holding that certificate has to distribute it to the recipient by itself. This is easily achieved by using a list of certificates as an object type wherever a single certificate seems intuitive at first glance. This approach is often used and can be seen in the SSL protocol as well. (cf. Section 3.8) The list is filled with all certificates in a chain leading to the first certificate that can be retrieved from the KDC.

BasicConstraint Extension

```
1 (cat BasicConstraints (ext X509Extension)
2      // critical for ASITA
3      (X509Extension.critical bool
4            (constr ?X509Extension.critical))
5      // if new certificates may be created
6      (cA               bool (default true))
7      // if yes, length of certificate chain
8      (pathLenConstraint int (default 1)) )
```

Figure 7.5: BasicConstraint Extension Field Category

A `BasicConstraint` certificate extension field as given in Figure 7.5 is used to denote the use of the certificate as either being a CA certificate or not. If a certificate is denoted with this extension as being a CA certificate, i.e. the attribute `cA` is set to `true`, the certificate can be used to verify the correctness of other certificates. Vice versa, it can be used to sign certificates for public keys. A certificate not having this attribute set can only be used for authenticating the subject itself, but no derived CA certificates of a chain.

The path length constraint reduces the usefulness of the certificate: for each length count, one further level of CA certificate chain nesting is allowed.

In the ASITA framework, this extension is considered critical. A user certificate will have the path length set to 1. This allows to generate a new CA certificate with path length 0 for static agents. Static agents will generate therefrom new short term certificates with `cA` set to `FALSE` and give that certificates to newly spawned mobile agents. The mobile agent will then not be able to generate further certificates in the chain, either voluntarily, due to bogus code, or by an attacker spying out the relevant data.

SubjectAltName Extension

```
1 (cat SubjectAltName (ext X509Extension)
2      // unique identifier of using agent
3      (agentName string needed) )
```

Figure 7.6: SubjectAltName Extension Field Category

This extension field is used to bind alternate names to the certificate in addition to those given in the subject attribute.

For the ASITA infrastructure, this field is used to bind the certificate to a specific agent. When the user issues new certificates to its agents, the new certificates still have the original `subject` attribute, i.e. an identifier of the responsible user on whose behalf the agent is acting. To be able to identify the autonomously acting agent, its unique name is implanted in this extension field. The exact syntax and semantics of that name depends on the agent architecture. In the exemplary implementation JIAC IV of the CASA system, this name is formed as `Name@protocol://dns.name.of.host:port[/facility]`.

KeyUsage Extension

```
1 (cat KeyUsage (ext X509Extension)
2      // allowed use for the corresponding
3      // private key, defined in X.509v3,
4      // 12.2.2.3
5      (usage int[] needed) )
```

Figure 7.7: KeyUsage Extension Field Category

The `keyUsage` X.509v3 extension is an important way for an agent to figure out, which certificate resp. corresponding private key to use in a given situation. It can be used to distinguished between various different uses. [ITU97b]

In the ASITA framework, it is encouraged to *not* set this extension to being critical, because this field is aimed at being a *hint* to the using agent, instead of being a *constraint.* On the other hand, an agent must be prepared to receive a failure notice and take fallback measures because of the presented certificate misses the critical flag and has been used in a non-conformant action due to differing security policies in other domains.

7.4.2 Certificate Revocation

Certificate revocation is an important mechanism for invalidating certificates. It is inherent to the relevant standards. [ITU97b, HFPS99] Astonishing enough, this essential part of the standard and crucial component of a functioning certificate infrastructure is usually neglected, even in the largest and most popular software distributions and systems. [Gue01] The ASITA framework, though, does provide for CRL mechanisms.

Figure 7.8 gives the category objects defined in the CAL language used for providing the certificate revocation management mechanism, it is a literal translation of the standard. The attribute `crlEntryExtensions` is provided to accomodate for further enhancements of the ASITA framework and is unused in the exemplary implementation. It might contain extensions as defined in [ITU97b] resp. [HFPS99], especially the `issuerAltName` extension might come in handy.

CRLs are distributed the same as certificates by security agents fulfilling the role of a KDC. (cf. Section 9.3.1) In contrast to the certificate case, “normal” agents themselves don’t distribute CRLs, so they don’t have to provide a

```
1  // signed certificate revocation list
2  // per X.509 resp. RFC 2459
3  (cat X509CRL
4       (tbsCertList TBSCertList      needed)
5       // how this CRL was signed
6       (algorithm AlgorithmIdentifier needed)
7       // binary signature object over
8       // DER-encoding of tbsCertList
9       (signature byte[]) )
10
11 // list with revoked certificates
12 (cat TBSCertList
13      (version int (default 2) fixed)
14      // contents is the same as above!
15      (algorithm AlgorithmIdentifier needed)
16      // who issued this CRL
17      (issuer DistinguishedName needed)
18      // begin of validity period
19      (thisUpdate DateTime needed)
20      // end of validity period
21      (nextUpdate DateTime needed)
22      // those certificates are invalid
23      (revokedCertificates RevokedCertificate[] needed) )
24
25 // a single revoked certificate
26 (cat RevokedCertificate
27      // as issued by the CA in the cert to be revoked
28      (userCertificate CertificateSerialNumber needed)
29      // timestamp as of when the cert was invalidated
30      (revocationDate DateTime needed)
31      // extensions for further applications
32      (crlEntryExtensions X509Extension[]) )
```

Figure 7.8: Certificate Revocation Categories

means of distributing them. If an agent caches CRLs for the duration of its validity, communication overhead can be reduced, but this increases the knowledge base of the agent.

7.5 Summary

This Chapter has given low-level mechanisms to secure the executing Java Runtime Environment against malicious code. Further, it propagates using SSL on a Java layer. Certificates are introduced as central aspects for authentication.

The next Chapter will raise the discussion of security aspects onto the agent level.

Chapter 8
Agent Security

"But even the man who has made his own plans,
when he comes to see things with his own eyes
will often think he has done wrong.
Firm reliance on self must make him proof
against the seeming pressure of the moment."
Karl von Clausewitz, On War

8.1 Synopsis

Chapter 8 discusses security features to be found on the agent level and for the communication between agents. As a first measure, the agent must be endowed with self-inspection functionalities. Therefore, an ontology is introduced to describe security-related functionalities and attributes.

Then, means to secure an agent against hosts is shown, particularly a component for securing its knowledge base is introduced. As has been forseen in the last Chapter, (cf. Section 7.3) the SSL protocol is made available on the "conscious" level of the agent.

Because using the SSL protocol has some limitations on its applicability, another component is given that allows to secure communication on the speech acts level, corresponding to the ISO/OSI layer 7, the application layer.

The Chapter finishes with the introduction of an encompassing and service-related authentication mechanism.

8.2 Agent's Security Awareness

The mechanisms of the following Sections will depend on the availability of several components and realized abilities in the agent. To ensure the consistency of the dependencies and to enable an agent to reason about its own composition, thus enabling it to load other components as needed and enforcing security policies, a way for specifiying dependencies between components and abilities will have to be defined.

8.2.1 Object Naming

```
1  (ont SecurityObjects:V_1
2       (cat SecurityComponent
3           // identity tag
4           (identifier     string  needed unique)
5           // descriptive name
6           (name           string)
7           // class name of the implementation
8           (className      string) )
9       (cat AgentAbility
10           // id tag of the ability
11           (name           string           needed unique)
12           // which component implements that ability
13           (component      SecurityComponent       needed) ) )
```

Figure 8.1: Security Object Naming Ontology

As a first step, the elements comprising the security aspects within an agent have to be named. This is achieved by the ontology as given in Figure 8.1 Examples of instantiated objects of that categories can be found in Section A.3.

The `className` attribute enables an agent to dynamically load other Java classes as needed to fulfill the dependencies.

8.2.2 Security Dependencies

```
(ont SecurityDependencies:V_1
     (cat PerAbilityRequirement
          // for which ability the dependencies are described
          (ability        AgentAbility            needed)
          // one of these other abilities must be fulfilled to
          // enable the ability. the order of the abilities
          // gives the preference
          (requiresOneOf  AgentAbility[]) ) )
```

Figure 8.2: Security Dependencies Ontology

As a second step for allowing the agent to manage its own composition of components consistently with security requirements, relationships between the categories of security objects have to be defined. The ontology therefore can be seen in Figure 8.2, examples can be found in Section A.4. The specification is realized as rather canonical mappings of the functionalities onto objects of the fact base of the agent, described by the ontology definition.

An agent's knowledge base is initialized only with objects that represent its loaded abilitites. Upon loading a component that component's initialization function is responsible for adding instances of the security objects and dependencies to the knowledge base of the agent that describe its abilities and dependencies.

There can be more than one `PerAbilityRequirement` object per described ability in the knowledge base of the agent. For each of the objects, at least one of the requirements in the `requiresOneOf` attribute must be fulfilled.

It is not necessary to define specific component dependencies, because each component provides abilities. That abilities are dependent on other abilities that are realized by components. So to check the dependency of a component upon other components it is sufficient to check the dependencies of its provided attributes.

After loading a component into the agent, the agent kernel has to re-check the dependencies and load more components as defined by its updated knowledge base.

8.2.3 Service Security Requirements

```
(ont ServiceSecurityRequirements:V_1
     (cat ServiceSecurityRequirements
          // all abilities must be present within
          // the providing agent
          (requires       AgentAbility{}  needed) ) )
```

Figure 8.3: Service Security Requirements Ontology

```
(ont Services:V_1
     (cat Service
          (name           string (constr
                 (not (comp Strings.equal ?name "")))
                 fixed needed unique)
          (keywords       string{})
          (language       string (default "CAL") fixed)
          (protocols      Protocol[] fixed needed)
          (ontologies     string{} fixed needed)
          (requires       ServiceSecurityRequirements)
          (gui            GUIServiceDescr)
          (scl            ServiceControlList) )
     (cat ServiceInst
          (protocol       Protocol)
          (conversation   string  fixed needed unique)
          (requires       ServiceSecurityRequirements)
          (certificate    Certificate) ) )
```

Figure 8.4: Service Ontology

For use of communication security, the wanted attributes have to be specified in the communication interface of an agent: the description of its services. This is achieved as another dependency relation by the ontology as given in Figure 8.3. All of the abilities named in the attribute set `requires` have to be present in the agent and will be tested against upon service use to allow access the service this requirement is attached to.

This requirement is bi-directional: it defines the requirements for being a service user as well as being a service provider.

In Figure 8.4 it is depicted how the security requirements are hooked into the remainder of the service use and provisioning infrastructure of the agent

system JIAC IV.

8.3 Knowledge Base Protection

The knowledge base of an agent representing its view of the world and holding not only the data to be acted upon in its life time, but holding the results of its work as well is critically endangered, if the agent leaves its home platform. As soon as it is migrating to remote agent places, the integrity of the whole agent is in question, its declarative as well as its procedural knowledge. That implies, that no remote host can be initally sure about the information provided by a mobile agent, nor may the agent's owner know if the data returned by "his" agent to its home platform is to be trusted.

In the base CASA system, the agent provides a management interface to the platform, respectively to the Manager Agent. No other regular access from the Manager Agent onto the managed agent is possible. This is not only a problem in the event of an errorneous agent running rampant, but also from a purely technical viewpoint: an executing host system has complete control over any agent it has received from the network.

If information retrieval or fraudulent behaviour is performed by the malicious host, it has two conceptually different points of attacks at the mobile agent, in ignoring the "standardized" access interface as provided in the CASA architecture framework. The host can either directly modify the data the mobile agent is working with, as well as for input as for output, or it can modify the execution logic (or both).

It was shown in [ST98], that for the general case the result of a polynom calculation on the ring $\mathbb{Z}/N\mathbb{Z}$ of a mobile agent *can* be protected against malicious hosts. Unfortunately, this case isn't very applicable to real-world problems. Even worse, the host system isn't required by any means to execute the code of the mobile agent at all, it can very well exchange code and act on its own behalf. For this reason, there is no real protection for the code of a mobile agent against a malicious host.

In contrast, it *is* decidable for a benevolent hosts if an agent's code has been tampered with on its journey. That will be described in Section 9.2.3 and is not further elaborated upon here in the Section dealing with the protected information in the agent itself, because the agent hasn't any possible way to reason about its own integrity without executing code that's very integrity

is in question.

On the other hand, where an evil host in the telematic domain might be interested in is not the processing logic of an agent, but the data it acts upon and receives from third parties. So practical mechanisms to enhance the confidence in the *data* of a mobile agent are introduced in this Section.

They will be realized by a dedicated agent compontent named `KBP`, according to its function of knowledge base protection. A stationary agent doesn't need the mechanisms as shown and consequently doesn't have to have the respective agent component.

8.3.1 Standards and Ontologies

```
ContentInfo ::= SEQUENCE {
    contentType ContentType,
    content
        [0] EXPLICIT ANY
        DEFINED BY contentType OPTIONAL }
ContentType ::= OBJECT IDENTIFIER
```

Figure 8.5: ContentInfo Type of PKCS#7

```
(ont ProtectedContent:V_1
     (cat ContentInfo
          (contentType    string
                  (constr (comp isElement
                  ?contentType { "data" "signedData"
                  "envelopedData" "signedAndEnvelopedData"
                  "digestedData" "encryptedData"))))
          // opaque BER encoding of corresponding
          // PKCS#7 object
          (encoding       byte[]) )
     (cat    AuthenticatedSafe
             (content        ContentInfo[]) ) )
```

Figure 8.6: Protected Content Ontology

The `ContentInfo` type is defined in [RSA93e], Figure 8.6 gives the corresponding agent ontology for the CAL language. The fields of type `ContentInfo` have the following meanings:

contentType indicates the type of content. It is an object identifier, which means it is a unique string of integers assigned by the authority that defines the content type.

content is the content. The field is optional and if it is not present its intended value must be supplied by other means. Its type is defined along with the object identifier for `contentType`. [RSA93e]

The standard [RSA93e] defines the six content types `data`, `signedData`, `envelopedData`, `signedAndEnvelopedData`, `digestedData`, and `encryptedData`.

```
1 (ont CriticalData:V_1
2      (cat LocalData
3           // may contain any CAL object
4           (data           any       needed) ) )
```

Figure 8.7: Critical Data Ontology

```
1 (matt expires int (ktype fact object goal)
2       (constr (>= ? 0))  // 0: doesn't expire
3       (default 0)
4 )
5 (obj  CD  CriticalData
6       (data   "something important")
7 ):(meta (expires 4321834567890))
```

Figure 8.8: Critical Data Object with Meta-Attribute

Objects of the category `LocalData` may be used to wrap any data object in the dynamic knowledge base of an agent. As shown in Figure 8.8, a meta-attribute declaration might be used to expire the data and direct the agent to remove the knowledge element, if the end of its life time as given in the `expires` meta-attribute is reached. In the given example, that is Monday October 11, 2004, CEST, 06:15:14.

8.3.2 Knowledge Base Division

For securing the knowledge of a mobile agent on its migration path against tampering as much as possible must be cryptographically signed by the sender. Unfortunately, with respect to security, the knowledge base of an agent is ever changing due to its work done. To protect as much of the agent's data as possible, it is proposed to divide the knowledge base of a mobile agent into a static and a dynamic part.

The static knowledge base contains data that will never change for the duration of the agent's travel. This especially includes data about the originating home platform and the certificate of the owner of the agent. This allows host systems on its way to ensure some integrity of the agent's data and to retain accountability of its work. (cf. Section 9.2.2)

The dynamic part of the agent's knowledge base will hold all changeable data the mobile agent starts with, and all data collected on its way.

Especially due to the autonomy of agents it might not be possible to determine beforehand, which part of its knowledge base will change and which will not. From a practical point of view it might very well be the case, that almost all data of an agent will be held in the dynamic part.

The CASA architecture has, in its language definition called CAL, the meta-attribute `fixed`, which inhibits changing of an attribute. The meta-attribute `MetaFixed` disallows changing anything of the object. This is restricted to non-owning objects, those that have object references to it can manipulate the "protected" objects at will. Certificate-based immutability is not supported. The next Sections deal with how to provide cryptographically secured data objects based upon the data types just introduced.

8.3.3 Outgoing Data

For purpose of authentically transporting and sending data to service providers, the sensitive information is distributed in `AuthenticatedSafe` objects, derived directly from [RSA97]. This pertains to static and dynamic knowledge alike, because the receiving system isn't aware of out of which part of the agent's knowledge base the data passed on is derived from.

Such an `AuthenticatedSafe` object is then cryptographically signed and put into a `ContentInfo` object as signed data. [RSA97]

The inner `ContentInfo` objects in the `AuthenticatedSafe` will contain pro-

tected data, as the application sees fit. For sending data from the home platform to a remote host to be processed there, the inner data will either be plain text, if the remote system doesn't support receiving encrypted data as input for its provided service.

Or, if the receiving system is able to process encrypted data, the inner layer of `ContentInfo` objects will be of the data type `envelopedData` containing symmetrically encrypted content, and the symmetric key encrypted with the public key of the recipient. The PKCS#7 standard defining this data type allows to encrypt the same symmetric key with several different certificate-derived public keys allowing to receive several recipients the same encrypted data without having to re-encrypt it once for each component. [RSA93e]

8.3.4 Incoming Data

For a mobile agent to receive authenticated or private data from another agent in the context of a service usage, the data already has to reach the agent in a protected way. The receiving agent itself would not be able to act on the data, because it neither can trust its own data on its current executing system, nor might the respective functions be executed anyway due to code manipulation by the platform.

The sending provider has to act upon the data before it supplies the information to the mobile agent. The treatment of the data is as specified in the previous Subsection. If the data shall reach the destination secured against prying electronic eyes, it has to be encrypted. This is done with the public key of the recipient. In the usual case, the information is to be authenticated as well, so that the service user has the confidence of getting the information from the right source.

For the information provider to have the confidence that it provides the service to the intended recipient, the certificate describing to whose public key the result shall be encrypted must be contained in the service request and must be signed by the requesting user as well. Else, the chain of accountability would be vulnerable by a man-in-the-middle attack by a malicious host exchanging the receiver's certificate in the mobile agent by its own and thus would get the information.

8.3.5 Short Term Certificates

For certain operations like contract signing, the mobile agent needs accountability and hence has to sign data with private key material. This is an obvious problem, because private data isn't private at all on a malicious system. Further, long-term key material is not permitted to leave the home platform of the agent.

For the cases, where despite these security issues an agent shall be empowered to have private data and a corresponding certificate, short-term certificates are introduced. Technically they don't differ from standard long-term certificates and hence don't require further elaboration on the structure here. (cf. Section 7.4)

These certificates are created and signed by the holder of a long-term certificate. The creator of the new asymmetric key pair ensures, that the new certificate can not be used to create further certificates, i.e. in the X.509v3 `BasicConstraints` certificate extension, the value of `cA` is set to `TRUE`, and the attribute of `pathLenConstraint` to `0`. That extension must be marked as critical. [ITU97b, HFPS99] The validity period of the certificate should be set as short as possible, but long enough so that the mobile agent can fulfill its expected task with it.

Each corresponding short-term private key should be protected by being wrapped in a `LocalData` object as shown in Figure 8.7, with an attached meta-attribute `expires` as depicted in Figure 8.8. The timer component of the agent will trigger the removal of the object out of the agent's dynamic knowledge base.

Within the validity interval of the short term certificate it is vulnerable to espionage attacks. If an attacker is able to extract the certificate and corresponding private data out of the agent, within the time window given in the certificate it is vulnerable to abuse. Therefore, short-term certificates and their corresponding private keys should be used in very rare and well-controlled circumstances.

8.3.6 Sensitive Information Removal

It is mandatory, that no private data used to generate any secured content, meaning keys for either signed or encrypted data, leaves the home platform. This is especially important for the user's long term secret keys used for electronic signatures, they are required by law to never leave his machine or are being disclosed in any way. (cf. Section 2.6) Except by using dedicated hardware like chip cards this can be achieved within the agent platform in two ways.

The first is to inject the protected data into the mobile agent upon its inception by the creating entity. This would relieve the mobile agent from handling any sensitive data by its own. The security of the data then depends solely on the protecting home platform and the communication partners on the way.

The second possibility is to flag the knowledge base information within an agent to be removed by itself before it migrates. The ontology object `LocalData` as shown in Figure 8.7 may be used to wrap any such data. Before migration, the `KBP` agent component will remove all objects out of the knowledge base of the agent referenced by this data type. On the Java language level this can be achieved by tagging a variable with the modifier `transient` which prevents the contained data from being serialized.

Immediately before migration, the `KBP` should remove itself out of the agent as well, because it isn't needed on remote agent places due to its inherent unreliability there. In Section 7.2 and Section 9.2.3 it is specified, that an agent should only be started by a benevolent host if its code integrity has been verified. Unfortunately, the mobile agent itself has no reliable way to determine, if its code has been tampered with, because it might have been started by an evil host after modification. As a side note, modifiying the code of an agent before executions doesn't prevent a malicious host from sending the agent to the next host system unmodified.

For the second alternative, special care must be taken within the implementation not to accidentally leave any data around, for example by not removing any references to the objects before serialization. This is an aspect of security by robustness and safety as introduced in Chapter 4 and Section 3.10.5. In this sense, the former way of injecting sensitive data into the mobile agent without it having to process it itself is to be preferred, if applicable.

