Robust and Private Computations of Mobile Agent Alliances

Zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

von der Fakultät für Informatik der Universität Fridericiana zu Karlsruhe (TH)

genehmigte

Dissertation

von

Regine Endsuleit

aus Freiburg i.Br.

Tag der mündlichen Prüfung:	11. Mai 2007
Erster Gutachter:	Prof. Dr. Jacques Calmet
Zweiter Gutachter:	Prof. Dr. Andrea Omicini

Für meinen Mann Ralf und die süßen Mäuse Marc und Emily

Deutsche Zusammenfassung

Das Paradigma der mobilen Software-Agenten hat sich nach wie vor in der praktischen Anwendung nicht in aller Konsequenz durchgesetzt. Dies liegt vor allem an der Schwierigkeit, einzelne autonome Agenten vor böswilligen Gastrechnern zu schützen. Man muß sich nicht nur mit einer möglichen Korruption von Code und Daten des Agenten befassen, sondern auch damit, daß er nicht korrekt ausgeführt, fehlgeleitet, ausspioniert oder gar gelöscht wird. Diese Arbeit präsentiert ein Sicherheitsmodell basierend auf Agentenallianzen, welche eine robuste und geheime Funktionsauswertung in einer nicht vertrauenswürdigen Umgebung wie dem Internet ermöglichen.

Der Schutz von mobilen Agenten ist ein sehr reges und anspruchsvolles Forschungsgebiet. Die meisten Publikationen bieten kryptographische Methoden an, mit denen gezielte Angriffe verhindert bzw. massiv erschwert werden (beispielsweise durch Verschlüsselung von Code und Daten). Wegen der Vielfältigkeit der Gefahren hat dies zum einen zur Folge, daß die meisten Ansätze sich lediglich mit einem Angriffstyp befassen können; zum anderen ist ein starker kryptographischer Schutz in der Regel mit einer zu hohen Komplexität verbunden. Auch gibt es bisher kein praktisch einsetzbares Sicherheitsmodell, welches ohne Verwendung einer vertrauenswürdigen Instanz beweisbare Sicherheit gegen die Gesamtheit der genannten Bedrohungen liefert.

Die vorliegende Arbeit verfolgt einen völlig anderen Ansatz. Inspiriert durch kryptographische Protokolle für sichere Mehrparteienberechnungen, werden Allianzen von Agenten erzeugt, die verteilte und fehltertolerante Berechnungen durchführen. Die Korrektheit dieser Berechnungen ist garantiert, solange ein aktiver Angreifer nicht mehr als t Agenten gleichzeitig korrumpiert. Der Wert t ist protokollabhängig und liegt für Allianzen, die aus n Agenten bestehen, zwischen $\lceil \frac{1}{2}n \rceil - 1$ und $\lceil \frac{1}{3}n \rceil - 1$. Die Eingaben der einzelnen Agenten für die gemeinsam zu berechnende Funktion werden mittels eines (t, n)-*Verifiable Secret-Sharing* Schemas in Form sogenannter t-Shares auf alle Agenten verteilt. Dies impliziert zum einen, daß nur bei Kooperation von mehr als t Agenten Information über die verteilten Daten bekannt wird, und zum anderen, daß die Integrität der Shares geprüft werden kann. Die eigentliche Funktionsberechnung entspricht der Auswertung eines arithmetischen Schaltkreises, wobei Additionen von jedem Agenten lokal und ohne Kommunikation berechnet werden können. Für jedes Multiplikationsgatter hingegen muß die Allianz kommunizieren. Dies bildet den beschränkenden Faktor des vorgestellten Modells für den Einsatz in realen Netzwerken.