8.3.7 Limitations

A disadvantage of encrypting input for remote systems is, that the set of remote communication partners must be known before the mobile agent starts its journey. Because the list of visited agent places is normally not known beforehand but instead being calculated on its journey, this secured way of information dissemination is in certain circumstances not practical.

Alternatives are either to forfeit encryption and send the information in plain, or to distribute the symmetric key needed to decrypt the information on a different channel directly from the home platform to the receiving platform. The former way leaves sensitive information open but at least provides authenticated information to the remote system. The latter alternative has the disadvantage that the home platform must be reachable by the remote system at information processing time. As mobile agents are often used in a scenario for mobility support and offline processing to reduce bandwith usage this might not be feasible.

Another problem of returned data is, that it is hard to determine if it is the complete set of data received by the agent on its path. Nothing hinders a malicious host from removing selected information out of the knowledge base of a mobile agent. This can only be detected, if there is a reproducable chain of migration and communication of the agent over its life time.

Therefore, at each remote agent place, the host system would have to link its output to all previous outputs at other marketplaces and sign that output and the link with its private key, verifyable by a certificate. The receiving entity can then test, if the chain of linked result sets is unbroken. This is only possible though, if there is only one complete result set instead of a multitude of answers. Further, the linking of the results to previous ones must be performed by the host, not the agent. This is similar to the `AppendOnlyContainer` mechanism of Ajanta. [Kar]

8.4 Transport Layer Communication Security

To provide communication security on the transport layer of the ISO/OSI reference model (layer 4), an SSL component will be added to the agent. Because the more obvious name “Transport Layer Security” is already in use

by the security protocol with the same name, the new component is instead called "Low Level Security."

Cipher Suite Name	Key Exchange	Encryption	Hash
SSL_NULL_WITH_NULL_NULL	NULL	NULL	NULL
SSL_RSA_WITH_NULL_MD5	RSA	NULL	MD5
SSL_RSA_WITH_NULL_SHA	RSA	NULL	SHA
SSL_RSA_EXPORT_WITH_RC4_40_MD5	RSA_EXPORT	RC4_40	MD5
SSL_RSA_WITH_RC4_128_MD5	RSA	RC4_128	MD5
SSL_RSA_WITH_RC4_128_SHA	RSA	RC4_128	SHA
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5	RSA_EXPORT	RC2_CBC_40	MD5
SSL_RSA_WITH_IDEA_CBC_SHA	RSA	IDEA_CBC	SHA
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA	RSA_EXPORT	DES40_CBC	SHA
SSL_RSA_WITH_DES_CBC_SHA	RSA	DES_CBC	SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES_EDE_CBC	SHA
SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA	DH_DSS_EXPORT	DES40_CBC	SHA
SSL_DH_DSS_WITH_DES_CBC_SHA	DH_DSS	DES_CBC	SHA
SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA	DH_DSS	3DES_EDE_CBC	SHA
SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA	DH_RSA_EXPORT	DES40_CBC	SHA
SSL_DH_RSA_WITH_DES_CBC_SHA	DH_RSA	DES_CBC	SHA
SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA	DH_RSA	3DES_EDE_CBC	SHA
SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA	DHE_DSS_EXPORT	DES40_CBC	SHA

Table 8.1: Cipher Suites of IAIK

Table 8.1 lists the cipher suites supported by the IAIK SSL library used. [Ins99b] A cipher suite is a combination of algorithms for different properties plus the mode the algorithm is used in. The first column of Table 8.1 lists the name given to the following combination, it is canonically derived from the following parameters, this name is to be used in later configuration. The column "key exchange algorithm" is a symbolic name for the public key algorithm used for certificate-based authenticated key exchange. To support all cipher suites, an agent has to have at least three differnet certificate types, namely for the RSA, the Diffie-Hellman, and DSA algorithm. (cf. Chapter 3) Exemplary object instance are given in Section A.2.

The next column lists the symmetric encryption algorithm to be used within the communication, (cf. Section 3.5) the key length, (cf. Section 3.4.2) and the mode the algorithm is to be used in, if the specification of the algorithm supports different modes. (cf. Section 3.5.3) The last column is the symbolic name of the algorithm to be used as a hash for providing integrity. (cf. Section 3.4.3)

Property	Meaning
`jiac40.ssl.requireClientCertificate`	Allowed values are `true` and `false`. If it is set to `true`, the client has to send at least one certficiate chain to the server.
`jiac40.ssl.requireServerCertificate`	Allowed values are `true` and `false`. If it is set to `true`, the server has to send at least one certficiate chain to the client.
`jiac40.ssl.requireTrustedRoot`	Allowed values are `true` and `false`. If it is set to `true`, the root certificate of the certificate chain has to be known as a root CA to the agent or must be in the list of trusted signers.
`jiac40.ssl.server.port`	The port, the component listens on for incoming connections.
`jiac40.pregenerateRSA`	Allowed values are `true` and `false`. If it is set to `true`, the component pregenerates an ephemeral RSA key upon startup. This slows agent startup but speeds up the first connection.
`jiac40.tempRSAKeyLength`	The key length of the temporary RSA key.
`jiac40.ssl.pregenerateDH`	The same as for RSA, but with the Diffie-Hellman algorithm.

Table 8.2: Mandatory Properties of the SSL component

Property	Meaning
`jiac40.ssl.session-management.client`	Allowed values are `true` and `false`. If it is set to `true`, client SSL sessions will be added to the session management part of the library.
`jiac40.ssl.session-management.server`	Allowed values are `true` and `false`. If it is set to `true`, server SSL sessions will be added to the session management part of the library.
`jiac40.ssl.socket.timeout`	Defines the number of seconds to wait for closing a socket after the last data transfer. The default is 300 (five minutes).
`jiac40.ssl.dh.param.basegenerator`	Externally determined base for Diffie-Hellman.
`jiac40.ssl.dh.param.primemodulus`	Externally determined modulus for Diffie-Hellman.
`jiac40.ssl.protocol.name`	Name of the SSL protocol as the protocol part of an agent address. Default is `ssl`.

Table 8.3: Optional Properties of the SSL component

The configuration of the SSL component is done by a Java property file loaded into the agent upon its startup. For reasonable use of the SSL component, the agent must have at least one certificate as well. The comprehensive list of properties to use for configuration is given in Table 8.2 and Table 8.3.

According to [Fou97], agent addresses are formed similar to URLs: [BLFM98]

```
<protocol>://<protocol-address>:<port>/<target>
```

To use the SSL component in agent communication it is sufficient to use the string `ssl` as the communication protocol. The configuration of both communication partners then determine which exact parameters will be used in the SSL session, transparently to the agents. (cf. Section 8.2.3)

How the agents know about the transport addresses of their peers is covered by the agent management system and is not part of this work. [Fou97]

8.5 Application Layer Communication Security

The use of SSL as a transport component for the agent architecture, as elegant and simple as it is, has some disadvantages as well. The first problem with that approach is that for use of a different transport layer, the agent needs to open another TCP socket. That increases resource consumption and might lead to bottle-necks in low-capability devices like small PCs or hand-held machines.

Using SSL isn't optimal either, if one wants to send only sporadic speech acts not being part of a larger session, like `inform` acts are designed for. SSL has to establish a session involving several communications in both directions. After using the session for a single speech act, it has to be closed down again. This is undesirable communication overhead.

Due to its direct mapping as a peer-to-peer communication protocol, SSL isn't capable of supporting multi-point communication. Instead, each communication partner has to establish a secure session to each partner it wants to communicate with. The order of complexity of communication paths is exponential.

The strength and purpose of SSL, protecting the contents of raw data, is probably the most important disadvantage in the agent world as well: between agents it is very common, that speech acts are brokered by respective entities, e.g. the ACC in the FIPA reference model [Fou97] or in the CASA model. [Ses02] Protecting the routing information against eavesdropping disables this feature. The simple approach of using SSL from the agent to the ACC, where the speech act is dismantled and re-packaged again for another SSL connection to the next hop isn't feasible, because that would disclose the entire contents of the speech act to each routing entity.

Therefore, a new communication component is provided, that enables authentication, confidentiality, and integrity on an end-to-end channel between the involved peers, but still allows for message routing by protecting only the contents of a speech act, and leaving its relay information open. The component is called "Speech Act Security" and definitions of objects are shown in Section A.2.

If in the following the terms "server" and "client" are used, this is not to be confused with information provisioning and consumption. Instead, it purely defines, which side of the communication initiates the secured communication (the client), and which side will respond to that request (the server). The naming stems from [FKK96] and is continued here for easier reference and understanding.

8.5.1 Speech Act Realization

The embedding of security into the speech act layer makes use of the `letter` wrapper object of [Fou97]. Similar to a mail envelope, by using a `letter` object a speech act is divided into an envelope, called exactly so, and a message body tagged `:message`, containing the communicative act. The envelope contains all information necessary for in-transit processing. The letter object itself may be wrapped further, however the ACC deems necessary for transit.

8.5.2 Connection-less SAS

This component offers two distinct message transport mechanisms. The first mechanism of the SAS component sends speech acts without employing sessions.

Content Type	Authentication	Encryption
`signedData`	yes	no
`envelopedData`	no	yes
`signedAndEnvelopedData`	yes	yes

Table 8.4: Protection by PKCS#7 Content Type

For sending protected speech acts out of a session, PKCS#7 objects are employed. [RSA93e] Table 8.4 shows which data type offers which kind of protection, reduced to the content types reasonable for SAS protection; authentication includes integrity and replay protection. Other data types of [RSA93e] are of no use for communication purposes or are only utility types used within other types.

The `:content` of the `:message` part of the `letter` object to be sent consists of an object of the category `ContentInfo` as depicted in Figure 8.6 in accordance with Figure 8.5. The opaque octet string `encoding` is the BER encoding [CCI88c] of an ASN.1 [CCI88b] object of the type `ContentInfo` as defined in [RSA93e]. (cf. Figure 8.5) depending on the nature of the content, the inner encodings of the PKCS#7 object might be restricted to being either DER encoding [ITU97b] or definite-length BER encoding. [RSA93e]

The `:language` message parameter is set to `cal`, the `:ontology` parameter is `ProtectedContent` denoting its interpretation by the component with the identifier `SAS` and is shown in Section A.2.

Because there is no established session with determined and associated capabilities of the communication path, the sender must take care only to use algorithms that can be understood by the recipient. The exact way to determine the capabilities of remote agents out of communication sessions resp. service usage is not part of this work, but the objects as defined in Section A.3 can be used in a dedicated protocol for feature detection.

To use the connection-less SAS for sending speech acts, the `:message` content must be generated by the application component sending the message. It uses the APIs as provided by the IAIK JCE library, [Ins99a] based on the knowledge the application component has about the receiver of the speech act. This ensures a maximum of flexibility.

Upon receipt of a speech act of the ontology `ProtectedContent`, the agent-internal message dispatcher will forward the speech act to the decoding part of the SAS component. The component then creates an internal representation of the speech act in the `:message` part of the envelope, decoding and authenticating it. If a content type was used that should ensure authentication, the authenticity of the message is tested. If it is authentic, a flag is added to the speech act representation tagging it as successfully authenticated.

If during the process of receiving a speech act of the ontology `ProtectedContent` a processing error occurs, e.g. authenticity could not be guaranteed or the encryption was unknown, a `not-understood` response is generated, if possible, depending on the attributes in the message `:envelope`. The speech act is then dropped and not internally rescheduled to the dispatching component.

8.5.3 Connection-oriented SAS

The second way of protecting speech acts is modelled according to the SSL layer and provides a connection-oriented transport capability. Not only is this more secure than connection-less speech act security, (cf. Section 8.5.4) but because of the availability of the service concept of CASA, it eases use for the application programmer as well.

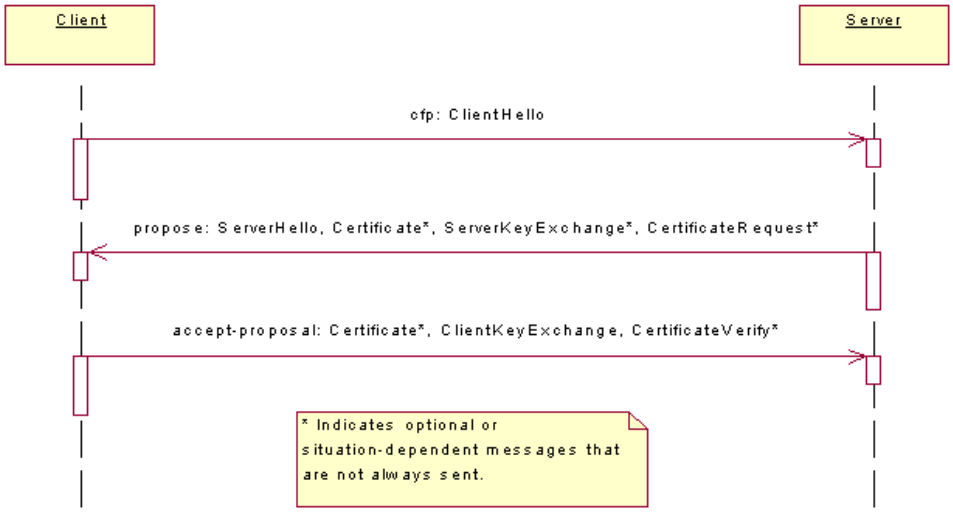

Figure 8.9: SAS Handshake Protocol

The protocol used resembles the SSL handshake protocol very closely, with

some simplifications, it is depicted in Figure 8.9. There is no explicit notification of cipher change as specified in [FKK96], (cf. Figure 3.12) instead that is performed implicitly with the `accept-proposal` message.

Re-keying within an ongoing communication is not supported in the proposed model. It is assumed, that service usage is rather short-lived and there is no need to re-key while actively using the service. If a long-term service is to be supported, e.g. an information push-service, this can be realized by using independent service accesses.

```
(ont SSLforSAS:V_1
     (cat ClientHello
          // random value used for shared secret generation
          (random         byte[]          needed)
          // the cipher suites supported
          // for this service use
          (requirements   AgentAbility[]  needed) )
     (cat ServerHello
          // random value used for shared secret generation
          (random         byte[]          needed)
          // the chosen AgentAbility
          (chosen         AgentAbility    needed)

          // certificate chain of the server
          (certs          Certificate[])
          // server key exchange data
          (keyexchange    ServerKeyExchangeData) )
     (cat ClientFinished
          // certificate chain of the client
          (certs          Certificate[])
          // client key exchange data
          (keyexchange    ClientKeyExchangeData needed) )
     (cat ProtectedData
          (protected      byte[]) ) )
```

Figure 8.10: SSL for Speech Act Security Ontology

For all following speech acts, the parameter `:language` is set to `cal`, whereas the parameter `:ontology` is set to `SSLforSAS`. In some communicated category objects of the handshake protocol (cf. Figure 8.10) an array of certificates is used. According to the SSL standard, this should be a chain of certificates, where each certificate's private key was used to sign the former, ultimately leading to a root certificate. Due to the proposed certificate infrastructure

agents will be able to test the validity of certificates even if the complete chain is not given. (cf. Chapter 10)

The communicative act of the first speech act is a call for proposal, defined as `cfp` in [Fou97], with an object of the category `ClientHello` as its `:content` attribute. The object's attribute `requirements` lists the cipher suites as understood by the client. (cf. Section A.3.4) If a certificate selection is part of the cipher suite specification, the client must be able to authenticate the server by that algorithm.

One of these cipher suites sent by the client must be `chosen` by the server for the service use. The certificate used by the server for authentication and sent in the attribute `certs` must correspond to the chosen cipher suite and can be missing, if the server may remain anonymous. The type `ServerKeyEchangeData` is to be defined similar to [FKK96] and defines the parameters for the selected key exchange algorithm as named in `chosen`. It may be missing, if not appropriate due to the chosen cipher suite.

The communicative act of the response is a `propose` [Fou97], containing a `ServerHello` object as the `:message`. If the server can't or won't accept a communication, a `refuse` is sent back.

In contrast to the SSL handshake protocol, the server doesn't have to send back a certificate request to the client. Instead, the client has to respond with a certificate of the same type as sent by the server in the `ServerHello` message object. This eases implementation, as dealing with different kinds of certificates for the same connection becomes obsolete. By reasoning on the application domain level it is assumed, that if information being so important, that the server has to authenticate for it, then that service will be offered only to registered consumers. So in this model and with connection-oriented speech act security, if agents want authenticated data, they have to authenticate themselves. This enhances trust between both communication parties, because it facilitates a *quit pro quo* attitude.

The initiator of the communication, if the proposal of the server is acceptable, answers with an `accept-proposal` communicative act,[Fou97] containing a `ClientFinished` object as the `:message`. The type `ClientKeyExchangeData` is to be defined according to [FKK96]. A certificate chain might be missing, if mutual authentication was not negotiated. If the initiator choses not to accept the proposal, a `reject` communicative act should be sent back. If internal computations depending on the server-provided information failed, a `failure` should be sent back.

After the three-way exchange of messages both parties share a common se-

cret. The exact way of deriving that pre-master secret depends on the key exchange algorithm used. From that, a key block is generated, that gives input for computing the message authentication code, the symmetric key, and the initialization vector. (cf. Chapter 3) The exact way of generating these material can be found in [FKK96].

After successful completion of the handshake protocol, the service object instance as seen in Figure 8.4 of the ongoing conversation is updated according to the negotiated *modus vivendi*. If authentication was demanded, the certificate object that was used for authentication is added to the `ServiceInst` object that describes an instantiated service use. (cf. Figure 8.4) Each further message of that conversation will be checked against the service needs and the remembered certificate. A missing certificate object in an `ServiceInst` object denotes anonymous use. Only after an incoming speech act is accepted as correct against the specification, it will be processed within an agent. Each speech act sent in the conversation will be transparently processed by the `SAS` agent component before sending it onto the network.

All further messages will be wrapped as `letter` objects. For compatibility reasons to [Fou97], the `:message` part of a `letter` has to be an ACL message. Unfortunately, the FIPA standard doesn't provide an adequate communicative act as needed here. The `evaluate` from KQML would be was is needed. [LF97] Instead, an `inform` will be used for all speech acts.

The `:conversation-id` has to be set in the message to reference the service object defining the negotiated parameters. The `:content` of the message of the letter is an opaque octet string, represented by an object of the type `ProtectedData`. After decryption, the result will be the protected speech act or another letter object.

In the context of the CASA architecture the session establishment has to be done as the first step of the service meta-protocol, before other parameters are negotiated, to already protect that other negotiation.

To enable multi-point communication, an agent has to perform different handshake protocols with each peer. The content to be protected will then processed according to the service parameters negotiated with each partner.

The configuration of the SAS component is done by its own initialization function. It has to load the appropriate facts into the knowledge base of the using agent. (cf. Section 8.2.2)

8.5.4 Comparison of SAS to SSL

The security of the session-oriented speech act security is the same as the SSL protocol, because the former resembles the latter.

Using the connection-less way of communication is prone to several different attacks, though. If the needed certificates are not known beforehand, the data is prone to man-in-the-middle attacks, because it can't be ensured, that the public key of the intended recipient is received. This can be overcome with the use of a certficiate infrastructure as is provided by this work. (cf. Section 10.2)

Another danger of non-session communication is message replay. To detect this, the sender must try to ensure liveness of the communication by sending fresh nonces in each communication. Without a handshake protocol though, it is hard for the recipient to decide about the validity of the nonce. Replay can only be detected, if the recipient stores older nonces. This leads to several different problems: The size of the database at the receiving side grows with each message, so there must be a heuristic to purge it regularly. On the other hand, the more nonces are stored, the more likely is a collision of nonces of different communications. In every case, an attacker can exploit the database heuristic to send messages with nonces not in the database.

Message insertion is another viable attack form. It becomes more likely the more predictable the variable contents of the speech act is. E.g., if nonces are chosen consecutively, the attacker can predict valid numbers and even hijack a session.

Leaving the sender and receiver part of the speech act open to be read by any party allows for traffic analysis attacks. These are neither part of this work, nor a serious concern today anyway, so they will not be elaborated upon further.

In summary, if possible and viable, transport layer security via SSL should be chosen in peer-to-peer communication. It is most performant in comparison and the less vulnerable to attack. If message routing and on-way inspection is needed, speech act layer security with a session should be used. Only if optimization requirements enforce serious restrictions on communication overhead and delay connection-less security should be chosen. Even then, application-specific data should ensure further liveness.

8.6 Service Authorization

As a significant security enhancement over other architectures, the proposed infrastructure has means to ensure authorization control over the access to provided services.

```
(ont ServiceControlMechanisms:V_1
     (cat ServiceControlList
          // if missing or empty -> deny
          (entries        ServiceControlEntry[]) )
     (cat ServiceControlEntry
          // all have to match
          (checkAttributes        CheckAttributes[])
          (logging                bool)
          (result                 string needed
                  (constr (comp isElement
                  ?result { "accept" "deny" "reject" })) ) )
     (cat CheckAttributes
                (attribute      string  needed)
                // if missing, fulfilled by default
                (pattern        string) ) )
```

Figure 8.11: Service Control Ontology

Figure 8.11 shows the ontology for the attributes attached to the service description, simplified by stripping of management data.

If a service is to be protected from unauthorized access, a service control list (SCL) is to be attached. A missing SCL grants access to all entities. For an existing non-empty SCL, a successful authentication of the service user must have been performed in a previous step.

If there is an SCL attached to the service, all service control entries (SCE), also called rules, of the SCL of that service are sequentially checked. A check of an SCE means to check all contained `checkAttributes`. For each of these, the attribute of the certificate as described by `attribute` in its string representation is matched against the regular expression as contained in `pattern`. A certificate attribute is any of the parts a certificate consists of, including extensions. If the patterns of all attributes to be checked match the rule is activated, i.e. the pattern are combined by an `AND` operation. If at least one pattern doesn't match, the next SCE is checked, i.e. the SCEs

within an SCL are `OR`-combined. If no SCE matches, the default action of the SCL is to function as a `deny` rule. This is particularly true for the special case of an SCL without any SCE.

If a rule is activated, the `result` string describes the action to be performed. To `accept` the incoming speech act means to grant service access. It is to be flagged at the service instance object describing the conversation, that all subsequent speech acts in that conversation id have to be authenticated by the certificate presented to the service access check. This is done by adding the certificate object granted access to the `ServiceInst` object as seen in Figure 8.4. If the certificate attribute object in the `ServiceInst` object is missing for an ongoing conversation, anonymous access to the service is granted.

If the `logging` boolean flag is set to true, the firing of the rule is appropriately logged by the respective facilities of the architecture. (cf. Section 9.2.3)

If the result of the use check is a `reject`, a results `failure` speech act is sent back to the requestor, informing it that access restrictions applied. The `deny` result forfeits an answer in the attempt not to give away any information to a possible attacker and even trying to hide the very existence of the requested service. In the CASA system, a `deny` result isn't feasible due to the meta-service protocol which doesn't provide for a suppressed answer to a service request.

The initialization of the service entries is done by the specific initialization function of the service to be protected upon loading the service into the agent. It is the obligation of an administrative service management team to appropriately define the service control policy for a service.

If an agent's authorization settings are to be modified remotely, the agent can offer the following services:

GetSCL: The agent receives a service identification and returns the SCL that is attached to the service.

SetSCL: The agent receives a service identification and an SCL and attaches the SCL to the service.

RemoveSCL: The agent receives a service identification and removes the associated SCL, effectively opening up the service use to anyone.

The service control management services themselves are governed by the service control mechanisms, so that they might be offered only to authorized agents, e.g. the configuration tool of the agent's home platform.

Using SCLs allows for flexible access permissions with positive and negative rules. As an enhancement over pure Java based access control mechanisms it enables dynamic policy definition during runtime. Further, due to the pattern matching capability, not each identity has to be entered into the policy database before communication takes places, thus making dynamic, spontaneous, and instanteous service usage possible.

8.7 Summary

This Chapter has provided an agent with the ability to reason about its own security functionality and to adapt according to the needs. SSL was made available on the semantical level, and its restrictions in elaborated situations pertaining to agents have been overcome by introducing another component related to communication, this one on a higher abstraction layer.

The service protection mechanisms as shown in this Section are an elegant solution for the whole of CASA architecture: all communication-based functionalities of CASA, including management functions like starting or migrating agents, are services in the CASA sense. That enables them to be implicitly protected by the means as described in this thesis. The service protection facilities apply unchanged, providing for consistent security features over the whole platform, without making the architecture over-complex with redundant functionality.

Chapter 9
Agent Community Security

"Speak friend and enter."
Inscription on the Doors of Moria

9.1 Synopsis

The security mechanisms to be developed in this thesis are extended onto the agent place level in this Chapter. Several infrastructure features are introduced to provide more security on the level of an agent community.

The Manager Agent (MA) as a system agent managing a platform is extended in several ways. It becomes a valuable role in preventing malicous code to get onto the platform and running. In the event of an incident, the MA will have mechanisms to deal with the situation.

A new system is introduced, the Security Agent (SA). That agent is responsible for all tasks pertaining to security on an agent place, as far as it exceeds basic platform management functionality as fulfilled by the MA. This especially includes distribution of certificates and their revocation lists, as well as gathering and providing information about remote agent places.

9.2 Manager Agent

In the CASA framework, there is one dedicated agent per agent place, the *Manager Agent* (MA for short). That agent constitutes the agent place and is the first agent thereon. It creates all further initial agents upon inception of the agent place. The Manager Agent handles agent migration from and to the place.

Because of the inherent autonomy of agents, the MA is not responsible for the agent's communication, which is handled by the agents themselves. (cf. Chapter 8) The protection of agents against each other is handled by the underlying platform, in the case of CASA that is the Java runtime environment. (cf. Chapter 7)

Instead, the manager has responsibility for protecting its managed agent place, especially against mobile agent intruders, and it plays a role in back-tracing incidents after subversion of mobile agents has been detected.

9.2.1 Platform Management

Upon creation of a Java runtime environment, after initializing the class loading and security management internal functionality, the first agent instantiated is the CASA Manager Agent.

The MA then loads further agents as specified in the CASA framework. This agent loading is subject to the mechanisms as layed out in Chapter 7.

After initial loading of agents and starting the agent place, further agents are created by the respective management service. [Ses02] As well, this agent creation process is governed by the basic security features as described in Chapter 7.

The agents' communication on the local agent place is subject to the same rules as if communicating with remote agents. The only difference is the use of a VM-internal message passing mechanism instead of full-fledged TCP/IP communication, thus speeding the communication up. There is no difference in the security mechanisms though, and as provided by the mechanisms of Chapter 7 the agents are not enabled to access each other in any other way except their well-defined service interfaces, irrespective of their virtual proximity.

9.2.2 Migration Restrictions

The Security Agent (abbreviated SA) of an agent place holds information to determine, if incoming or outgoing migration to or from remote places is allowed. (cf. Section 9.3.2)

Within the migration protocol of the CASA platform as enhanced by this work, there is a distinct phase, where the Manager Agents of both sides consult their respective Security Agent and ask it for permission to migrate. (cf. Figure 9.1)

It is important to recognize, that the migration of an agent by its Manager Agent is a CASA service usage. That enables the use of communication security mechanisms as shown in Chapter 8. In the further proceeding of this Section it is implied, that those mechanisms are employed and migration of agents is secured against eavesdropping and tampering with.

For the source platform, in the case of a disallowing entry in the access list, it provides some confidentiality, that a mobile agent will not be sent to a malicious remote platform. In the case of an allowing entry for the remote platform, the sending Manager Agent has some notable trust, that the remote agent place is not hostile to the mobile agent.

For the receiving Manager Agent, if it is allowed to receive the mobile agent, the MA will have some confidence, that the agent comes from a benevolent platform. In the case of a denying entry, the agent place is protected against a mobile agent from a hostile remote platform.

If in either case there is no entry regarding the remote platform, or the entry is `Unknown`, there is no confidence for the Manager Agent whatsoever with respect to the benevolence of the other platform.

In the pro-active case, a mobile agent might consult the SA itself before running into the fallback case of an unsuccessful migration by the MA.

If the Manager Agent does not employ the capabilities of the Security Agent the CASA architecture provides the configuration property `migration` which might disallow agents migrating to the local managed place. Without the SA, there is no way for the platform to disallow outgoing migrations.

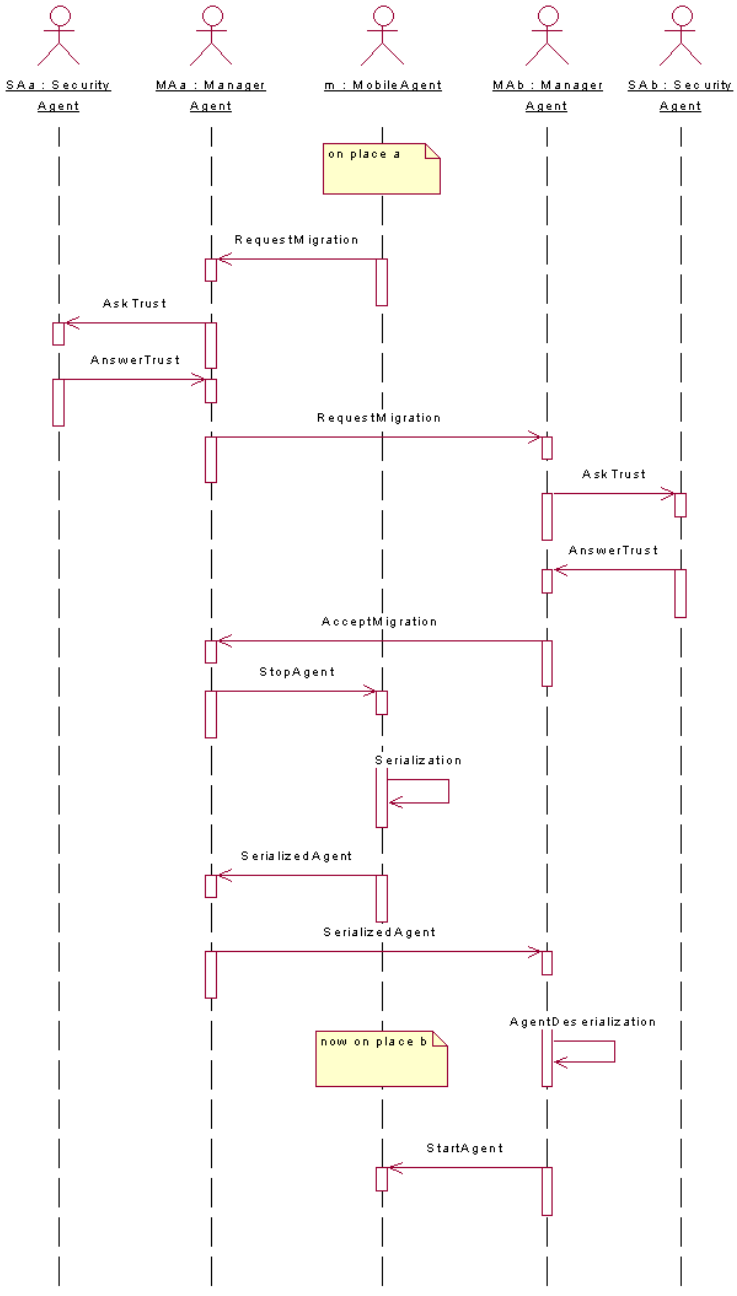

Figure 9.1: Enhanced Migration Protocol

9.2.3 Backtracing Mobile Agent Incidents

Section 7.2 shows the mechanisms in place to secure the agent platform against intruders, specifically mobile agents from various sources. As has been discussed in Section 8.3, mobile agents may have been tampered with on their journey. If not to protect these tampering, but at least make them detectable and backtracable, benevolent host platforms shall have means to support this.

There are several distinct parts of the mobile agent, that can be identified. Pertaining to the code, it is the class definitions and the class instances that are transported. With respect to the data part, the knowledge base is of special importance, and it is divided into a static and a dynamic part. (cf. Section 8.3)

Due to the strong migration concept of CASA, where the state of the agent is transported along with its code, the class instances of the components the agent consists of might change between invocations on different platforms and thus the code classes must be treated as dynamic data. Therefore, of the agent's code only the class definitions are immutable. They can be signed by the home platform and must be checked against integrity by each receiving agent place's manager before starting it. The receiving Manager Agent should log the successful or non-successful receiving of the class definitions of a mobile agent. For limiting the misuse potential, short-term certificates should be used for signing. (cf. Section 8.3.5)

The code class instances might change between runs of the agent. Therefore, the correctness of the code can not be deduced by a Manager Agent. Instead, the Manager Agent has to calculate and log a cryptographically strong checksum over the code part upon receipt and again before sending the mobile agent to its next execution place. (cf. Section 3.4.3) That information might help if an incident of misuse has been detected and it is tried to determine, where the agent were still intact and where it has been modified. The same actions are applied to the dynamic knowledge base of the agent.

Similar to the immutable code part of the agent, the static knowledge base can be signed by the mobile agent's owner upon creation, and is then checked for integrity upon receiving the agent at each place. The respective Manager Agent logs the result of the check. The same treatment is applied to the static knowledge base data of the mobile agent.

Alongside with each of the above information, the remote agent place, either the sender or the receiver, depending on the use case, must be logged too, so

that the itinerary of the mobile agent is reconstructable. For later reference to the data, the global unique identifier of the agent as defined in the CASA framework is to be put into the log as well.

If a receiving Manager Agent detects an agent as being tampered with, the MA neither reconstructs nor starts it. The sending Manager Agent is informed and the migration is aborted. Further, the owner of the mobile agent as can be deduced from the signed static knowledge base content is informed about the corrupted agent.

The sending Manager Agent will save the serialized agent for later analysis in a dedicated area as well and not re-start it again. Both sides log the incident. The saved agents' data should expire after a reasonable amount of time so that the storage area doesn't clutter and denial of service attacks by flooding the area with intentionally broken agents is less likely.

For these functionalities it is advised, that a secure logging facility will be employed. Exactly how that is achieved is out of the scope of this document, but there are several options to examine. For example, in the context of a Unix system, the standard `syslogd` can be used, potentially over a secured network on a remote machine for enhanced tamper resistance. On a Windows NT server machine, the logging facility of the operating system could be used. In an SNMP environment, those event mechanisms can be employed. As another option, the management framework of the CASA architecture provides for logging functionality.

It is noteworthy, that a malicious host can still exchange the signed data with its own and sign that, so that the modification isn't noticed at first, but the change gets accountable, because the host has to sign with its one of its own valid and distributed certificate. That effectively changes ownership of the data.

The performed tests and their results can be used for a dynamic trust establishment scheme. Though the basics thereof are laid out in this work, a concrete scheme will not be given. (cf. Section 9.3.2)

9.3 Security Agent

To assist the Manager Agent of an agent place in the management of security related relationships to other places, the *Security Agent* (abbreviated as SA) is introduced as a standard architecture agent to the CASA framework.

The employment of another dedicated agent for management of the trust lists has several benefits: the platform becomes more resistent against attacks, because even if one of the agents, either the Manager Agent or the Security Agent, have been tampered with the other is still operational. The platform can be secured even better, if the Security Agent is placed on another agent platform and potentially on another executing host that is more secure than the other places.

Another benefit lies in the applicability of the agent feature "autonomy." Partition of the functionality leaves for the Manager Agent the possibility of disregarding the hints of the Security Agent and decide about migration attempts irrepective of the Security Agent. Further, in this way not only the Manager Agent has access to migration-related security information, but other agents might employ the services of the Security Agent as well, thus allowing them to work more autonomous than without.

Not the least, introducing a dedicated security agent provides for a further means to scale the service provisioning domain: one SA can offer its services to several agent places, or the load can be shared by multiple SAs for only one domain.

The Security Agent has two main tasks: certificate distribution and trust management, each realized in a dedicated agent application component.

9.3.1 Certificate Distribution

For a complete certificate infrastructure a key distribution center (KDC) is needed. [ITU97b] The KDC accepts keys from a certificate authority (CA) and makes them available to requesting entities, in this case, all agents on places in the domain that the SA serves. Certificates are thoroughly discussed in Section 3.7.5 and are defined for use in the CASA framework in Section 7.4.

The SA provides the following certificate-related services to the domain:

AddCertificates: The SA receives a list of certificates. After validation

of each certificate and check against any CRLs they are added to its knowledge base and can furtheron be retrieved by other agents. If only a single certificate is to be communicated the list contains only the one certificate.

RemoveCertificates: The SA receives a list of certificates. Each certificate is removed from its knowledge base, if it exists there. If only a single certificate is to be communicated the list contains only the one certificate.

GetCertificates: The SA receives a list of certificate or subject identifiers. For each identifier, the SA sends back all known valid certificates.

NewCRL: The SA receives a new certificate revocation list. That list supersedes an already known CRL from the same CA, if it is newer than the existing list. After the CRL has been successfully validated, the SA checks its knowledge base of certificates and removes those, that have become invalid due to the specifications of the new list.

GetCRLs: The SA receives a list of subject identifiers, that are interpreted as certificate authorities. For each of these identifiers, the SA returns the CRL of that CA, if known.

Supplying these services to its domain, the agents in that domain are now able to perform actions upon certificates they don't know beforehand. The services potentially leading to a modification of the SA's knowledge base should be restricted to only very few and trusted entities.

Due to the potentially high bandwidth need and processing overhead incurred, and not to compromise the security of the SA, the SA does *not* receive or distribute agent-related or short term certificates. (cf. Section 7.4) Instead, each agent itself is responsible for distributing its applicable short term certificates along with any authenticated data, e.g. the SSL protocol handles this.

9.3.2 Agentplace Trust Lists

The Security Agent manages trust relationships to other agent places and provides this information to the agents of its domain, especially the Manager Agent.

```
(ont ATL:V_1 (incl SecurityRequirements:V_1)
     (cat AgentplaceTrustList
          // the ATL managing agent
          (owner agentname  needed)
          // the entries
          (mtes  ATLEntry[] needed) )
     (cat ATLEntry
          // target for the relationship
          (forAgentplace agentname needed)
          // kind of relationship
          (values    TrustEntry[]  needed) )
     // possible relationships
     (cat TrustEntry
          (value string (constr (comp element ?value {
                 "Unknown"
                 "MigrateTo"
                 "AcceptMigration"
                 "SignCommunication"
                 "EncryptCommunication"
                 "DontCommunicate"
          }))
                 (default "Unknown") needed)
          // optionally needed parameters for
          // secured communication
          (param AgentAbility[]) ) )
```

Figure 9.2: Agentplace Trust Lists Ontology

Each Security Agent manages its own distinct relationship list. The list holds several entries of the type `ATLEntry`. Each entry defines the relationship to one remote agent place. The relationship might consist of several values. `Unknown` symbolizes no information about the remote place and is semantically equivalent to no entry at all. `MigrateTo` allows migration to the remote agent place, whereas `AcceptMigration` accepts incoming mobile agents from there. With `SignCommunication` the need for authenticated communication is described, `EncryptCommunication` specifies to encrypt all communication

to that place. `DontCommunicate` is mutually exclusive to all of the above entries and urges every agent that cares, not to communicate with any agent on the remote agent place at all. The identification of an agent place is achieved by the global unique identifier of its Manager Agent.

```
(matt confidence real (ktype fact)
      (constr (and (>= ? 0.0) (<= ? 1.0)))
      (default 1.0)
      (match (>= ? 0.5)) )
(matt source agentname
      (ktype fact object goal))
```

Figure 9.3: Meta-Attribute Declarations

The CASA ontology language allows for meta-attributes. That feature makes it possible to put some finer granularity on the managed relationship, most prominently to add a confidence value. A typical confidence declaration in CASA is shown in Figure 9.3. As an exemplary side-effect, it defines, that a match by the platform-internal algorithm is only achieved with the fact, if the confidence level is at least 50%. Further, the meta-ontology declarations provide for a `source` agent of a fact.

```
(obj OAP    TrustEntry
     (value "MigrateTo"):(meta
            (confidence 0.72)
            (source "Manager@iiop://192.168.117.35:5201") ) )
```

Figure 9.4: TrustEntry Object

A `TrustEntry` object can then be instantiated as shown in Figure 9.4, which reduces the confidence of the Security Agent in the relationship to the remote system to 72%. This enables the Security Agent and other agents, that are responsible for evaluating relationships to remote agent places, most importantly the Manager Agent of the agent place, to adapt the relationships according to the experience with remote systems.

For providing the gathered information and to manage the relationships, the Security Agent offers the following services to its domain:

GetTrust: The SA receives the identifier for a remote system and returns the `ATLEntry` thereof. This service should be restricted to agents of the supported domain.

SetTrust: The SA receives a `TrustEntry` object. It sets the `source` meta-attribute to the sender and inserts the object into its trust list, possibly overriding any previous value. This service must be restricted to only very trusted agents, like the Manager Agents of the supported domain.

RemoveTrust: The SA receives the identifier for a remote system and removes the corresponding `ATLEntry` out of its knowledge base.

ChangeTrust: The SA receives the identifier for a remote system and a floating-point value and changes the confidence value by the given amount. The `source` meta-attribute of the knowledge entry is updated accordingly.

For all communication and migration to or from an agent place, the respective agent, be it the Manager Agent or any other on the local system, is now enabled with knowledge gathered and evaluated by the local community about the remote system. The agent might now autonomously decide to honor the hints as given by the SA, or to defy it and take the risk it was warned against. (cf. Section 9.2.2)

9.4 Summary

This section extends the standard Manager Agent of an agent place by security-related mechanisms, allowing it to protect its managed place to fall victim to malicious code and behaviour. If security-sensitive incidents happen, the situation can be reconstructed and appropriate measures be taken.

Due to the introduction of the Security Agent, an agent place will hold information about its environment of other places and the MA is enabled to pro-actively prevent code exchange with malicous systems.

Figure 9.5 shows two agent places. The left has all agents playing a role in securing the place. The Security Service Agent on the left place and the Certificate Authority Agent on the right-hand agent place are introduced in the next Chapter as providing security on a global scale, across borders of agent places.