Da Protokolle für sichere Mehrparteienberechnungen n feste Parteien vorsehen, muß für mobile Agenten das Problem der Migration gehandhabt werden. Bereits in einem statischen Szenario ist es unabdingbar, daß alle Mitglieder einer

Allianz auf unterschiedlichen Hosts ausgeführt werden. Andernfalls könnte eine Allianz kompromittiert werden, obwohl weniger als t Server maliziös sind. Dieses Sicherheitsproblem verschärft sich in einem dynamischen Szenario, in welchem die Agenten von Host zu Host migrieren können und dabei ihren Berechnungszustand mitführen. Ein Angreifer könnte durch Ausführung mehrerer Agenten zu unterschiedlichen Zeitpunkten ausreichend viel Information erlangen, um den Berechnungszustand aus den gesammelten t-Shares zu rekonsturieren. Folglich ist es notwendig, außerhalb der anwendungsspezifischen Berechnungen (der Auswertung des Schaltkreises) weitere verteilte Protokolle zur Verfügung zu stellen, welche eine sichere Migration erlauben. In der vorliegenden Dissertation wird eine Lösung aufgezeigt, die auf Resharing-Protokollen, Zertifikaten und Mehrheitsentscheiden beruht.

Ein randomisierte Resharing erlaubt zum Zeitpunkt der Migration eine Neuverteilung des Berechnungszustandes einer Allianz, indem mithilfe der aktuellen t-Shares neue t-Shares für die neuen Hosts erzeugt werden. Das Verfahren stellt sicher, daß die alten Shares schließlich nutzlos werden. Mit dieser Technik kann also der Berechnungszustand sicher von einer Menge von Hosts auf eine andere übertragen werden.

Für statische Daten, wie beispielsweise den Schaltkreis, reicht eine digitale Signatur des Allianzenerzeugers aus, um Hosts von der Integrität der Daten überzeugen zu können. Problematischer ist dynamisches Wissen, wie zum Beispiel eine Liste der aktuellen Hosts der Allianz, welches nicht verteilt aufbewahrt wird, aber auch nicht signiert werden kann. Das Modell für sichere Multiagentenberechnungen sieht aus Effizienzgründen für diese Art von Daten Mehrheitsentscheide vor. Sowohl Hosts als auch die Allianzen bedienen sich dieses Mittels.

Neben dem theoretischen Rahmenwerk für Allianzen schlägt die Dissertation eine konkrete Implementierung vor, welche eine quadratische Kommunikationskomplität erreicht. Dieses Ergebnis ist optimal für den Fall, daß kein Broadcast-Kanal zur Verfügung steht. Dies ist in realen Netzen, wie dem Internet, meist der Fall. Der Einsatz einer Broadcast-Simulation ist deswegen notwendig, welche die Ursache für die quadratische Komplexität des Gesamtprotokolls ist.

Für den Implementierungsvorschlag werden verschiedene Anwendungen vorgestellt, deren Komplexität analysiert wird. So zeigt sich, daß das Modell zwar derzeit für beliebige Anwendungen zu aufwendig, aber für eine sichere Implementierung verschiedener kryptographischer Primitive durchaus praktikabel ist. So zum Beispiel für eine verteilte, fehlertolerante Signatur. Mit steigender Bandbreite wird sich die Praktikabilität des Modells in Zukunft erhöhen.

Dank der Natur der verteilten Berechnungen in dem vorgestellten Modell, können korrumpierte Agenten entdeckt, eliminiert und ersetzt werden, ohne daß Kommunikation mit einer vertrauenswürdigen Instanz notwendig wird. Werden zu keinem Zeitpunkt mehr als t Agenten gleichzeitig korrumpiert, so beeinflussen sämtliche der eingangs genannten Angriffe die Korrektheit und Geheimhaltung der Berechnungen nicht. Somit liegt nun endlich ein theoretisches Rahmenwerk vor, welches den Einsatz von mobilen Agenten für sensitive Anwendungen in offenen und nicht vertrauenswürdigen Netzen gestattet.