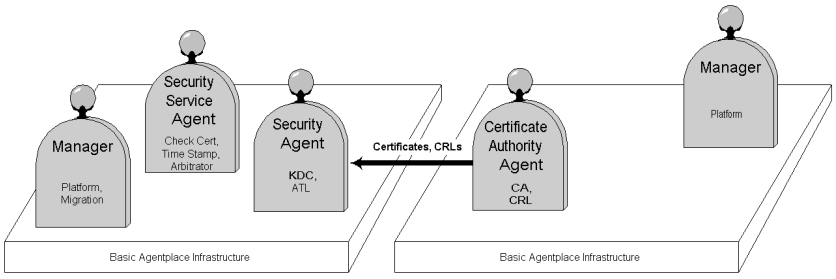

Figure 9.5: Community Security

Chapter 10
Global Security

"One who is not acquainted
with the designs of his neighbors
should not enter into alliances with them."
Sun Tzu, The Art of War

10.1 Synopsis

After providing basic mechanisms on the execution level, then raising the abstractness over the agent level to a community of agents, now mechanisms will be introduced to provide security on a domain or even global level.

The certificates as has been dealed with so far in Part III are hierarchic and thus have to be traced back via a chain to a root. This Chapter introduces the Certificate Authority Agent to provide that root, issuing certificates and certificate revocation lists to users.

A Security Service Agent offers value-added services that allow agents to give up own functionality for reasons of speed and size. Further, means to conduct automated e-business as close to legal requirements as possible are enabled. All those services might be offered by providers charging for the provisioning.

10.2 Certificate Authority Agent

For building a complete certificate infrastructure, all certificates have to be traced back to an ultimate root certificate. That is used by the Certificate Authority (CA) to derive all other certificates from. The distinct property of the root certificate is, that it is signed by the secret key belongig to itself. (cf. Section 3.7.5)

Currently, there are already established commercial certificate authorities available, e.g. [Ver97], [T-T01], or [Hei99]. There are several reasons though, why those CAs might not be sufficient or not applicable to the situation at hand: First, commercial CAs don't offer or support the certificate extensions as introduced and proposed in Section 7.4. Those extensions are very helpful for the security of agent communities.

Another reason not to use commercial certificate services is the non-availability of standardized online ways to retrieve or check certificates. All rely on proprietary "protocols," commonly that means manual access of plain web pages by humans, if any online access mechanisms exist at all; certificate revocation lists are even less supported. These mechanisms are clearly not appropriate for automated retrieval. There are standadization efforts for online protocols on the way, but a wider adoption is not to be expected anytime soon. [AF99, MAM+99]

Neither the latency time for issuing a new certificate nor the price for one certificate request should be neglected. Unfortunately, with increasing security of the issued certificate, the time until one receives the certificate and the price go up. If one nevertheless accepts the dependability on third party certificates, that introduces new points of failures in the business process. If the CA is not available for any reason, there won't be new certificates either, hampering sales numbers.

The most security-relevant consideration is, how trusted the third party is to issue reliable certificates in a well defined and secured process, to only entities with appropriately and thoroughly proven identities, and how stable do they stick to their policy, and how well-defined, concise, and applicable to the situation at hand is that policy?

On the very pragmatic side of software development it is very helpful neither to depend on outside services nor to play around with "real world" data, instead some "play ground" certificates come in handy to work with.

For all those reasons, an agent-based Certificate Authority Agent (CAA) will be introduced. That CAA supports standard mechanisms of certificate management, including certificate revocation list management, and uncommon extended attributes. It should be considered as applicable in several environments, and its actual use depends on the policy to be implemented. For agent-based communication, it is "nice" to have the CA online and available by services, though from the security point of view that is undesirable.

The certificate management functionalities of the CAA include issue, distribution, and revocation of certificates; certificate revocation lists (CRLs) are distributed as well.

For simplicity, the added complexity and abstraction entity of a separate Registration Authority (RA) as introduced in [AF99] is not used. It is assumed, that the tasks of the RA are handled by the CA itself, which is compliant to [AF99] that notes the RA as being optional. This does not constitute a restriction on the overall security infrastructure, because all issued certificates are fully interoperational with standardized X.509v3 certificates. [ITU97b] Merely, it is an assumption about policy and the use of own tools, instead of deployed ones.

The CAA is only concerned with long term certificates issued on the identities of users. The corresponding secret keys never leave the user's platform. The generation and handling of short term keys for the use in mobile agents was described in Section 7.4.

The implementation of the running prototype code is based on the IAIK libraries and references their classes at some points. [Ins99a, Ins99b]

10.2.1 Certificate Issuing

Because confirming the validity of the association between the key and the identity can't reliably be automated, human involvement in the process is assumed.

The certificate request is generated by an arbitrary agent, in the common case this is an user agent, based on the explicit interactive trigger by a user; a prototypical interactive application GUI can be seen in Figure 10.2. The corresponding CAL category object definition as contained in an appropriate ontology is given in Figure 10.1, they are direct mappings of the [RSA93b] standard. For transporting the request to the CAA, it is signed and wrapped in a PKCS#10 object.

```
(cat CertificateRequest
     // the java object as provided by the IAIK library
     (reqValue class:iaik.pkcs.pkcs10.CertificateRequest)

     // the following values are all contained in the
     // above object already but are used for the
     // agent's reasoning mechanisms

     (certificationRequestInfo CertificationRequestInfo needed)
     (signatureAlgorithm       AlgorithmIdentifier      needed) )
(cat CertificationRequestInfo
     (version              int (default 0)      needed)
     (subject              DomainName           needed)
     (subjectPublicKeyInfo SubjectPublicKeyInfo needed)
     (attributes           Attribute[]) )
(cat Attribute
     (AttrType             ObjectId)
     (value                class{}:java.lang.Object) )
```

Figure 10.1: Certificate Request Categories

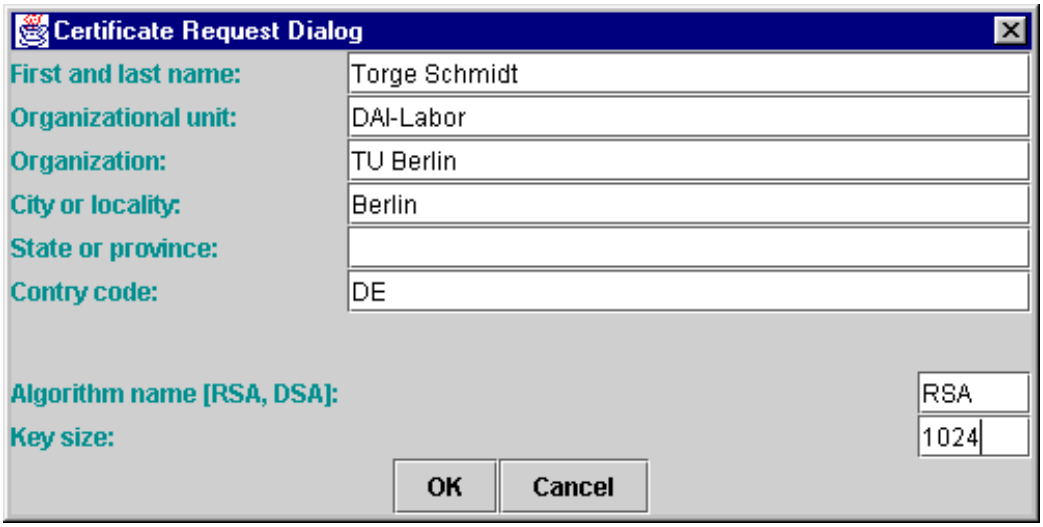

Figure 10.2: Certificate Request GUI

If instead, based on the policy definition, the CA is available online for accepting certificate requests, it would have to offer a respective service to agents. Although, again, the confirmation, that the key belongs to the identity can reliably be achieved only by out-of-bands mechanisms between humans in a different workflow.

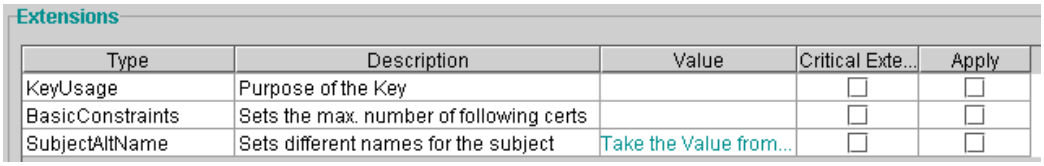

Figure 10.3: Certificate Extensions GUI

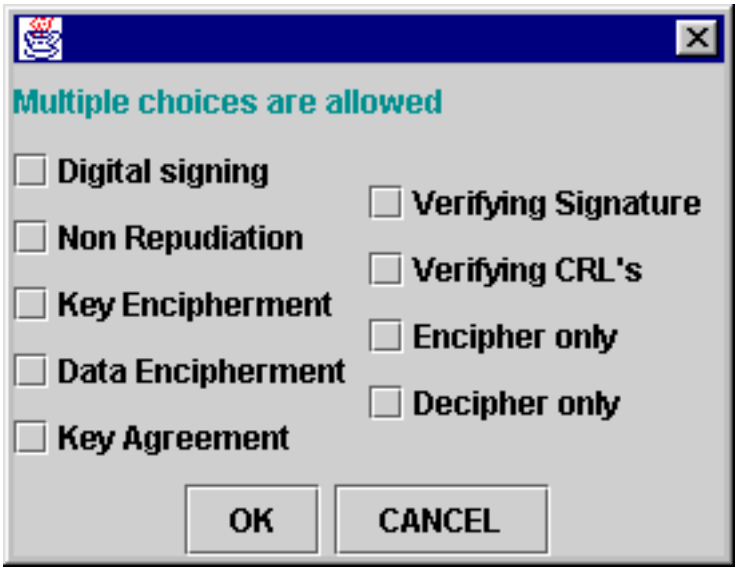

Figure 10.4: Certificate Usage GUI

Figure 10.3 shows the GUI of the CAA defining the extensions of end-user certificates. Clicking on the `Value` column in the `KeyUsage` line opens up a window as can be seen inFigure 10.4. The certificate can now be restricted to certain operations. With the interactive window as shown in Figure 10.5, the `BasicConstraint` that defines the permissable certificate chain length following the issued certificate can be set. (cf. Section 7.4)

Figure 10.5: Certificate Chain Length GUI

10.2.2 Certificate Distribution

After the certificate has been issued, it must be distributed to the requesting person, and ultimately to its user agent. The exact process again depends on the policy of the CA. A typical out-of-band way would be either to save the certificate on a portable media like a floppy or a chipcard. In the prototype implementation, this is done as a DER encoded X.509 certificate. [cci88a]

The new certificate has to be made publicly available as well, so that communication partners of the certificate's subject are enabled to verify the communication based on the public part of the key pair as contained in the certificate. Therefore, the prototypical implementation of the CAA provides for a means to distribute the certificates of the CAA to associated key distribution centers.

In the context of the prototype, the certificate management component of security agents offer a service to receive certificates. The service has to be protected with service control mechanisms, so that only trusted CAAs may manipulate the certificate knowledge base of the SAs. It is important to note, that it is not the CAA that provides the service, it just acts in the user role of the service provided by the SA. The CAA still doesn't offer a service and can't be reached online, depending on the above-mentioned certificate issuing policy, of course.

10.2.3 Certificate Revocation

A working certificate revocation mechanism plays an important part in a certificate-based authentication framework as proposed in [ITU97b]. Certificate revocation becomes necessary, if the secret key belonging to the public key as certified to be associated with a subject's identity becomes compromised, perhaps due to loss of the media holding the private key or eavesdropping on the storing system.

If a certificate is compromized, the owner informs the CA and requests it to revoke the certificate. After successful authentication, that the rightful owner of the certificate is trying to invalidate it, the request is processed and the identifier of the certificate is added to a revocation list, that is managed by the CA.

```
(cat RevocationRequest
     // the encoded request
     (encoded      byte[])

     // the following values are all contained in the
     // above object already but are used for the
     // agent's reasoning mechanisms

     // unique identifier for the certificate,
     // in conjunction with...
     (serialNumber CertificateSerialNumber needed)
     // ...issuing CA
     (issuer       DomainName              needed)
     // since when it is known that the certificate
     // is compromized
     (badSinceDate DateTime) )
```

Figure 10.6: Certificate Revocation Request Category

Figure 10.6 shows a simple category definition for the revocation of certificates. It must be directed to the CA that issued the certificate. Again, the method of delivery to the CA depends on the policy. The CA might offer a service for online access, or a dedicated revocation agent handles the requests and forwards it to the CA; or, the CA has to import the revocation request from a file that was moved to the CA's system by an out-of-band mechanism.

Because no online protocol has found any wide-spread use yet, and the check of the revocation request should involve manual authentication as well, for

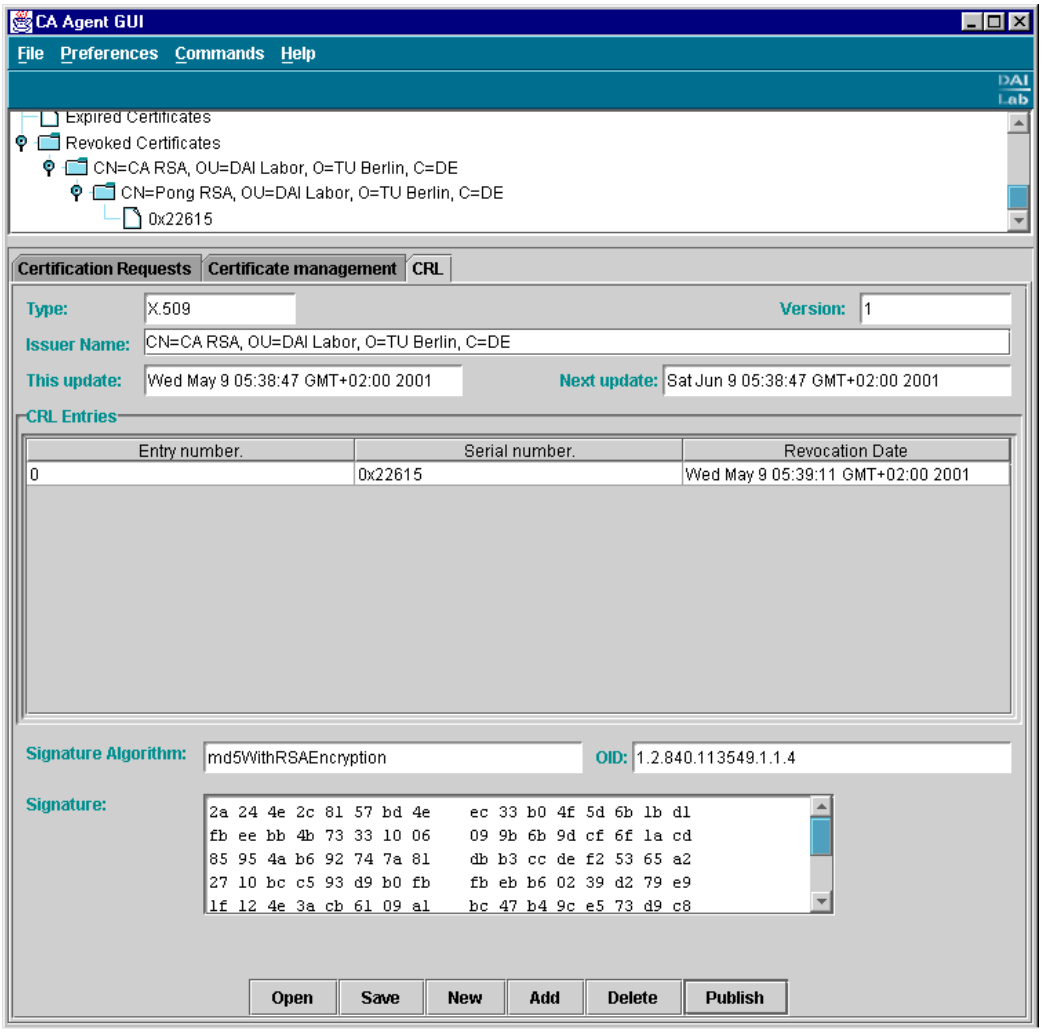

Figure 10.7: Certificate Revocation GUI

the prototypical implementation it is assumed, that an operator uses a GUI to invalidate certificates. Figure 10.7 shows the respective operator window of the CAA.

The effect of using the GUI is to generate a certificate revocation list (CRL). In that list each CA holds certificates that have been revoked prior to the end of their validity lifetime. The certificate is uniquely identified with the serial number of the certificate in conjunction with the CA subject name. The corresponding ontology objects were given in Figure 7.8.

The CRL is propagated to the associated KDCs. In the context of the agent architecture, Security Agents fulfill that role. For better scalability, the proposed KDCs themselves may form a hierarchy, so that the CRLs get pushed over several levels without involving the CA. That is purely a question of deployment depending on the architectural analysis of the system to be realized.

Each CRL might have a proposed time attached, when the next CRL will be published by the CA. (cf. Figure 3.11) KDCs may use this value to proactively seek for a newer CRL at their known CA(s). Again, depending on the policy, the CA might either not be available online at all, or there is a dedicated agent "near" the CA answering the service requests, or the CA might answer itself.

10.3 Security Service Agent

To enable agents to take part in more complex and legally binding protocols, as far as this is possible due to restrictions by law, (cf. Section 2.6) the proposed infrastructure adds extended services to the environment.

To show the business potential of the infrastructure, these value-added services are provided by a separate agent, the Security Service Agent (SSA). The locality of the agent is not obligated to be somewhere "near" any of the aforementioned agents, though it is advisable to have fair access to Certificate Revocation Lists. Each of the following services might be required to be subscribed to, and can be offered on a pay-per-use basis. To reduce system load, each of the services can be offered by several agents.

As examples for these value-added services, three prototypical ones are shown: certificate check, time-stamping, and an arbitrated contract conlucion service. These can be used for building e-commerce applications.

10.3.1 Check Certificate Service

If an agent has the appropriate component and functionality of certificate checking, it could potentially check received certificates itself. (cf. Section 7.4) This might be a lengthy process, though, because it involves retrieving several certificates, until it ultimately leads to the certificate of a known CA. This repeated certificate retrieval adds significant overhead to the communication of an agent. Even worse, afterwards the agent has to retrieve the appropriate CRLs and has to test each of the certificates in the chain against the respective CRL.

That overhead, latency, and need for data storage can be reduced and an agent will get smaller by "outsourcing" the certificate verification process as a separate service to a dedicated agent, in this case the SSA.

Then, if an agent gets presented an unknown certificate it would invoke the respective Check Certificate Service at a well-known SSA. The SSA would check the certificate and send the result back to the requesting agent. To circumvent spoofing attacks, the communication with the SSA must be authenticated. The requesting agent then has only one certificate to check, namely that of the SSA, which can be done beforehand its potential journey, if it is a mobile agent, and noted as being trusted in the agent's knowledge-base for the duration of its travel. On the SSA side, check results might be cached and allow for much faster responses than agents would be able to achieve if they'd work it out alone.

10.3.2 Time Stamp Service

Another valuable service in the context of mutual contract signing for e-commerce settings is getting a time-stamp for some data. That can later be used by an entity to prove that at a particular moment it was in possession of certain information. Relevant applications include disputes over copyright or patent issues, where it is important to reliably prove who was the first.

A typical timestamp service has the following properties: [Sch96] First, it must be impossible to change a single bit of the document without that change being apparent. Second, it must be impossible to timestamp a document with a date and time different from the present one.

In a typical realization as presented in the prototype, a trusted third party is involved, especially to ensure the second of the above properties.

Arbitrated Protocol

The naïve way of the user (also known as *Alice*) sending the whole document to the arbitrator (named *Trent*) has some obvious disadvantages: it creates a huge amount of transfer volume, the arbitrator has to store it and needs an ever enlarging database, and the privacy of the document's content gets violated. The chance of a transmission error happening increases with the transmitted volume as well.

Therefore, instead of sending the entire document, Alice only submits a secure hash of the document. Due to the properties of a cryptographic hash function, if she can later present a document that when hashed produces the hash as signed by Trent, it can be deduced that she must have been in possession of the document at the time of signing the hash. (cf. Section 3.4.3)

```
1  (cat ArbitratedTimeStampRequest
2       (hash          byte[]          needed) )
3  (cat ArbitratedTimeStamp
4       (hash          byte[]          needed)
5       (timestamp     DateTime        needed)
6       (signer        DomainName      needed)
7       (signatureAlgorithmIdentifier
8                      AlgorithmIdentifier needed)
9       // the signature itself
10      (encrypted     byte[]          needed) )
```

Figure 10.8: Arbitrated Timestamp Categories

The service use-case is simple. The provider Trent offers a service for signing arbitrary data called `hash` in the category objects as given in Figure 10.8. The user agent sends the data and receives back a timestamp and a signature with the subject identifier of the certificate that's respective private key was used to sign the data.

10.3.3 Contract Conclusion Service

One common situation in real-world business is mututal contract signing. Neither party of a contract wants to sign it out of fear for be taken in by the other party. The other part of the problem is to later prove the signing process and to call for an independent witness. This is normally solved by involving a notary as a trusted third party. Both signers meet there and sign the contract under supervision of the notary who then testifies the correct process and attests the validity of the contract.

In the context of an agent environment, this functionality can be realized by providing a specialized service for mutual contract signing. Figure 10.9 is an UML sequence diagram showing the service usage, the CAL category definitions are given in Figure 10.10.

The contract data to be signed has to be agreed upon between both parties before this service use can begin; it is not part of the contract conclusion protocol itself.

In the CASA framework, a service is between exactly two entities, the user and the provider. Therefore, the service provisioning from the arbitrator Trent is shown as two distinct service usages that are synchronized. The synchronization is done by the mechanisms of the knowledge base in contrast to the explicit communication as shown in Figure 10.9, plain UML lacks the power to express the underlying mechanisms.

Again, not the contract itself is sent, but only a hash thereof. Both sides send their signed hash over the contract data to the arbitrator Trent. That agent checks each signature and compares the contract data. If all is well so far, Trent informs both service using agents informally, that the other side has signed. To circumvent race problems and cheating by denial of service attacks at critical stages of the protocol, Trent waits for a confirmation from both sides before creating the real signature. It then sends its signed data to both parties and waits for their acceptance statement. If both sides confirm the receipt of the correct signature, the contract is concluded. If one of the sides do not confirm the correctness of the process, an immediate offline escalation process has to be initiated.

A provider of the contract conclusion service might offer this as a value-added service for e-business applications and charge money for it. For billing purposes, and so that no third party might spoof the confirmation messages, the service requirements should include authentication.

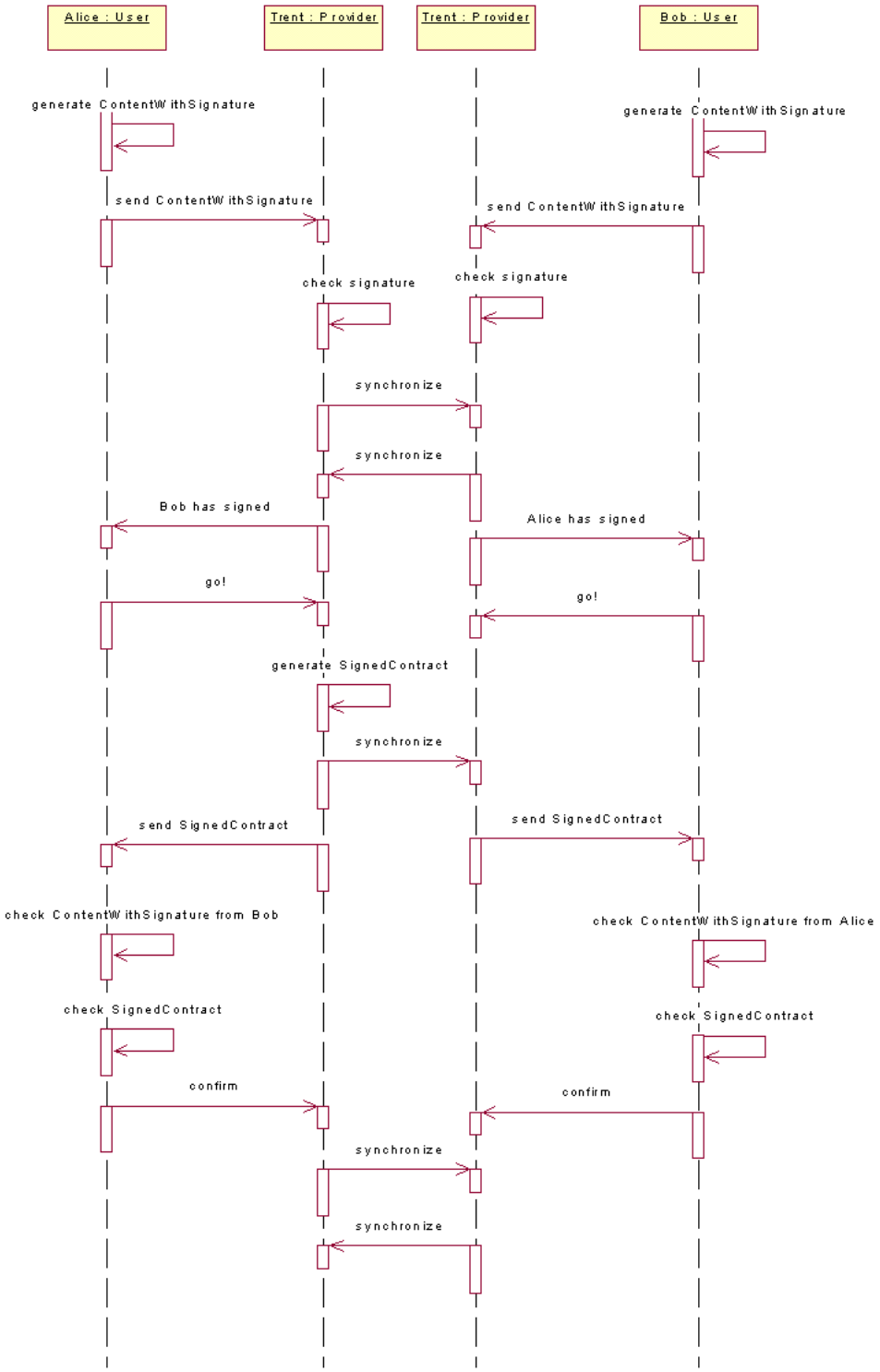

Figure 10.9: Simultaneous Contract Signing Protocol

```
1  (cat ContentWithSignature
2       (hash          byte[]          needed)
3       (signer        DomainName      needed)
4       (signatureAlgorithmIdentifier
5                      AlgorithmIdentifier needed)
6       // the signature itself
7       (encrypted     byte[]          needed) )
8  (cat SignedContract
9       (signatureA    ContentWithSignature needed)
10      (signatureB    ContentWithSignature needed)
11      (timestamp     DateTime        needed)
12      // the arbitrator
13      (signer        DomainName      needed)
14      (signatureAlgorithmIdentifier
15                     AlgorithmIdentifier needed)
16      // the signature itself
17      (encrypted     byte[]          needed) )
```

Figure 10.10: Simultaneous Contract Signing Categories

10.4 Summary

Introducing an agent-based certificate infrastructure enables to define certificates based on a known and trusted root. Companies are enabled to provide their one internal or external certificate root without having to trust an unknown third party. Further, implementing a certificate distribution architecture based on agents allows for other agents to use their native communication language for accessing certificates and associated data. The agents are enabled to pro-actively use the data of certificates and reason about the content. No implementation of proprietary protocols for each specific CA is needed.

Value-added services are given as examples for security- and e-commerce-related features possible in an agent environment. They can be used as building blocks for a more comprehensive market system out of the scope of this work.

The thesis ends with the following Chapter 11, summarizing the work done, comparing it to other works in the field, and giving an outlook into the future.

Chapter 11
Conclusions

"This is the end
Beautiful friend
This is the end
My only friend, the end
Of our elaborate plans, the end
Of everything that stands, the end."

The Doors

11.1 Synopsis

This final Chapter will wrap up the whole thesis work. The results are compared against the requirements determined previously. The paper is set into relation to other works in the field and specific achievements are highlighted. A look into the future shows, where more development can be put into to provide for a even more sophisticated solution for agent-based e-commerce application in the telecommunicaiton domain.

11.2 Analysis

The presented work fulfills all of the domain demands of Part I and of the technical requirements as analyzed in Chapter 6.

Specifically, data integrity, cryptographic entity authentication, and confidentiality of communication is preserved by employing standardized protocols for inter-agent communication. Authentication is based on hierarchical certificates. Authorization of service use is enabled by providing service control mechanisms.

Code mobility is supported by securing the executing agent platform against incoming code. Further, mobile code is protected as much as possible against tampering with and spying upon by malicious systems. If a fraud attempt is detected, agent place trust lists are given to react upon the incident.

The autonomy of agents and demands by regulations of cryptography by law is accomodated for by giving agents a way to reason about its security-related capabilities and to adjust its own composition accordingly. Its capabilities can be parametrized. Users can specify a set of functionality that is required for service usage or provisioning.

Agent places and domains of agents can be additionally secured by mechanisms on the community and global level. Not only is it possible for the systems of an agent place to make assumptions about remote agent places, but in addition services are provided to enable secure and legally binding contracts as much as possible. Those mechanisms and service can be used in other work to devise secured agent-based markets for real goods exchange.

11.3 Achievements and Related Work

The main achievement of this work is the introduction of a comprehensive framework for secure agent systems. The architecture as proposed encompasses several levels of the resulting systems:

- The level of Java programming, basic runtime security, and raw communication.

- Security of the agent itself, its consciousness and ability to reason about its own mechanisms, and of the communication with other agents, extending to service usage authorization.
- Agent places are enriched with features for securing against malicious mobile code and other fraudulent places.
- The certificate infrastructure is rooted globally, certificate revocation is supported, and value-added services are provided for the realization of agent-based e-commerce solutions.

This sum of features and functionalities haven't been found in other agent systems yet. Typical means so far include using SSL for communication and authentication of agents based on certificates, though the important aspect of certificate revocation was no issue anywhere. Resource protection is a rare commodity. None of the analyzed agent platforms provide for any agent-specific features making use of agent's typical properties like autonomy or self-inspection. Especially the very important feature of authorization is completly neglected by all other platforms. New in the ASITA framework is the introduction of mechanisms on the community and global level as well.

The resulting system is a vastly expanded approach to security for agent systems, supporting mobile agents and withstanding the typical set of attacks on communicating systems. An accompanying first prototypical implementation of an extension to the CASA platform in the form of the first inception of JIAC IV developed at the DAI-Labor prove the feasibility of the approach. [FBK+01]

11.4 Further Work

This work can be expanded on multiple tracks. On the technical level, there is a new Java Development Kit on the Horizon, tagged 1.4. [Sun00b]. That version will be delivered with the SSL component which has been added in this work in the form of the ISASILK library. [Ins99b] Other cryptographic functions, like an improved certificate handling, is included as well, in this work it is built on the Java Cryptography Extension implementation of IAIK. [Ins99a]

In addition to the SSL communication, the Internet Protocol based IPSec communication mechanisms can be chosen to be able to communicate with

system running that slowly deploying standard. [KA98] This might get momentum if the IP protocol in version 6 will roll-out widely. [DH98]

The hierarchical root of certificates has several problems in its trust association and need for a central certifiying site. Certificate revocation is cumbersome as well. New ideas in the expressive power and management of certificates led to the Simple Public Key Infrastructure. [Ell99a, Ell99b] Employing those certificates might reduce the workload for certificate distribution and validation. Further, they support credentials, authority delegation, and anonymity, adding to the possibilities available to future service access and management. Another certificate-related track is the introduction of Registration Authorities as suggested by [AF99].

The trust relationship as introduced in this work is not too elaborated. There are at several helpful ways for enhancements. One could be to extend the agent place trust lists as developed in Section 9.3.2 to include communication allowance with other places so that agents can decide not only if they should migrate to remote places as it is now, but they might even refuse communication altogether with a malicious remote site. The level of trust itself could be quantified instead of a dual choice as presented here. Trust engines could be used to evaluate trust decisions more formally and automated. [BFIK99] Evenmore, the trust relationships should be more discriminating.

To support development and management of a running agent-based system, tools have to be developed. This is especially true the more complex a system gets. Specifically, a tool should allow to define service security requirements (cf. Section 8.2.3) and service authorization (cf. Section 8.6) and to assign them to agents.

For added value for real-world goods exchange inclusion of payment methods must be promoted and a reliable and accountable framework for electronic goods delivery is needed, building on the means provided here.

11.5 Summary

This work provided an encompassing and advanced security infrastructure for multi-agent-applications in the telematic area. The summary can best be shown by key pictures of the work. Figure 11.1 depicts the introduced security-related components of an agent. In Figure 11.2 it is seen, how a communication is processed and which measures are taken before it is acted

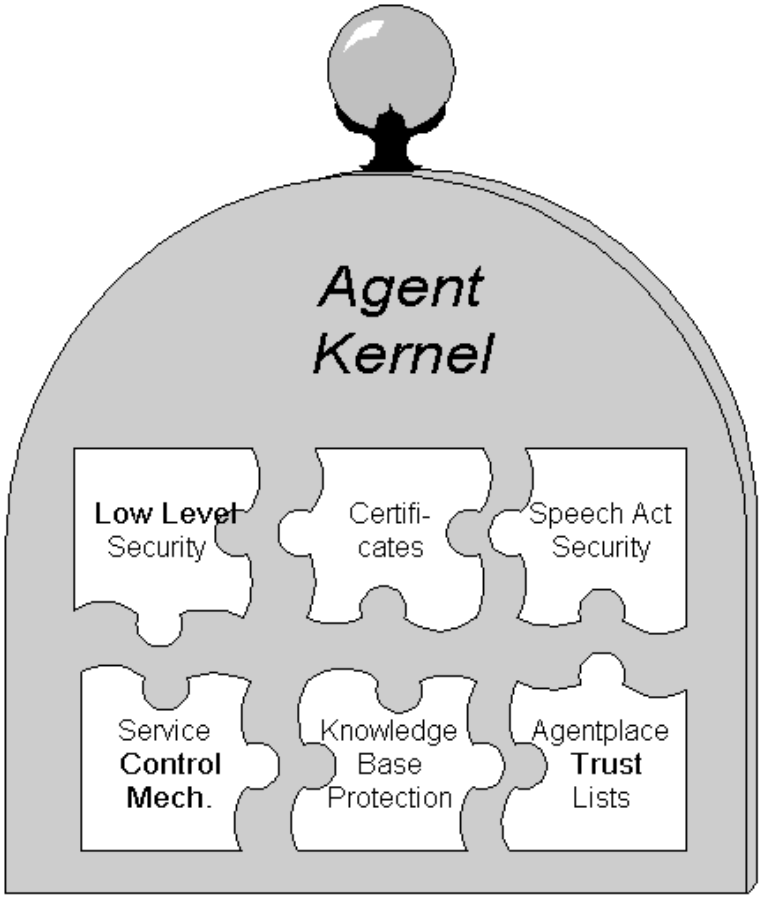

Figure 11.1: Security Components (again)

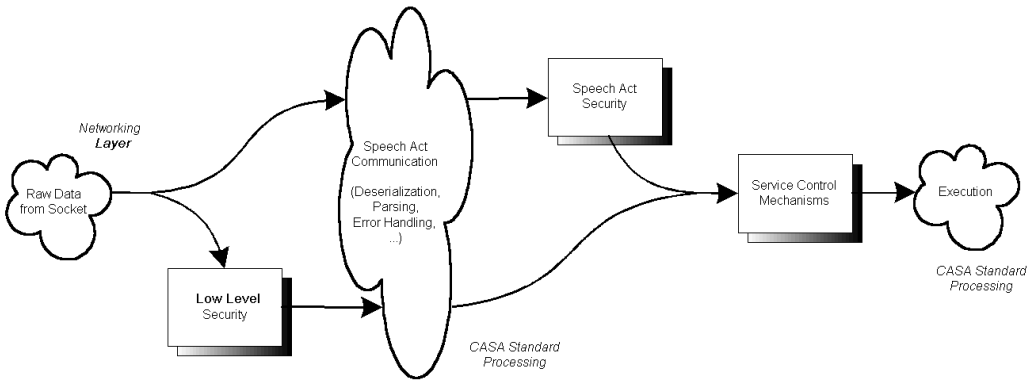

Figure 11.2: Communication Security (again)

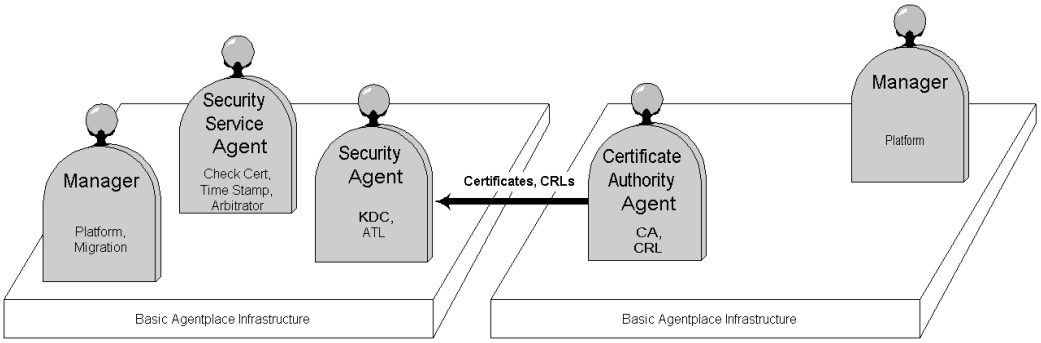

Figure 11.3: Community Security (again)

upon by the single agent. The graphic in Figure 11.3 shows agents with security tasks on distributed agent places.

"Ich habe fertig!"

Giovanni Trapattoni

Part IV

Appendices

Appendix A
CAL

The Appendix starts with a brief introduction to the CASA Agent Description Language (CAL). The exhaustive definition is given in [Ses02]. Longer CAL excerpts from the implementation follow. They are referenced and explained in the running text.

A.1 CAL Introduction

CAL is the agent description language of CASA. It is used to describe an agent on the knowledge level, and is employed to define terminologic, declarative, procedural, and communicative knowledge. The declarative part allows the representation of world states, based on ontologies. It is a combination of logic and object-oriented concepts. Values have types, categories are used to group objects, functions and relations as operators, and meta-knowledge types.

Ontologies in CAL are defined as follows:

```
Ontology   =   (ont OntoName1 (incl OntoName2+) OntoDecl*)
OntoDecl   =   CategoryDecl | FunctionDecl | ComparisonDecl
```

The ontology definition is preceeded by the literal `ont`. Then, the name of the ontology is given as `OntoName1`. Optionally, an ontology may be derived from other ontologies `OntoName2`,... This works as inheritance in object-oriented languages. An ontology consists of declarations of categories, functions, and comparisons.

Categories represent objects and are defined within ontologies as follows:

CategoryDecl = (cat CatName1 (ext CatName2+) AttributeDecl*)
AttributeDecl = (AttName Type Keyword*)

`CategoryDecl` defines the declaration of a category. `cat` is a literal. `CatName1` is the arbitrary name of the category defined. `(ext CatName2+)` is optional and allows to derive CatName1 from the categories CatName2,... This works as inheritance in object-oriented languages. A category might consist of several attributes, whose definition follow. An attribute has at least a given name and a type. Optionally, an attribute might further be specified, e.g. with constraints. Typical `Keyword`'s are `needed` to declare, that this attribute must contain a known value, and `default`, which allows to initialize an attribute instance upon inception to a pre-defined value. `unique` declares, that no other object instance of the same category might contain the same value in this attribute.

Functions and comparisons are defined as follows:
FunctionDecl = (fun Type1 FunName Type2*)
ComparisonDecl = (comp CompName Type3+)

Each declaration is preceeded by the respective literal, `fun` or `comp`. For a function, the type `Type1` of its value is given. It follows its name `FunName`. It follows a possible empty enumeration of types for the positional parameters of the function. For a comparison, at least one `Type3` parameter must be given.

In the JIAC reference realization of CASA, an object can be instantiated as follows:
ObjInst = (obj Ident Categ AttribInst*)
AttribInst = (AttName Value)

The literal `obj` starts the declaration. `Ident` is an identifiert for the object, which must be unique in the fact base. `Categ` is the category, of which this object is an instance of. It follows a list of attribute initializations. The list of initialized attributes must at least consist of all attributes marked as `needed` in the category declaration.