Contents

Acknowledgement

I am much obliged to my supervisor, Prof. Dr. Jacques Calmet, for his support, trust and humanity. He was always available when needed and at the same time he gave me full freedom for my research. But not only on this professional level he earns my gratitude. I consider it a remarkable act that he made it possible for me to continue my research without interrupt when I delivered my two children Marc and Emily. Both of them were allowed to stay in my office for several months. Without this obligingness my thesis would not have been completed.

Prof. Dr. Andrea Omicini from the University of Bologna, Italy, earns my gratitude as he agreed on being my referee before we even met. He did this job very professionally and reliably although he was not familiar with the universitary system in Karlsruhe. From the personal point of view, I also owe him a lot of thanks as he never complained about the recurrent delays concerning the completion date of my thesis which were mainly caused by the time I had to spend on futile attempts to organise a suitable child care for my little ones.

For his LaTeX support, I am much obliged to Dr. Markus Grassl. During the last years, he spent several hours helping me to fight against this great but also complicated system.

My warm thanks go to my friends Roland Weber, Holger Hellmuth and Britta Goebel. Roland not only read my thesis and made numerous valuable corrections but he also supported me mentally in times of crisis. Holger cared for the numerous problems I had with my computer(s) which – as usually – always mistimed their appearance and most oftenly seemed to lack any logical explanation. Britta always finds the time to drink some coffee or to go to lunch with me. Thus, she successfully distracts me from everyday problems. Also her pragmatism often helped me to put things in perspective.

Last but not least, I am deeply indebted to my husband Ralf Eberhardt. I cannot count the weekends and evenings he had to spend with the children, making it possible for me to work. Without his help, understanding and patience it would not have been possible to balance job and children.

Introduction

Mobile Agents – A Question of Trust

Mobile Agents – What is missing? The title is from [RHR97] and it implies the quintessence of the apparent inconsistency dominating the field of mobile agents. There is a lively interest in the use of mobile agents all over the world. From day to day, researchers at universities and large companies are developing new platforms and agent models. Research conferences are interested in and often enough devoted exclusively to mobile agent technology. Mobile agents have a high potential, and although one is still missing a so-called *killer application* they deserve this attention because of their potential usage for a almost infinite number of different applications. Their prime advantages can be classified into software distribution on-demand, asynchronous [Cor] and autonomous [Wei00] execution of tasks, reduction of communication costs [SS97, CPV97, CK97] and scalability due to dynamic placement of functions.
Why then do real-world systems not yet tap into the full potential of the mobile agent paradigm? When searching for commercial application domains of agents, one rapidly recognises that they mostly eke out a miserable existence. Besides, one can find agents in various closed systems in which the components have detailed knowledge about each other and thus may trust each other. For instance, the so-called *secured domains* of a corporate network fulfil this condition. Within such a domain an agent could migrate from host to host, but the responsibility of an agent is generally restricted to a specific monitoring task, and it simply communicates a system behaviour that deviates from predefined rules to the responsible instance. Normally, important decisions are made by a human being.[1] Current agents have only a restricted functionality, decision mechanism and communication capabilities. Still, mobile agents roaming the Internet in their user's name, concluding contracts and making autonomous decisions seem to be out of reach.
What are the reasons for this gap between research goals and results, between the desires of users and the quite demoralising reality? One could argue about missing artificial intelligence or about the difficult handling of heterogeneous operating systems, databases and networks. But in fact, these problems are secondary. They make the implementation of agent platforms more challenging and complex, but solutions are available. The most important reasons for the mobile agent paradigm's failure in implementation are that people do not trust this technology far enough to delegate important and security sensitive tasks like signing a contract to a mobile agent and that the legal conditions are often miss-

[1] This is also caused by legal restrictions

ing. Both are connected to each other as legislation will provide the legal foundation for this technology as soon as it becomes widely accepted. Trust between human beings, between humans and computers, and even between computers is a building block for any interaction. Human trust significantly influences the trade potential of a company or, on the private level, the possibilities to gain and keep friends. Consequently, trust is one of the most important components that rules our daily live. We buy our food at a grocery store where we trust its owner to act in our interest, only selling food of a certain quality. When making a decision for an Internet provider we choose the one we trust that it is reliable, available, secure and well-priced.

But where does this trust come from? How do we determine the level of trust we have in someone or something? Trust is usually measured by statistical values. In human interaction these values are implicitly derived from experience of life allowing to judge the behaviour of people. If another person never betrayed us, we feel secure in trusting this person. Whereas a treachery decreases the level of trust. Trust between systems or between humans and systems is achieved similarly by statistical evaluations of longterm observations. Factors that are taken into account are for example downtime, response time, reliability, availability and correctness. Not knowing the counterpart of the interaction makes it more difficult for humans to determine the level of trust. On the other hand, in contrast to a purely human interaction, cryptographic methods can be used to increase the trust people have in a system. These methods may detect deviations from given rules or protocols and thus discourage server operators from misbehaving. Although there is only one cryptographic primitive that has been proven to be secure, namely the one-time pad, there are many methods that make a betrayal computationally infeasible. Consequently, integrating cryptographic security into a system generally creates a high level of human trust in this system.[2]

If we return to mobile agents that are supposed to act autonomously in their users name, it is obvious that a very high level of trust is required. How could be as "insane" as to send an agent to an unknown host, where it is executed in an uncontrollable manner, possibly spied out or even manipulated, and finally trust the results the agent returns? Without satisfactory security measures the answer to this question must be "nobody". This is only one part of the interaction. A similar question must be asked for the server executing an agent. Why should someone allow an unknown program to enter a computer and to be executed? In times of viruses, trojans and worms, this seems to be crazy behaviour.

In order to make mobile agents practical, agent platforms have to provide a satisfactory level of security for both, agents and hosts. While there are solutions that allow to protect hosts from malicious code, the protection of mobile agents from malicious hosts is disproportionately more difficult. The server has full control over the agents it is supposed to execute. Consequently, besides a short discussion of host security, this thesis focusses on the presentation of an agent security model.

The following sections provide a brief overview on the security concerns arising in multi-agent systems followed by a glimpse of the security model that is presented later.

[2]This statement is not really correct as most people cannot understand and thus evaluate the algorithms. Cryptography helps experts to trust a system, and we can hope to have a certain transitivity by the trust people have in experts.

Host and Agent Security

Mobile agent platforms in open networks cause threats not only for the participating servers but also for the agents themselves. In order to protect hosts participating in a multi-agent system, one has to face primarily one attack: The agent can carry malicious code, for example a virus or worm that could destroy data or operating system components. The host could even be used to perform Denial-of-Service attacks on other hosts of the agent platform or on arbitrary servers in the Internet.
To prevent the hosted agent from behaving maliciously, sandboxes can be used to restrict the agents rights to access system resources, for example in Java[3] [Jav06]. Besides, host security can be improved by a correct setup of operating systems and regular software fixes. This means that the most restrictive installation should be chosen. Users should get exactly those system and access rights they need to fulfil their tasks and nothing beyond. Since host security mainly depends on secure software and operating system design, this topic is mainly the responsibility of the software producing companies and is subject to their commercial interests.
The protection of mobile agents seems more challenging as there are various different attacks. These attacks can be categorised as follows:

- hurting the execution integrity
- malicious routing
- code and data manipulation
- code and data spying

The question, how one could secure a mobile agent from a malicious environment has lead to an almost unmanageable number of approaches covering the different security issues. Unfortunately, most approaches only handle one security aspect, as for example the detection or prevention of malicious routing. Lots of them use cryptographic methods like digital signatures or authentication, some have a rather statistical nature like those following the code obfuscation approach in order to delayed specific attacks beyond a critical time interval. An effective protection measure is a trusted third party that regularly communicates with the agent and checks its integrity. But this kind of protection conflicts with the autonomy of agents and thus is not fully compliant with the agent paradigm. Furthermore, such an instance is a single point of failure and as such a popular target for Denial-of-Service (DoS) attacks. It is obvious that it is very difficult if not impossible to secure a single autonomous mobile agent since the executing host has full control over code and data. Therefore, a few publications consider co-operating agents that control each other from time to time. This is a very promising approach but until now there is no model for co-operating agents that can provide security guarantees in a larger extend without the use of a trusted third party. This thesis fills this gap by presenting a framework that allows co-operating agents to robustly compute a user-defined function on private inputs in open and untrusted networks like the Internet.