A.2 Component Objects

```
// Objects representing Components
(obj CMC     SecurityComponent
     (identifier     "CMC")
     (name           "CertificateManagement") )
(obj KBP     SecurityComponent
     (identifier     "KBP")
     (name           "KnowledgeBaseProtection") )
(obj SAS     SecurityComponent
     (identifier     "SAS")
     (name           "SpeechActSecurity") )
(obj LLS     SecurityComponent
     (identifier     "LLS")
     (name           "LowLevelSecurity")
     // not "transport layer security" because that
     // contflicts with the name of the successor of SSL
)
(obj SCM     SecurityComponent
     (identifier     "SCC")
     (name           "ServiceControlMechanisms") )
(obj MTM     SecurityComponent
     (identifier     "MTC")
     (name           "MarketplaceTrustMechanisms") )
```

A.3 Ability Objects

A.3.1 Certificate Management

```
////////////////////////////////////////////////////////////
// Objects representing abilities of certificate management
(obj CMCPROVIDECERTIFICATES  AgentAbility
     (component      CMC)
     (name           "CMC_PROVIDE_CERTIFICATES") )
(obj CMCCOMMUNICATECERTIFICATES      AgentAbility
     (component      CMC)
     (name           "CMC_COMMUNICATE_CERTIFICATES") )
```

```
// an enumeration of the supported certificate types
// (see SSL standard)
(obj CMCCERTRSASIGN          AgentAbility
     (component      CMC)
     (name           "CMC_CERT_RSA_SIGN") )
(obj CMCCERTDSSSIGN          AgentAbility
     (component      CMC)
     (name           "CMC_CERT_DSS_SIGN") )
(obj CMCCERTRSAFDH           AgentAbility
     (component      CMC)
     (name           "CMC_CERT_RSA_FIXED_DH") )
(obj CMCCERTDSSFDH           AgentAbility
     (component      CMC)
     (name           "CMC_CERT_DSS_FIXED_DH") )
(obj CMCCERTRSAEDH           AgentAbility
     (component      CMC)
     (name           "CMC_CERT_RSA_EPHEMERAL_DH") )
(obj CMCCERTDSSEDH           AgentAbility
     (component      CMC)
     (name           "CMC_CERT_DSS_EPHEMERAL_DH") )
```

A.3.2 Knowledge Base Protection

```
(obj KBPAUTHENTICATEDSAFE    AgentAbility
     (component      KBP)
     (name           "KBP_AUTHENTICATED_SAFE") )
```

A.3.3 Low Level Security

```
////////////////////////////////////////////////////////////////
// Objects representing abilities of transport layer security,
// they are a direct canonical mapping of the cipher suites as
// defined in the SSL standard
(obj SSLNULLWITHNULLNULL             AgentAbility
     (component      LLS)
     (name           "SSL_NULL_WITH_NULL_NULL") )

// The following CipherSuite definitions require that the server
// provide an RSA certificate that can be used for key exchange.  The
// server may request either an RSA or a DSS signature-capable
// certificate in the certificate request message.
```

```
13
14 (obj SSLRSAWITHNULLMD5               AgentAbility
15      (component      LLS)
16      (name           "SSL_RSA_WITH_NULL_MD5") )
17 (obj SSLRSAWITHNULLSHA               AgentAbility
18      (component      LLS)
19      (name           "SSL_RSA_WITH_NULL_SHA") )
20 (obj SSLRSAEXPORTWITHRC440MD5        AgentAbility
21      (component      LLS)
22      (name           "SSL_RSA_EXPORT_WITH_RC4_40_MD5") )
23 (obj SSLRSAWITHRC4128MD5             AgentAbility
24      (component      LLS)
25      (name           "SSL_RSA_WITH_RC4_128_MD5") )
26 (obj SSLRSAWITHRC4128SHA             AgentAbility
27      (component      LLS)
28      (name           "SSL_RSA_WITH_RC4_128_SHA") )
29 (obj SSLRSAEXPORTWITHRC2CBC40MD5     AgentAbility
30      (component      LLS)
31      (name           "SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5") )
32 (obj SSLRSAWITHIDEACBCSHA            AgentAbility
33      (component      LLS)
34      (name           "SSL_RSA_WITH_IDEA_CBC_SHA") )
35 (obj SSLRSAEXPORTWITHDES40CBCSHA     AgentAbility
36      (component      LLS)
37      (name           "SSL_RSA_EXPORT_WITH_DES40_CBC_SHA") )
38 (obj SSLRSAWITHDESCBCSHA             AgentAbility
39      (component      LLS)
40      (name           "SSL_RSA_WITH_DES_CBC_SHA") )
41 (obj SSLRSAWITH3DESEDECBCSHA         AgentAbility
42      (component      LLS)
43      (name           "SSL_RSA_WITH_3DES_EDE_CBC_SHA") )
44
45 // The following CipherSuite definitions are used for
46 // server-authenticated (and optionally client-authenticated)
47 // Diffie-Hellman.  DH denotes cipher suites in which the server's
48 // certificate contains the Diffie-Hellman parameters signed by the
49 // certificate authority (CA).
50
51 (obj SSLDHDSSEXPORTWITHDES40CBCSHA   AgentAbility
52      (component      LLS)
53      (name           "SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA") )
54 (obj SSLDHDSSWITHDESCBCSHA           AgentAbility
55      (component      LLS)
```

```
56        (name           "SSL_DH_DSS_WITH_DES_CBC_SHA") )
57   (obj SSLDHDSSWITH3DESEDECBCSHA       AgentAbility
58        (component      LLS)
59        (name           "SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA") )
60   (obj SSLDHRSAEXPORTWITHDES40CBCSHA   AgentAbility
61        (component      LLS)
62        (name           "SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA") )
63   (obj SSLDHRSAWITHDESCBCSHA           AgentAbility
64        (component      LLS)
65        (name           "SSL_DH_RSA_WITH_DES_CBC_SHA") )
66   (obj SSLDHRSAWITH3DESEDECBCSHA       AgentAbility
67        (component      LLS)
68        (name           "SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA") )
69
70   // DHE denotes ephemeral Diffie-Hellman,
71   // where the Diffie-Hellman parameters are signed by a DSS or RSA
72   // certificate, which has been signed by the CA.  The signing
73   // algorithm used is specified after the DH or DHE parameter.  In all
74   // cases, the client must have the same type of certificate, and must
75   // use the Diffie-Hellman parameters chosen by the server.
76
77   (obj SSLDHEDSSEXPORTWITHDES40CBCSHA  AgentAbility
78        (component      LLS)
79        (name           "SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA") )
80   (obj SSLDHEDSSWITHDESCBCSHA          AgentAbility
81        (component      LLS)
82        (name           "SSL_DHE_DSS_WITH_DES_CBC_SHA") )
83   (obj SSLDHEDSSWITH3DESEDECBCSHA      AgentAbility
84        (component      LLS)
85        (name           "SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA") )
86   (obj SSLDHERSAEXPORTWITHDES40CBCSHA  AgentAbility
87        (component      LLS)
88        (name           "SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA") )
89   (obj SSLDHERSAWITHDESCBCSHA          AgentAbility
90        (component      LLS)
91        (name           "SSL_DHE_RSA_WITH_DES_CBC_SHA")
92        (requires       { AgentAbility : CMCPROVIDECERTIFICATES } ) )
93   (obj SSLDHERSAWITH3DESEDECBCSHA      AgentAbility
94        (component      LLS)
95        (name           "SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA")
96        (requires       { AgentAbility : CMCPROVIDECERTIFICATES } ) )
97
98   // The following cipher suites are used for completely anonymous
```

```
99  // Diffie-Hellman communications in which neither party is
100 // authenticated.  Note that this mode is vulnerable to
101 // man-in-the-middle attacks and is therefore strongly discouraged.
102
103 (obj SSLDHANONEXPORTWITHRC440MD5     AgentAbility
104      (component      LLS)
105      (name           "SSL_DH_ANON_EXPORT_WITH_RC4_40_MD5") )
106 (obj SSLDHANONWITHRC4128MD5          AgentAbility
107      (component      LLS)
108      (name           "SSL_DH_ANON_WITH_RC4_128_MD5") )
109 (obj SSLDHANONEXPORTWITHDES40CBCSHA  AgentAbility
110      (component      LLS)
111      (name           "SSL_DH_ANON_EXPORT_WITH_DES40_CBC_SHA") )
112 (obj SSLDHANONWITHDESCBCSHA          AgentAbility
113      (component      LLS)
114      (name           "SSL_DH_ANON_WITH_DES_CBC_SHA") )
115 (obj SSLDHANONWITH3DESEDECBCSHA      AgentAbility
116      (component      LLS)
117      (name           "SSL_DH_ANON_WITH_3DES_EDE_CBC_SHA") )
```

A.3.4 Speech Act Security

```
1  ////////////////////////////////////////////////////////////////
2  // Objects representing abilities of speech act security
3  // There are different objects for use in-session and
4  // out-of-session
5
6  // If no sessions are wanted there is no key exchange.  then,
7  // the sender defines the symmetric key.
8  // format for "name":
9  //  SAS_<KeyExchange>_<Authen>_<Signature|Mac>_<symmetric Encrytion>
10 // format for <Signature> Method::==Algorithm[/param]
11 // format for <symmetric Encryption>::==Algorithm[/param][_strength]
12
13 (obj SASRSARSASHANULL               AgentAbility
14        (component      SAS)
15        (name           "SAS_RSA_RSA_SHA_NULL") )
16 (obj SASRSARSASHADES40CBC            AgentAbility
17        (component      SAS)
18        (name           "SAS_RSA_RSA_SHA_DES40CBC") )
19 (obj SASRSARSASHAIDEACBC             AgentAbility
20        (component      SAS)
```

```
21          (name           "SAS_RSA_RSA_SHA_IDEACBC") )
22  (obj SASRSARSASHARC440              AgentAbility
23          (component      SAS)
24          (name           "SAS_RSA_RSA_SHA_RC4_40") )
25  (obj SASRSARSASHARC4128             AgentAbility
26          (component      SAS)
27          (name           "SAS_RSA_RSA_SHA_RC4_128") )
28  (obj SASDHRSAHMACSHARC4128          AgentAbility
29          (component      SAS)
30          (name           "SAS_DH_RSA_HMAC/SHA_RC4_128") )
```

A.3.5 Service Control Mechanisms

```
1  ////////////////////////////////////////////////////////////
2  // Objects representing abilities of service use authorization
3  // service control
4  (obj SCMSERVICEAUTHORIZATION        AgentAbility
5          (component      SCM)
6          (name           "SCM_SERVICE_AUTHORIZATION") )
7  // management of service control mechanisms by speech acts
8  (obj SCMCOMMUNICATESCLS             AgentAbility
9          (component      SCM)
10         (name           "SCM_COMMUNICATE_SCLS")
11         (requires       { AgentAbility : CMSPROVIDECERTIFICATES } ) )
```

A.4 Security Dependencies Objects

Here dependencies between abilities are defined.

```
1  // Objects representing requirements for the KBP component
2  (obj KBPWITHRSAREQ                  PerAbilityRequirement
3       (ability        KBPWITHRSA)
4       (requiresOneOf  [ AgentAbility : CMCCERTRSASIGN ]) )
5
6  // Objects representing requirements for the SSL component
7  (obj SSLRSAWITHNULLMD5REQ           PerAbilityRequirement
8       (ability        SSLRSAWITHNULLMD5)
9       (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
```

```
(obj SSLRSAWITHNULLSHAREQ            PerAbilityRequirement
     (ability        SSLRSAWITHNULLSHA)
     (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
(obj SSLRSAEXPORTWITHRC440MD5REQ     PerAbilityRequirement
     (ability        SSLRSAEXPORTWITHRC440MD5)
     (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
(obj SSLRSAWITHRC4128MD5REQ          PerAbilityRequirement
     (ability        SSLRSAWITHRC4128MD5)
     (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
(obj SSLRSAWITHRC4128SHAREQ          PerAbilityRequirement
     (ability        SSLRSAWITHRC4128SHA)
     (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
(obj SSLRSAEXPORTWITHRC2CBC40MD5REQ  PerAbilityRequirement
     (ability        SSLRSAEXPORTWITHRC2CBC40MD5)
     (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
(obj SSLRSAWITHIDEACBCSHAREQ         PerAbilityRequirement
     (ability        SSLRSAWITHIDEACBCSHA)
     (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
(obj SSLRSAEXPORTWITHDES40CBCSHAREQ  PerAbilityRequirement
     (ability        SSLRSAEXPORTWITHDES40CBCSHA)
     (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
(obj SSLRSAWITHDESCBCSHAREQ          PerAbilityRequirement
     (ability        SSLRSAWITHDESCBCSHA)
     (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
(obj SSLRSAWITH3DESEDECBCSHAREQ      PerAbilityRequirement
     (ability        SSLRSAWITH3DESEDECBCSHA)
     (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
(obj SSLDHDSSEXPORTWITHDES40CBCSHAREQ PerAbilityRequirement
     (ability        SSLDHDSSEXPORTWITHDES40CBCSHA)
     (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
(obj SSLDHDSSWITHDESCBCSHAREQ        PerAbilityRequirement
     (ability        SSLDHDSSWITHDESCBCSHA)
     (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
(obj SSLDHDSSWITH3DESEDECBCSHAREQ    PerAbilityRequirement
     (ability        SSLDHDSSWITH3DESEDECBCSHA)
     (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
(obj SSLDHRSAEXPORTWITHDES40CBCSHAREQ PerAbilityRequirement
     (ability        SSLDHRSAEXPORTWITHDES40CBCSHA)
     (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
(obj SSLDHRSAWITHDESCBCSHAREQ        PerAbilityRequirement
     (ability        SSLDHRSAWITHDESCBCSHA)
     (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
(obj SSLDHRSAWITH3DESEDECBCSHAREQ    PerAbilityRequirement
```

```
53      (ability        SSLDHRSAWITH3DESEDECBCSHA)
54      (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
55  (obj SSLDHEDSSEXPORTWITHDES40CBCSHAREQ PerAbilityRequirement
56      (ability        SSLDHEDSSEXPORTWITHDES40CBCSHA)
57      (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
58  (obj SSLDHEDSSWITHDESCBCSHAREQ       PerAbilityRequirement
59      (ability        SSLDHEDSSWITHDESCBCSHA)
60      (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
61  (obj SSLDHEDSSWITH3DESEDECBCSHAREQ   PerAbilityRequirement
62      (ability        SSLDHEDSSWITH3DESEDECBCSHA)
63      (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
64  (obj SSLDHERSAEXPORTWITHDES40CBCSHAREQ PerAbilityRequirement
65      (ability        SSLDHERSAEXPORTWITHDES40CBCSHA)
66      (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
67  (obj SSLDHERSAWITHDESCBCSHAREQ       PerAbilityRequirement
68      (ability        SSLDHERSAWITHDESCBCSHA)
69      (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
70  (obj SSLDHERSAWITH3DESEDECBCSHAREQ   PerAbilityRequirement
71      (ability        SSLDHERSAWITH3DESEDECBCSHA)
72      (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
73
74  // Objects representing requirements for the service control component
75  (obj SCMSERVICEAUTHORIZATIONREQ      PerAbilityRequirement
76      (ability        SCMSERVICEAUTHORIZATION)
77      (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
78  (obj SCMCOMMUNICATESCLSREQ           PerAbilityRequirement
79      (ability        SCMCOMMUNICATESCLS)
80      (requiresOneOf  [ AgentAbility : CMCPROVIDECERTIFICATES ]) )
```

A.5 Service Security Requirements Objects

This is an example of protecting a service called `dummy`. The service shall be available only to authenticated users, security shall be on the transport layer. The certificates shall be based on the RSA algorithm. SHA-1 shall be the hash algorithm, RC4 with 128 bit key length or IDEA in CBC mode is the symmetric encryption algorithm. The user must be authorized to use the service.

```
(obj DummyServiceLLS                AgentAbility
     (component      LLS)
     (name           "Dummy Service Transport Layer Security") )
(obj DummyServiceLLSRequirements     PerAbilityRequirement
     (ability        DummyServiceLLS)
     (requiresOneOf  [ AgentAbility : SSLRSAWITHRC4128SHA SSLRSAWITHIDEACBCSHA ]) )
// combined with authorization
(obj DummyServiceRequirements        ServiceSecurityRequirements
        (requires       { AgentAbility :
                DummyServiceLLSRequirements SCMSERVICEAUTHORIZATION }) )
```

Glossary

Some of these entries stem from [Cyb01], others are from [WM-01]. Entries from [McL93] and [Mat97] are found here as well. Other entries are adapted from [Shi00].

3DES: Abbreviation for *Tripple DES:* Strengthened but even slower version of DES (qv) with effectively its original key (qv) size trippled.

ACC: Abbreviation for *Agent Communication Channel:* Accoding to the FIPA (qv) standard, it is the standard communication channel between agents on an agent platform and between agent platforms.

ACL/1: Abbreviation for *Access Control List:* Mechanism to restrict access to objects to authenticated (qv) and authorized (qv) subjects.

ACL/2: Abbreviation for *Agent Communication Language:* A defined syntax with accompanying semantics for inter-agent communication, e.g. as defined by FIPA. (qv)

Active Attack: The attacker modifies the attacked data stream.

Adjudicator: In a security-aware environment, a neutral party, that, after a dispute about an already completed transaction arises, acts as a judge to set things right.

AI: Abbreviation for *Artifical Intelligence:* The effort to create intelligent behaviour by machines.

AMS: Abbreviation for *Agent Management System:* According to the FIPA (qv) standard, it is the actively managing entity on an agent platform. It controls the life cycle of agents and resource usage, including the ACC. (qv)

Arbitrator: In a security-aware environment, as neutral party in an ongoing communication protocol is charged with ensuring, that a contract conclusion or information exchange is done fairly.

ASITA: Abbreviation for *Advanced Security Infrastructure for Multi-Agent-Applications in the Telematic Area:* The title of this work, and the architecture as developed herein.

ASN.1: Abbreviation for *Abstract Syntax Notation One:* The standard [cci88a] for describing data objects.

ASP: Abbreviation for *Applications Service Provider:* An organization that provides applications to a user's desktop or other computing device, rather than requiring them to invest in local software.

Asymmetric Cipher: Encryption, (qv) where the key to be used to decrypt the ciphertext (qv) is different from the encryption key. Both keys are correlated, but it is a "hard" problem to derive one from the other.

Authentication: It is the process of ensuring, that in a communication protocol a presented identity is impersonated rightly by the presenting entity. Further, an authenticated communication includes ensuring integrity of the data exchanged.

Authorization: Granting authorization is to allow an acting subject an operation on an object. The subject is some kind of active entity, usually a program acting on behalf of an user. The operation depends on the object, in the context of data files this might be read, write, execute, delete, change, and similar operations. The object is normally some kind of passive entity, e.g. a data file or process being acted on. The subject must be successfully authenticated (qv) before authorization can be performed reasonably.

AOT: Abbreviation for *Agent-oriented Technology:* The technology domain of software agents.

API: Abbreviation for *Application Programming Interface:* APIs allow to program to a pre-constructed interface (the API) instead of programming a device or piece of software directly. This allows for faster development, since programming to a device's API is designed to be easier than programming to a device directly.

Applet: A Java program that can be embedded in a web page and which is run by the web browser as the execution environment.

B2B: Abbreviation for *Business to Business:* Refering to electronic trade or partnering between organizations, often over an extranet or e-marketplace.

B2C: Abbreviation for *Business to Consumer:* Referring to electronic trade (e-commerce) between businesses and end consumers, as opposed to B2B e-business between organizations alone. (qv)

BER: Abbreviation for *Basic Encoding Rules:* The standard [CCI88c] for representing ASN.1 (qv) data types as strings of octets.

BDI: Abbreviation for *Belief-Desire-Intention:* A common agent model, that constructs agents as cognitive entities with a view of the world (beliefs), goals to reach (desires), and a plan to reach them (intentions).

Bean: The Java (qv) concept and implementation of a component architecture.

Block Cipher: Encryption (qv) function, that works on a fixed block length of data at a time. The last block of data in a bulk exchange requires padding. (qv)

Block Counter: Extra sequential numbering added to a block of data to gain replay (qv) protection.

Bluetooth: A standard for short-range wireless communication between computing devices and associated peripherals, including laptop and mobile computers, PDAs, (qv) and mobile phones.

Broker: The broker mediates information, often pertaining to the availability of services, between actors.

Brute Force Attack: Trying out all possible combinations. This is feasible for small key sizes and other specifc situations.

BXA: Abbreviation for *Bureau of Export Administration:* The administrative bureau of the Department of Commerce of the United States of America currently in charge of the Export Administration Regulations, covering cryptographic software.

CA: Abbreviation for *Certification Authority:* A trusted third party issuing certificates. (qv)

CAA: Abbreviation for *CA Agent:* In the ASITA (qv) concept implementation the name of an agent fulfilling the role (qv) of a CA. (qv)

CAL: Abbreviation for *CASA Agent Language:* The agent description language used in CASA. (qv) It is described briefly in Section A.1.

CASA: Abbreviation for *Component Architecture for Service Agents:* The agent architecture described in [Ses02].

Certificate: The association between a public key and an identity, signed (qv) by a trusted third party, the CA. (qv)

Cipher: Class of functions for encrypting (qv) and decrypting. (qv) Ciphers deal with the characters of a message, at the syntactical instead of the semantical level as does a code. (qv)

Ciphertext: The result of applying a cipher function to a plaintext (qv): the encrypted output.

Code: A cryptosystem that deals with linguistic units: words, phrases, sentences.

Collision: If it is a "hard" problem to find two random messages that generate the same hash (qv) value, the hash algorithm is said to be collision-free.

Compression Function: Function to reduce the amount of data. Usually, it is a loss-less transformation, but in some cases it incurrs data loss and hence is not reversible, as is the case for hashes. (qv)

Confidentiality: The property that information is not made available or disclosed to unauthorized individuals, entities, or processes.

Connectivity Provider: Offers network access.

Consumer: User of a service.

Content Provider: A special case of a provider, (qv) that only serves content without supporting or wrapping services.

CRL: Abbreviation for *Certificate Revocation List:* A list of revoked certificates (qv) no longer valid.

Cryptanalysis: Trying to break the security of a message.

Cryptanalyst: Person working on the field of cryptanalysis. (qv)

Cryptographer: Person working on the field of cryptography. (qv)

Cryptography: The science of keeping messages secure.

Cryptologist: Person working on the field of cryptology. (qv)

Cryptology: The sum of cryptanalysis (qv) and cryptography. (qv)

DAI-Labor: Abbreviation for *Distributed Artificial Intelligence Laboratory:* Working group at the Technische Universität Berlin, where JIAC (qv) was developed.

DARPA: Abbreviation for *Defense Advanced Research Projects Agency:* An important research sponsor of the United States of America military.

Deciphering: Synonymous but less common word for decryption (qv) as defined in [Int89].

Decryption: The opposite of encryption. (qv)

DER: Abbreviation for *Distinguished Encoding Rules:* A subset of BER, (qv) which gives exactly one way to represent any ASN.1 (qv) value as an octet string, layed down in [ITU97b].

DES: Abbreviation for *Data Encryption Standard:* A specific block cipher (qv) as defined in [oST99].

DF: Abbreviation for *Directory Facilitator:* A standard agent defined by FIPA, (qv) that provides information about the provisioning of services.