[3]Most platforms that are currently in use are implemented in Java.

Secure Multi-Agent Computations

The proposed model is based on cryptographic protocols for secure multi-party computation (SMPC). The nature of these protocols perfectly fits the needs of mobile agents that are supposed to live in open and unauthenticated networks like the Internet: they allow robust and private computation of a predefined function. Defining so-called *secure multi-agent computation* (SMAC) my aid of SMPC allows an *Alliance* of n co-operating agents to guarantee the correct and private computation of a user-defined functionality as long as no more than a (protocol specific) upper limit t of agents is located on malicious hosts at the same time. The part of the Alliance knowledge that is supposed to be kept secret, for example a private key, is coded into redundant shares by using a (t, n)-secret sharing scheme [Sha79]. Computations on these shares are defined over an arithmetic circuit and are performed in a robust way by distributed computations of the whole Alliance. Maliciously influencing the common computations is only successful if more than t corrupted Alliance members co-operate. Analogously, a reconstruction of shared data is only possible if at least $t+1$ shares are known.[4] The security guarantees that are given by SMPC are quite impressive and solve most of the security problems arising in multi-agent systems. Unfortunately, SMPCs requires a *fixed* group of n parties. This assumption cannot be kept in a mobile agent scenario. There is an undefined state during the migration process of one agent, when the old host as well as the new one are involved in a common transaction, and thus $n + 1$ parties exist. Consequently, one has to fill the gap arising from the transition from the static scenario, where all n agents are constantly hosted on different servers, to the dynamic scenario which allows migration without losing the security properties. Basically, there are two problems that need to be addressed:

1. The necessity of suitable methods that allow a secure migration.
2. The collection of shares over the lifetime of an Alliance has to be prevented in order to avoid mobile adversaries (see chapter 4, page 53).

The first problem is solved by using majority decisions. This mechanism allows an Alliance to perform all computations necessary for the actual migration robustly, but not privately. The lack of privacy for those computations is not security relevant since no secret data is reconstructed and the results (like the destination host) have to be public anyway. The second threat is addressed by applying a suitable re-sharing method after each migration. Thus, all old shares are disabled. A positive side-effect of this method is, that it also allows to exclude agents from the common computation that have been detected as corrupted. Instead, a new agent can be created via distributed computations of the Alliance. Consequently, an Alliance has self-healing capabilities that significantly reduce the probability of an Alliance to get corrupted while performing its task.
Agent Alliances do not aim at preventing from attacks. Instead, the main intention is the robustness and privacy which allows to provide solutions to all The computations are performed in a way that guarantees that

- corrupted agents can be localised and eliminated,

[4] The exact number depends on the number of actually manipulated shares. In the worst case, $3t + 1$ shares are necessary to perform the necessary fault correction.

- malicious input of up to t corrupted agents is irrelevant for the correctness of the common computation,
- there is a certain degree of robustness against Denial-of-Service (DoS) attacks, and
- shared data remains confidential at any time.