DN: Abbreviation for *Distinguished Name:* A set of RDNs (qv) fully identifiying an entity in the context of X.500. [ITU93]

DNS: Abbreviation for *Domain Name Server (or Domain Name System):* The computer facility that translates "human friendly" Internet domain names into machine-readable IP addresses.

Domain Name: The unique identifier associated with each particular computer attached to the Internet. For network communications a DNS (qv) server has to translate such domain names into machine-readable IP addresses.

DSA: Abbreviation for *Digital Signature Algorithm:* The asymmetric (qv) algorithm used in the DSS. (qv)

DSS: Abbreviation for *Digital Signature Standard:* A standard for digital signatures, (qv) defined in [oST00].

E-Business: E-business encompasses all forms of on-line electronic trading, taking in the more narrowly defined concept of B2C (qv) e-commerce, (qv) plus B2B (qv) electronic trading and process integration, as well as the internal use of IP (qv) and related technologies for process integration inside organizations.

E-Commerce: In a strict sense, e-commerce is usually taken to encompass B2C (qv) electronic trading. This distinguishes "e-commerce" from the broader arena of "e-business." (qv)

E-Marketplace: Electronic marketplaces are B2B (qv) online trading forum, often dedicated to e-business (qv) between companies and their customers and suppliers in a particular industry or sector thereof.

ECC: Abbreviation for *Elliptic Curve Cryptography:* Asymmetric (qv) cipher (qv) based on the "hard" problem of discrete ellipsis calculations in an algebra over finite fields. Relatively new branch in cryptography, (qv) its maturity is disputed.

ElGamal: Asymmetric (qv) cipher (qv) based on the "hard" problem of calculating discrete logarithms in a finite field, named after its inventor.

Enciphering: Synonymous but less common word for encryption (qv) as defined in [Int89].

Encryption: Hiding the contents of a communicated message, so that unauthorized (qv) parties, that don't have access to a certain secret, i.e. the key, are not able to semantically understand the contents. The reverse operation of restoring the original contents is called decryption. (qv)

FCAPS: Abbreviation for *Fault, Configuration, Accouting, Performance, Security:* The aspects of system management according to [ITU97a].

FIPA: Abbreviation for *Foundation for Intelligent Physical Agents:* A standardization organization for AOT. (qv)

FIPA-ACL: ACL (qv) defined by FIPA. (qv) The content language is SL0. (qv)

Firewall: Firewalls are network security filters used to protect individual computers and/or computer networks (such as intranets or extranets) from access by unauthorized users.

GPRS: Abbreviation for *General Packet Radio Service:* the technology standard used by mobile phones to provide up to a 56Kbs always-on data connection.

GSM: Abbreviation for *The Global System for Mobile communications:* The technology standard used by mobile phones (including WAP (qv) phones), and that provides digital voice and a low-speed data services.

GSN: Abbreviation for *General Software Note:* Amendment to the Wassenaar Arrangement on Export Controls for Conventional Arms and Dual-Use Goods and Technologies of the European Union pertaining to computer software.

GUI: Abbreviation for *Graphical User Interface:* A GUI is a means of interacting with a computer using a tracking device (such as a mouse) to move a pointer to manipulate control menus or icons that represent computer functions and data.

Hash: Cryptographic checksum of fixed length, that is easily calculated but where it is a "hard" problem to find either the original value or another value with the same checksum.

HMAC: Abbreviation for *Keyed-Hashing for Message Authentication:* A MAC (qv) calculation indexed by a symmetric (qv) key (qv), thus ensuring integrity and authenticity.

Illocutionary Act: Applied in the speech act theory (qv) to the *force* that an expression of some specific form will have when it is uttered.

Initialization Vector: Random data used as feedback input for a mode (qv) of a block cipher (qv) before real feedback from the algorithm is available. Makes codebook attacks much harder by virtually enlarging the key size.

Integrity: Property of a communication, ensuring that no one is able to modify the communicated data without this change being noticed.

Intelligent Home: A house or apartment equipped with computers to assist or control household tasks. For example, a refrigerator might keep an account of its contents and contact suppliers to replenish itself.

Internet: The International Network is an open, loose conglomeration of interconnected (as opposed to directly connected) computer networks based on IP (qv) and associated network communications standards.

IP: Abbreviation for *Internet Protocol:* The open computer network communications standard [Inf81a] upon which the Internet (qv) is based.

ISO: Abbreviation for *Organization for Standardization:* A worldwide federation of national standards bodies from some 140 countries. The mission of ISO is to promote the development of standardization and related activities in the world with a view to facilitating the international exchange of goods and services, and to developing cooperation in the spheres of intellectual, scientific, technological and economic activity.

ITAR: Abbreviation for *International Traffic in Arms Regulation:* United States of America regulation about the export of warfare material, including cryptography, currently managed BXA. (qv)

Itinerary: The route of a migrating (qv) agent. In some texts this is particularly used for a route pre-computed and given to the agent before the first migration.

Java: Java is a programming language and associated execution environment and specifically intended for the development of platform-independent applications.

JCA: Abbreviation for *Java Cryptography Architecture:* Classes of Java (qv) related to cryptography.

JCE: Abbreviation for *Java Cyrptography Extension:* Implementation of the JCA, (qv) distributed separatly from the JDK (qv) due to legal restrictions.

JDK: Abbreviation for *Java Development Kit:* Comprehensive set of Java development tools, freely available from Sun Microsystems.

JESS: Abbreviation for *Java Expert System Shell:* A rule engine and scripting environment written entirely in Sun's Java language. (qv)

JIAC: Abbreviation for *Java-based Intelligent Agent Componentware:* Software-agent toolkit of DAI-Labor, (qv) in its incarnation *JIAC IV,* a specific implementation of the CASA (qv) framework.

Jini: Jini network technology provides an infrastructure for delivering services in a network and for creating spontaneous interaction between programs that use these services regardless of their hardware or software implementation.

JRE: Abbreviation for *Java Runtime Environment:* The native program executing Java programs.

JVM: Abbreviation for *Java Virtual Machine:* A VM (qv) for the Java (qv) language.

KDC: Abbreviation for *Key Distribution Center:* Distributes certificates (qv) issued by a CA. (qv)

Key: Input data for encryption (qv) and decryption (qv) functions. In symmetric (qv) algorithms the key has to be a pre-established shared secret between the communicating partners. In an asymmetric (qv) protocol, the secret key is held by only one party, the public key is, as the name suggests, public.

KIF: Abbreviation for *Knowledge Interchange Format:* Content language used in KQML (qv) speech acts.

KQML: Abbreviation for *Knowledge Query and Manipulation Language:* A part of the "Knowledge Sharing Effort" project of DARPA. It defines syntax and semantics of speech acts for inter-agent communication. [DAR, LF97] The KQML-specification only covers the speech acts and their communicative role, but doesn't define the contents, which is specified in an own language, KIF. (qv)

Locutionary Act: Applied in the speech act theory (qv) to the simple act of saying something.

M-Commerce: Referring to all forms of e-commerce (qv) that take place when a consumer makes an online purchase using a mobile device such as a WAP phone or wireless PDA.

MA: Abbreviation for *Manager Agent:* The system agent constituting and managing an agent place.

MAC: Abbreviation for *Message Authentication Code:* (opposed to *Media Access Control* in network technology) A special application of hashes (qv) to prove the integrity (qv) of a message.

Man-in-the-Middle: An attacker, located between two communicating parties in the data path of a network.

MASIF: Abbreviation for *Mobile Agent System Interoperability Facility:* A set of standards of the OMG (qv) dealing with mobile agents. De facto obsoleted by the FIPA (qv) effort.

Migration: Changing one's location, especially used for the movement of an agent from one agent place to another.

Mobile Computer: A computing device, larger than a PDA or palmtop but smaller and far lighter than a laptop. Mobile computers offer "instant on" computing by having a compact operating system and applications permanently installed on an internal ROM chip.

Mode: The combination of a cipher (qv) and some feedback data to gain more security by interlocking single chunks of data.

MTP: Abbreviation for *Message Transport Protocol:* According to FIPA, (qv) a standardized protocol for message exchange between agents.

Non-repudiation: The impossibility for the signer of a message to later deny, that he signed the document.

NSA: Abbreviation for *National Security Agency:* An agency of the United States of America to produce foreign intelligence. Its very existence was long time denied by the government, hence its abbreviations were often interpreted as *No Such Agency.* The largest known employer of mathematicians and cryptographers in the world.

OMG: Abbreviation for *Object Management Group:* Non-profit organization developing commercially viable and vendor independent specifications for the software industry with several large companies as members.

One-Time Pad: The only provably unbreakable cipher. (qv) Impractible for normal use, because the key (qv) material is of the same size as the plaintext (qv) and can never be re-used. This makes transmitting the key material the same problem as transmitting the intended message, gaining nothing.

Ontology: In the agent domain, an ontology is used to define a common vocabulary between agents. An ontology is a set of definitions for basic expressions in a domain. It comprises of categories, objects, attributes, relations, and constraints.

OOT: Abbreviation for *Object-oriented Technology:* The technology domain of object-oriented software development.

Orange Book: A standard [TCS85] about Trusted Computer System Evaluation Criteria of the US Department of Defense, named after the color of its binder.

OSI: Abbreviation for *Open System Interconnection:* A set of standards published by the ISO. (qv)

Padding: Extension of some data to conform to a given block length, used in block ciphers. (qv)

Passive Attack: The attacker only listens to the attacked data stream.

PDA: Abbreviation for *Personal Digital Assistant:* A pocket-sized computer, typically operated via a touchscreen stylus rather than a keyboard.

Perlocutionary Act: Term applied in the speech act theory (qv) to the *effect* brought about by an utterance in the particular circumstances in which it is uttered.

PGP/1: Abbreviation for *Partial Global Planning:* A co-operation protocol avoiding redundancy.

PGP/2: Abbreviation for *Pretty Good Privacy:* A cryptographic system and software for secured message exchange based on peer-to-peer exchanged certificate instead of hierarchically distributed certificates as is the case with CAs (qv) and KDCs. (qv)

Plaintext: The message to be hid by applying a cipher (qv) function to it. Also the result of correctly appying a deciphering (qv) function to a ciphertext. (qv)

Platform Provider: Offers the execution environment, where instances of other roles meet and communicate.

PKCS: Abbreviation for *Public Key Cryptography Standards:* Set of popular de-facto standards dealing with public key cryptography by the company RSA Security.

Privacy The right of an entity (normally a person), acting in its own behalf, to determine the degree to which it will interact with its environment, including the degree to which the entity is willing to share information about itself with others.

Provider: Offers services.

Proxy: A computer process that relays a protocol between client and server computer systems, by appearing to the client to be the server and appearing to the server to be the client. In more general, a messenger between two parties.

RDN: Abbreviation for *Relative Distinguished Name:* A 2-tupel of attribute and value in the context of X.500, [ITU93] used to construct a DN. (qv)

Reactive Agent Architecture: A common agent model, that constructs agents as entities purely reacting on sensoric input from its environment.

Replay Attack: Re-inserting or re-ordering message blocks, active attack by a man-in-the-middle. (qv)

Retailer: A special case of a provider, (qv) but a retailer makes available services of other providers, potentially by adding value due to intelligent combination of services.

RMI: Abbreviation for *Remote Method Invocation:* The methods of remote Java (qv) objects can be invoked from other Java virtual machines, possibly on different hosts.

Roaming: Synonym for migrating. (qv)

Role: A functional position in an organization or process.

RSA Cipher: Abbreviation for *Rivest-Shamir-Adleman Cipher:* Probably the best known and analyzed asymmetric (qv) cipher (qv), named after its inventors, based on the "hard" problem of factoring large numbers. Widely in use, but incumbered by patent issues until recently.

SANP: Abbreviation for *Speech Act based Negotiation Protocol:* A co-operation protocol for agents, able to cope with conflicts.

SA: Abbreviation for *Security Agent:* Standard agent of the ASITA (qv) architecture that supports security functionality to other agents, like fulfilling the role (qv) of a KDC (qv) in a domain.

SAS: Abbreviation for *Speech Act Security:* Security properties applied to speech acts, (qv) performed on the ISO (qv)/OSI (qv) application layer (layer 7).

Security Manager: Java system object regulating security within a VM. (qv)

Set-top Box: Low-cost devices that connect to a conventional television set and a phone line to permit low-cost Internet access.

Signature: A hash over a document, performed with a secret key, that can be verified with the corresponding public key.

SL0: Abbreviation for *Structured Language 0:* Content language for FIPA-ACL. (qv)

SMS: Abbreviation for *Short Messaging Service:* The technology standard used for text-messaging on mobile phones.

Social Engineering: Attack not on the system but on the people using them, e.g. tricking them into revealing their password.

Speech Act Theory: Describes how words are often used to *do* things, rather than merely to *comment* on a state of affairs in the world. Comprises of a locutionary, (qv) illocutionary, (qv) and perlocutionary (qv) act.

SSA: Abbreviation for *Security Service Agent:* A standard agent that offers value-added security-related services to other agents in the ASITA (qv) framework. Those services are not essential for the secure functioning of an agent place, they are provided by the SA, (qv) instead they aid in the creation of e-business (qv) applications and services.

SSL: Abbreviation for *Secure Socket Layer:* an Internet security standard (cryptography protocol) developed in order to provide a reliable and private connection over the world-wide web and potentially other forms of computer network.

Steganography: This is the science of hiding the very existence of data. This includes hiding large amounts of stored or communicated data in other data. Another aspect of steganography is the concealment of performed communication, or the hiding of communicating partners.

Stream Cipher: Encryption (qv) function, that works on only one bit of data at a time.

Symmetric Cipher: Encryption, (qv) where the key to be used to decrypt the ciphertext is the same as was used for encryption.

TCB: Abbreviation for *Trusted Computing Base:* A small and verifable component of a computer system, that preserves the integrity of sensitivity labels and uses them to enforce a set of mandatory access control rules, defined in the "Orange Book." (qv)

Telematics: Synthesis of telecommunication and information.

Thread: A thread of execution in a program. The Java Virtual Machine (qv) allows an application to have multiple threads of execution running concurrently.

Toolkit: The sum of an architecture, an execution environment, infrastructure services, and tools for the realization and management of applications and services.

Truncation: Arbitrarily reducing the length of an output, often used for hashes. (qv) It is under dispute, if this strengthens the system by making it even harder to guess the hashed data, or if it weakens the system because it is now easier to find collisions. (qv)

Trust: This is a very intuitive and hence debatable property. One interpretation is the extent to which someone who relies on a system can have confidence that the system meets its specifications, i.e., that the system does what it claims to do and does not perform unwanted functions. Another definition is, an entity can be said to 'trust' a second entity when the first entity makes the assumption that the second entity will behave exactly as the first entity expects, this trust may apply only for some specific function.

Ubiqutous Computing: The concept of building computers into our everyday working and living environments to such an extent that data, rich media, and network access become constantly, frictionlessly, and transparently available. Developments in ubiquitous computing – such as multimodality interfaces – involve making technology conform to human requirements.

UMTS: Abbreviation for *Universal Mobile Telephone System:* The technology standard to be used in mobile phones which will provide up to a 384Kbs data connection to the Internet.

URL: Abbreviation for *Uniform Resource Locator:* It is the address of a certain file or directory on the Web. They consist of two main parts. The first part of a URL indicates what protocol is used, the second is an address meaningful in the context of the protocol.

VM: Abbreviation for *Virtual Machine:* Especially used in the Java (qv) context, where it denotes the runtime engine of the architecture, providing a virtual machine with capabilities independent of and abstracted from the host machine it is running on.

WAP: Abbreviation for *Wireless Application Protocol:* A standard for data communications that provides a relatively low speed (typically 9.6Kbs) Internetconnection to a mobile phone via GSM. (qv)

X.500: A standard that defines a global naming scheme.

X.509: A standard defining certificates (qv) to be used in a X.500 (qv) context.

Bibliography

[AAB+98] Hal Abelson, Ross Anderson, Steven M. Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Peter G. Neumann, Ronald L. Rivest, Jeffrey I. Schiller, and Bruce Schneier. The RISKS of Key Recovery, Key Escrow, & Trusted Third Party Encryption. http://www.cdt.org/crypto/risks98/, 1998.

[AF99] Carlisle Adams and Stephen Farrell. Internet X.509 Public Key Infrastructure Certificate Management Protocols. IETF RFC 2510, March 1999. Proposed Standard.

[Alb98] Sahin Albayrak. Introduction to Agent Oriented Technology for Telecommunications. volume 36 of *Frontiers in Artifical Intelligence and Applications*, pages 1–18, Amsterdam, Netherlands, 1998. IOS Press.

[And72] J. P. Anderson. Computer Security Technology Planning Study. Technical Report I, US Air Force, Hanscom AFB, Bedford, Mass., October 1972. ESD-TR-73-51.

[Aud01] Iris Auding. Sorge um Datenmissbrauch bremst E-Commerce. *heise online*, May 2001.

[Aus62] John Langshaw Austin. *How to Do Things with Words*. Oxford University Press, 1962.

[AW99] Sahin Albayrak and Dirk Wieczorek. JIAC — A Toolkit for Telecommunication Application. volume 1699 of *Lecture Notes in Computer Science, Lecture Notes in Artificial Intelligence*, pages 1–18, Berlin, Heidelberg, New York, Barcelona, Hong Kong, London, Milan, Paris, Singapore, Tokyo, 1999.

Springer. Proceedings of third international workshop, Stockholm, Sweden.

[BF95] M. Barbuceanu and M.S. Fox. COOL: A Language for Describing Coordination in Multi Agent Systems. In *Proceedings of the International Conference On Multi-Agent Systems*, pages 14–24, San Francisco, CA, USA, 1995. V. Lesser.

[BFIK99] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote Trust-Management System Version 2. IETF RFC 2704, September 1999.

[BG88] A.H. Bond and L. Gasser, editors. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann Publishers, San Mateo, CA, USA, 1988.

[Bib93] W. Bibel. Wissensrepräsentation und Inferenz, 1993.

[BLFM98] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax. IETF RFC 2396, August 1998.

[Bro86] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 1(2):14–23, 1986.

[Bun95] Deutsche Bundesregierung. Verordnung über die technische Umsetzung von Überwachungsmaßnahmen des Fernmeldeverkehrs in Fernmeldeanlagen, die für den öffentlichen Verkehr bestimmt sind (Fernmeldeverkehr-Überwachungs-Verordnung – FÜV), May 1995. Online at http://www.regtp.de/imperia/md/content/tech_reg_t/ueberwachu/2.pdf.

[Bun96] Deutscher Bundestag. Telekommunikationsgesetz (TKG). BGBl. I S. 1120, July 1996. Geändert durch Art. 2 Abs. 34 des Begleitgesetzes zum Telekommunikationsgesetz vom 17. Dezember 1997 (BGBl. I S. 3108), geändert durch Art. 2 Abs. 6 des Sechsten Gesetzes zur Änderung des Gesetzes gegen Wettbewerbsbeschränkungen vom 26. August 1998 (BGBl. I S. 2544), online at http://www.regtp.de/gesetze/start/fs_04.html.

[Bun97] Deutscher Bundestag. Gesetz zur Regelung der Rahmenbedingungen für Informations- und Kommunikationsdienste (Informations- und Kommunikationsdienste-Gesetz – IuKDG). BT-Drs. 13/7934 of June 11, 1997, June 1997.

[Cas95] C. Castelfranchi. Guarantees for autonomy in cognitive agent architecture. *Lecture Notes in Computer Science*, 890:56–??, 1995.

[cci88a] Recommendation X.208 – Specification of Abstract Syntax Notation One (ASN.1), 1988.

[CCI88b] CCITT. Recommendation X.208: Specification of Abstract Syntax Notation One (ASN.1), 1988.

[CCI88c] CCITT. Recommendation X.209: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1), 1988.

[Cer97] Certicom Corp. An Introduction to Information Security. http://www2.certicom.ca/ecc/wpaper.htm, March 1997.

[Cer99] Certicom Corp. Certicom News. http://www.certicom.com/news.htm, August 1999.

[Cer00] Certicom Corp. Whitepapers. http://www2.certicom.ca/ecc/wpaper.htm, August 2000. Version 4.1.

[CL90] P. R. Cohen and H. J. Levesque. Intention is Choice with Commitment. *Artificial Intelligence*, 42:213–261, 1990.

[CML86] S.E. Conry, R.A. Meyer, and V.R. Lesser. Multistage Negotiation in Distributed Planning. In Bond and Gasser [BG88].

[Com97] European Commission. Communication from the Commission Towards A European Framework for Digital Signatures And Encryption. October 1997. Online at http://www.ispo.cec.be/eif/policy/97503toc.html.

[Cur98] Matt Curtin. Snake Oil Warning Signs: Encryption Software to Avoid. http://www.interhack.net/people/cmcurtin/snake-oil-faq.html, April 1998.

[CW92] M. Chang and C.C. Woo. SANP: A Communication Level Protocol for Negotiations. In Y. Demazeau and J.P. Müller, editors, *Decentralized A. I. – Proceedings of the 3rd European Workshop on Modelling Autonomous Agents in Multi-Agent Worlds (MAAMAW-92)*, pages 31–54, 1992.

[Cyb01] Cyber Business Centre at Nottingham University Business School. Glossary, 2001. Online at http://www.nottingham.ac.uk/cyber/Gloss.html.

[DAR] The DARPA Knowledge Sharing Initiative. *Specification of the KQML Agent-Communication Language*.

[dB94] B. den Boer and A. Bosselaers. Collisions for the compression function of MD5. *Lecture Notes in Computer Science*, 765:293–??, 1994.

[DBP96] H. Dobbertin, A. Bosselaers, and B. Preneel. RIPEMD-160: A Strengthened Version of RIPEMD. *Lecture Notes in Computer Science*, 1039:71–82, 1996.

[DH98] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6). IETF RFC 2460, December 1998.

[Die00a] Dr. Oliver Diedrich. Europäisches Patentamt befürwortet Software-Patente. *heise online*, September 2000.

[Die00b] Dr. Oliver Diedrich. Hyperlink-Patent zeigt Absurdität von Software-Patenten. *heise online*, June 2000.

[Die00c] Dr. Oliver Diedrich. Keine Software-Patente nach amerikanischem Muster in Europa. *heise online*, July 2000.

[DL91] E.H. Durfee and V.R. Lesser. Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1167–1183, 1991.

[DM90] E.H. Durfee and T.A. Montgomery. A Hierarchical Protocol for Coordinating Multiagent Behavior. In *Proceedings of AAAI-90*, pages 86–93, 1990.

[Dob97] H. Dobbertin. RIPEMD with Two-Round Compress Function Is Not Collision-Free. *Journal of Cryptology: The Journal of the International Association for Cryptologic Research*, 10(1):51–69, Winter 1997.

[Dr.01] Dr. Dorothee Dersch and Dr. Torsten Eymann and Stefan Sackmann and Florenza Goanta and Ralf Baumann. Digital Business Agents – Nutzen und Potenziale von Multi-Agenten-Systemen: Grundlagen, Chancen, Geschäftsmodelle. Technical report, Diebold Deutschland GmbH, Frankfurter Straße 27, D 65760 Eschborn, 2001.

[ECS94] Donald E. Eastlake, Stephen D. Crocker, and Jeffrey I. Schiller. Randomness Recommendations for Security. IETF RFC 1750, December 1994.

[EG85] T. El-Gamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE TIT*, IT-31(4):469–472, 1985.

[ELG99] Hans Joachim Einsiedler, Alain Léger, and Marie-Pierre Gleizes. ABROSE: A Co-operative Multi-Agent Based Framework for Electronic Marketplace. Technical report, European Commission, ACTS Programme, CLIMATE cluster, September 1999. Published in "Agents Technologies in Europe – ACTS Activities," online at http://www.infowin.org/ACTS/ANALYSYS/PRODUCTS/THEMATIC/AGENTS/ch3/abrose.htm.

[Ell99a] Carl M. Ellison. SPKI Requirements. IETF RFC 2692, September 1999.

[Ell99b] Carl M. Ellison. The nature of a useable PKI. *Computer Networks*, 31(8):823–830, 1999.

[Ern01] Ernst & Young Deutsche Allgemeine Treuhand AG. Information Security Survey 2001. Technical report, ISAAS – Information Systems Assurance and Advisory Services auf einen Blick, 2001. Online at http://www.ernst-young.de/pdf/Information_Security_Survey.pdf.

[FBK+01] Stefan Fricke, Karsten Bsufka, Jan Keiser, Torge Schmidt, Ralf Sesseler, and Sahin Albayrak. Agent-based telematic ser-

vices and telecom applications. *Communications of the ACM*, 44(4):43–48, April 2001.

[FG96] Stan Franklin and Art Graesser. Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents. In *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*. Springer-Verlag, 1996.

[FKK96] Alan O. Freier, Philip L. Karlton, and Paul C. Kocher. The SSL Protocol. IETF Draft `draft-freier-ssl-version3-02.txt`, November 1996. Version 3.0.

[FN71] R. E. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 5(2):189–208, 1971.

[Fou97] Foundation for Intelligent Physical Agents (FIPA). Fipa 97 specification version 1.2. http://drogo.cselt.stet.it/fipa/spec/fipa97/fipa97.htm, 1997.

[Fou00] Foundation for Intelligent Physical Agents (FIPA). Mission statement. http://www.fipa.org/about/mission.html, 2000.

[Gal88] J.R. Galliers. *A Theoretical Framework for Computer Models of Cooperative Dialogue, Acknowledging Multi-Agent Conflict*. PhD thesis, Open University, UK, 1988.

[GBS00] Bill Joy Gilad Bracha, James Gosling and Guy Steele. *The Java Language Specification*. Java. Addison-Wesley, second edition edition, 2000. Online at http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html.

[Geo00] George Cybenko and Bob Gray and David Kotz and Daniela Rus. D'Agents. http://agent.cs.dartmouth.edu/, 2000.

[GF] Michael R. Genesereth and Richard E. Fikes. *Knowledge Interchange Format*, version 3.0 edition. Reference Manual.

[Gil96] D. Gilbert. Intelligent Agents White Paper. 1996. IBM Intelligent Agent Center of Competency.

[GK94] Michael R. Genesereth and Steven P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53, July 1994.

[Gol96] Dieter Gollmann, editor. *Fast software encryption: third international workshop Cambridge, UK, February 21–23, 1996: proceedings*, volume 1039 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[Gon98] Li Gong. Java Security Architecture (JDK1.2). http://java.sun.com/products/jdk/1.2/docs/guide/security/spec/security-spec.doc.html, October 1998.

[Gue01] Gregory L. Guerin. Microsoft, VeriSign, and Certificate Revocation. http://amug.org/~glguerin/opinion/revocation.html, April 2001. Last revised May 13, 2001.

[GW94] Christine Guilfoyle and Ellie Warner. Intelligent Agents – the new Revolution in Software. Technical report, OVUM, 1994.

[HCK95] Colin G. Harrison, David M. Chess, and Aaron Kershenbaum. Mobile Agents: Are they a good idea? Technical report, IBM, 1995.

[Hei99] Heise Verlag Online. c't CA Policy Zertifizierungsrichtlinien. http://heise.de/ct/pgpCA/policy.shtml, May 1999.

[Hew77] Carl Hewitt. Viewing Control Structures as Patterns of Passing Messages. *Artificial Intelligence*, 8:323–363, 1977.

[HFPS99] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 Public Key Infrastructure, Certificate and CRL Profile. IETF RFC 2459, January 1999.

[Hig00] Daniel Higgs. Network Exploit Present Between Keyboard and Chair. http://www.osopinion.com/Opinions/DanielHiggs/DanielHiggs1.html, 2000.

[IKV] IKV++ GmbH. Grasshopper Basics And Concepts. http://www.grasshopper.de/download/doc/BasicsAndConcepts2.2.pdf.

[Inf81a] Information Sciences Institute. Internet Protocol. IETF RFC 791, September 1981.

[Inf81b] Information Sciences Institute. Transmission Control Protocol. IETF RFC 793, September 1981.

[Ins98] MageLang Institute. Fundamentals of Java Security. http://developer.java.sun.com/developer/onlineTraining/Security/Fundamentals/index.html, October 1998.

[Ins99a] Institute for Applied Information Processing and Communications, Graz University of Technology. IAIK Java Cryptography Extension. http://jcewww.iaik.tu-graz.ac.at/jce/jce.htm, July 1999.

[Ins99b] Institute for Applied Information Processing and Communications, Graz University of Technology. IAIK–iSaSiLk. http://jcewww.iaik.tu-graz.ac.at/iSaSiLk/iSaSiLk.htm, July 1999.

[Int89] International Organization for Standardization. Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 2: Security Architecture, June 1989.

[ist99] Information society technologies: 1999 workprogramme. Technical report, European Commission, 1999. Online at http://www.cordis.lu/ist/home.html.

[ITU93] ITU-T Telecommunication Standardization Sector of ITU. Information Technology – Open Systems Interconnection – The Directory: Models, November 1993. ITU-T Recommendation X.501, ISO/IEC International Standard 9594-2.

[ITU97a] ITU-T Telecommunication Standardization Sector of ITU. Information technology – Open Systems Interconnection – Systems management overview, 1997. ITU-T Recommendation X.701, ISO/IEC International Standard 10040.

[ITU97b] ITU-T Telecommunication Standardization Sector of ITU. Information Technology – Open Systems Interconnection – The Directory: Authentication Framework, 1997. ITU-T Recommendation X.509, ISO/IEC International Standard 9594-8.

[J.E94] J.E. White. Telescript technology: The foundation for the electronic marketplace, 1994. Version 4.1.

[Jos95] Joseph A. Bank. Java Security. http://www-swiss.ai.mit.edu/~jbank/javapaper/javapaper.html, December 1995.

[KA98] Stephen Kent and Randall Atkinson. Security Architecture for the Internet Protocol. IETF RFC 2401, November 1998.

[Kar] Neeran Karnik. *Security in Mobile Agent Systems*. PhD thesis. Ajanta agent system, online at http://www.cs.umn.edu/Ajanta/papers/nmk_thesis.ps.

[KBC97] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed–Hashing for Message Authentication. IETF RFC 2104, February 1997.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. 1999.

[Koc95] Paul Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. December 1995.

[Koo00] Bert-Jaap Koops. Crypto Law Survey. August 2000. Version 18.1.

[LF94] Yannis Labrou and Tim Finin. A semantics approach for KQML – a general purpose communication language for software agents, 1994.

[LF97] Yannis Labrou and Tim Finin. A Proposal for a new KQML Specification, 1997.

[LY99] Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification*. Java. Addison-Wesley, second edition edition, 1999. Online at http://java.sun.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html.

[Mag] General Magic. Web home page. http://www.generalmagic.com/.

[MAM+99] Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Carlisle Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP. IETF RFC 2560, June 1999. Proposed Standard.

[Mat97] P.H. Matthews, editor. *The Concise Oxford Dictionary of Linguistics*. 1997.

[McL93] Kenneth McLeish, editor. *Bloomsbury Guide to Human Thought*. 1993.

[Met99] Riku Mettala. Bluetooth Protocol Architecture. http://www.bluetooth.com/developer/whitepaper/whitepaper.asp, August 1999. Bluetooth White Paper, Version 1.0.

[Möc99] Frank Möcke. EU-Verbraucherschutz bedroht den Mittelstand. *heise online*, August 1999.

[Möl97] Ulf Möller. Kryptographie: Rechtliche Situation, politische Diskussion. http://www.fitug.de/ulf/krypto/verbot.html, August 1997.

[MSST98] Douglas Maughan, Mark Schneider, Mark Schertler, and Jeff Turner. Internet Security Association and Key Management Protocol (ISAKMP). IETF RFC 2408, November 1998.

[NCS] Rainbow Series Library. Technical report. Online at http://www.radium.ncsc.mil/tpep/library/rainbow/5200.28-STD.pdf.

[NCS00] Evaluated Products List. Technical report, May 2000. Online at http://www.radium.ncsc.mil/tpep/epl/index.html.

[Net99] Network Associates Inc. *An Introduction to Cryptography*. 3965 Freedom Circle, Santa Clara, CA 95054, USA, PGP Version 6.5.2 edition, October 1999.

[Nor00] Nortel Networks Corporation. FIPA-OS V2.0.0 Distribution Notes. http://prdownloads.sourceforge.net/fipa-os/FIPA_OSv2_0_0.pdf, 2000.

[Obj00] ObjectSpace. Voyager Security Developer's Guide. http://support.objectspace.com/doc/Security/index.htm, 2000. Version 4.0.

[Org99] European Patent Organization. Promoting innovation through patents – Green Paper on the Community patent and the patent system in Europe. February 1999. Online at http://http://europa.eu.int/comm/internal_market/en/intprop/indprop/paten.pdf.

[oST95] National Institute of Standards and Technology. Secure Hash Standard, FIPS PUB 180-1, April 1995.

[oST96] National Institute of Standards and Technology. Electronic Data Interchange (EDI), FIPS PUB 161-2, April 1996.

[oST99] National Institute of Standards and Technology. Data Encryption Standard (DES), FIPS PUB 46-3, October 1999. TripleDES also published as ANSI X9.52.

[oST00] National Institute of Standards and Technology. Digital Signature Standard (DSS), FIPS PUB 186-2, January 2000.

[otA90] Headquarters Department of the Army. *Basic Cryptanalysis, Field Manual No. 34-40-2*. US Army, Washington, DC, September 1990.

[otOoDTC99] Director of the Office of Defense Trade Controls. International Traffic in Arms Regulations (ITAR), April 1999. Put online by Society for International Affairs at http://www.siaed.org/itar/itar120.html.

[plc99] British Telecommunications plc. Zeus web home page. http://www.labs.bt.com/projects/agents/zeus/index.htm, 1999.

[Pro01] The FreeBSD Documentation Project. Freebsd handbook. http://www.freebsd.org/doc/en_US.ISO_8859-1/books/handbook/index.html, 2001.

[Reg97] Regulierungsbehörde für Telekommunikation und Post. Entwurf Maßnahmenkatalog für digitale Signaturen – auf Grundlage von SigG und SigV, November 1997. Version 1.0.

[Reg00] Regulierungsbehörde für Telekommunikation und Post. Fragen und Antworten. http://www.regtp.de/tech_reg_tele/start/in_06-02-03-00-00_m/index.html, September 2000.

[Rei96] Deutscher Reichstag. Bürgerliches Gesetzbuch. RGBl. pg. 195, August 1896. Also published in BGBl. III 400-2.

[RG85] J.S. Rosenschein and M.R. Genesereth. Deals among rational agents. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, pages 91–99, Los Angeles, CA, USA, 1985.

[Riv92] Ronald L. Rivest. The MD5 Message-Digest Algorithm. IETF RFC 1321, April 1992.

[RN95] Stuart Russel and Peter Norwig. *Artificial Intelligence, A Modern Approach*. Prentice-Hall, 1995.

[RSA93a] RSA Laboratories. PKCS#1: RSA Encryption Standard. ftp://ftp.rsa.com/pub/pkcs/ps/pkcs-1.ps, November 1993. Version 1.5, also RFC 2437.

[RSA93b] RSA Laboratories. PKCS#10: Certification Request Syntax Standard. ftp://ftp.rsa.com/pub/pkcs/ps/pkcs-10.ps, November 1993. Version 1.0, also RFC 2314.

[RSA93c] RSA Laboratories. PKCS#3: Diffie-Hellman Key-Agreement Standard. ftp://ftp.rsa.com/pub/pkcs/ps/pkcs-3.ps, November 1993. Version 1.4.

[RSA93d] RSA Laboratories. PKCS#5: Password-Based Encryption Standard. ftp://ftp.rsa.com/pub/pkcs/ps/pkcs-5.ps, November 1993. Version 1.5.

[RSA93e] RSA Laboratories. PKCS#7: Cryptographic Message Syntax Standard. ftp://ftp.rsa.com/pub/pkcs/ps/pkcs-7.ps, November 1993. Version 1.5, also RFC 2315.

[RSA93f] RSA Laboratories. PKCS#8: Private-Key Information Syntax Standard. ftp://ftp.rsa.com/pub/pkcs/ps/pkcs-8.ps, November 1993. Version 1.1.

[RSA97] RSA Laboratories. PKCS#12: Personal Information Exchange Syntax Standard. ftp://ftp.rsa.com/pub/pkcs/pkcs-12/PKCS12.PDF, April 1997. Version 1.0.

[RSA00] RSA Laboratories. FAQ about Today's Cryptography. http://www.rsasecurity.com/rsalabs/faq/, August 2000. Version 4.1.

[San01] Sandia National Laboratories. the Java Expert System Shell. http://herzberg.ca.sandia.gov/jess/, May 2001.

[Sar89] Desmond J. Sargent. Information handling system and terminal apparatus therefore, October 1989. Filed as US Patent 4,873,662.

[Sch96] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, Inc., 2. edition, 1996.

[Sch97] Peter Schefe. Eine kleine Sicherheitsphilosophie der Softwaretechnik. *Telepolis*, December 1997.

[Sch98] Bruce Schneier. The Fallacy of Cracking Contests. *Crypto-Gram*, December 1998.

[Sch99a] Bruce Schneier. Cryptography: The Importance of Not Being Different. *Crypto-Gram*, April 1999.

[Sch99b] Bruce Schneier. Snake Oil. *Crypto-Gram*, February 1999.

[Sea69] John R. Searle. *Speech Acts*. Cambridge University Press, 1969.

[Sec00] SecurityFocus.com. Security Focus.com News. http://www.securityfocus.com/, September 2000.

[Ses02] Ralf Sesseler. *Eine Architektur zur Interaktion von Agenten basierend auf Diensten*. PhD thesis, Technische Universität Berlin, 2002.

[Sha63] Claude E. Shannon. The Mathematical Theory of Communication. 1963. Originally from Bell System Technical Journal, July and October 1948.

[Shi00] R. Shirey. Internet Security Glossary. IETF RFC 2828, May 2000. Informational.

[Sho93a] Yoav Shoham. Agent-Oriented Programming. *Artifical Intelligence*, 60:51–92, 1993.

[Sho93b] Yoav Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, March 1993.

[Smi80] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 1980.

[SN95] Paul A. Samuelson and William D. Nordhaus. *Economics*. McGraw-Hill Higher Education, 15th edition, 1995. International Edition.

[Spa95] Eugene H. Spafford. A Few Comments on "Hacker Challenges". In Carl Landwehr, Hilarie Orman, editor, *Electronic CIPHER*, number 12. IEEE Computer Society's TC on Security and Privacy, February 1995. Online at http://www.itd.nrl.navy.mil/ITD/5540/ieee/cipher/old-issues/issue9602.

[SS75] J.H. Saltzer and M.D. Schroeder. The Protection of Information in Computer Systems. In *Proceedings of the IEEE*, volume 63, pages 1278–1308, September 1975.

[ST98] T. Sander and C. Tschudin. Towards mobile cryptography. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 215–224, Oakland, CA, USA, May 1998. IEEE Computer Society Press.

[Sun98a] Sun Microsystems, Inc. Permissions in JDK1.2. http://java.sun.com/products/jdk/1.2/docs/guide/security/permissions.html, October 1998.

[Sun98b] Sun Microsystems, Inc. Security Managers and JDK 1.2. http://java.sun.com/products/jdk/1.2/docs/guide/security/smPortGuide.html, October 1998.

[Sun99a] Sun Microsystems, Inc. Jini Technology Architectural Overview. http://www.sun.com/jini/whitepapers/architecture.html, January 1999.

[Sun99b] Sun Microsystems, Inc. What is the Java Platform? http://java.sun.com/nav/whatis/, October 1999.

[Sun00a] Sun Microsystems, Inc. JAVA 2 SDK, Standard Edition, Version 1.2. http://java.sun.com/products/jdk/1.2/, August 2000.

[Sun00b] Sun Microsystems, Inc. JAVA 2 SDK, Standard Edition, Version 1.4. http://java.sun.com/j2se/1.4/, August 2000.

[Sun00c] Sun Microsystems, Inc. Java Cryptography Extension (JCE) 1.2.1. http://java.sun.com/products/jce/, September 2000.

[Sun01] Sun Microsystems, Inc. JavaTM Secure Socket Extension (JSSE) 1.0.2. http://java.sun.com/products/jsse/, 2001.

[Syc88] K. Sycara. Resolving Goal-Conflicts via Negotiation. In *Proceedings of AAAI-88*, pages 245–250, 1988.

[T-T01] T-TeleSec. Richtlinien für die Vergabe von ServerPass Zertifikaten. https://wwwca.telesec.de/Pub_Cert/ServPass/cps/index.html, May 2001.

[TCS85] Department of Defense Trusted Computer System Evaluation Criteria. Technical report, Fort Meade, MD 20755-6000, USA, December 1985. DoD 5200.28-STD, aka "Orange Book," online at http://www.radium.ncsc.mil/tpep/library/rainbow/5200.28-STD.pdf.

[Töb99] Hermann Többen. *Konzeption eines marktorientierten Routingverfahrens für ATM-Netze auf der Basis Intelligenter Agenten*. PhD thesis, Technische Universität Berlin, dec 1999.

[Tok00] Tokyo Research Laboratory, IBM Japan, Ltd. Aglets Software Development Kit. http://www.trl.ibm.com/aglets/, June 2000.

[Var00] Various. Security Focus.com Bugtraq Mailing List Archive. http://www.securityfocus.com/bugtraq/archive/, September 2000.

[Ver97] VeriSign Inc. Verisign Certification Practice Statement. https://www.verisign.com/repository/CPS/, May 1997.

[vNM44] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[Wei] Lauren Weinstein. The PRIVACY Forum. http://www.vortex.com/privacy/.

[WJ95] Michael Wooldridge and Nicholas R. Jennings. Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995. Online at http://www.elec.qmw.ac.uk/dai/pubs/KER95/.

[WJ98] Michael Wooldridge and Nicholas R. Jennings. Pitfalls of Agent-Oriented Development. In *Proceedings of 2nd International Conference on Autonomous Agents (Agents-98)*, pages 385–391, 1998.

[WM-01] WM-Net. Web developers online glossary, 2001. Online at http://wm-net.com/glossary/.

[Yah01] Yahoo. Microsoft acknowledges secret code in software. http://smallbusiness.yahoo.com/entrepreneur.html?s=smallbiz/articles/20010514/microsoft_ackno, May 2001.