There is a variety of protocols for secure multi-party computation. But the field is still challenging and the complexity of most protocols leads to their impracticality. The thesis will shortly discuss some exemplary protocols. The Alliance model currently uses the synchronous model of Hirt and Maurer [HM01], because it is the most efficient one[5] that has been published until now. Therefore, the discussion will focus on a security model based on this protocol.
Until today, agent Alliances are the first model that provides such a high level of security without requiring a trusted third party. Other models aiming on code privacy [ST98c] through function hiding are defined for few function classes only and thus drastically restrict the potential user-defined functionalities. The Alliance model is based on arithmetic circuits. It has been implicitly shown in [Gál95] that for practical implementations, where only gates with finite fan-in are required, these circuits have the same computational power as Boolean circuits. Unfortunately, computations on such circuits cause a high communication complexity when conventional programming languages are simulated. Nonetheless, the Alliance model is the first formal model that demonstrates that mobile agents *can* be secured. This represents an outstanding result as a lot of researchers did challenge this. The complexity of the model may still be a bit too high to use Alliances in a broader range. Nevertheless, for sensitive applications the quadratic complexity is acceptable, and increasing computational power and Internet bandwidth will make the model even more practical in the future.

A Word on Concurrency

Mostly, a server that is offering an agent platform will execute several agents at the same time. This is the same with agent Alliances. Thus, the protocol for SMAC is executed concurrently. From other cryptographic protocols, i.e. zero-knowledge protocols, it is known that the concurrent execution of several instances of the same protocol can cause proven secure protocols to become insecure. Sometimes this is caused by gaining secrets through a clever combination of information flowing between the parties participating in each protocol instance. In other cases, a co-operation of malicious parties from each protocol instance is necessary.
Therefore, the aspect of concurrency should not be disregarded when trying to build an agent model with proven security. This thesis will therefore offer a discussion of the exemplary security model for the concurrent execution of cryptographic multi-party protocols from [Can00] in Appendix B. It is known that the SMPC protocol of Hirt and Maurer is secure in the framework. For the SMAC protocol such a proof is not given, as this is beyond the scope of this thesis. It seems probable, that the SMAC protocol, too, is secure in this

[5] $O(m \cdot n^2)$ communication and computation costs, where m is the number of multiplication gates and n the number of parties.

framework as the additional sub-protocols have on influence on the correctness of the distributed computations. They are meant for migration only.

Structure of this Thesis

Chapter 1

After a short introduction of the mobile agent paradigm this chapter presents some common notions of the agent field. Especially, the notion of mobility and trust are introduced. Furthermore, there is a comparison between the agent and the client/server paradigm which is followed by possible applications for mobile agents.

Chapter 2

Chapter 2 discusses the security issues arising in a mobile agent scenario. On the one hand, threats for the hosts are shortly discussed and solutions to these threats are presented. As this thesis presents a model for agent security, the remainder of this chapter consists of a broad presentation of agent security. In this context, the existing approaches are assigned to classes, for example tracing, code obfuscation, encrypted functions and fault-tolerance. Besides software-based methods, trusted hardware is discussed.

Chapter 3

The model for secure multi-agent computation is based on many sub-protocols that have to be introduced and understood before the actual protocol can be given. In order to give the reader a first glimpse on the security model at an earlier stage, this chapter introduces the notion of agent Alliances which perform secure multi-agent computations. At this point, the presentation is relatively abstract as the necessary fundament is not yet available.

Chapter 4

The most important sub-protocol of secure multi-agent computation is one for secure multi-party computation. At the same time, some other distributed sub-protocols are based on similar techniques. Therefore, this chapter is devoted to an overview on cryptographic protocols for SMPC. It concentrates on synchronous protocols as these are more efficient than asynchronous ones. As such, they make use of secret sharing schemes. So, in order to impart an understanding of the functionality of multi-party computations, the chapter starts with an explanation of Shamir's secret sharing scheme [Sha79], followed by the more powerful verifiable secret schemes and secure multi-party computation. The exemplary protocol of Hirt and Maurer from [HM01] is presented in depth, as it is proposed to be used for agent Alliances.

Chapter 5

Finally, in chapter 4 the model for secure multi-agent computation is presented. First, the different phases in an Alliance's life are discussed in a detailed but

more informal way. Then, the actual protocol along with its sub-protocols is given.

Chapter 6

After the more theoretical considerations on the security of the model, this chapter analyses the protocol behaviour under real network assumptions. It analyses how the probability of compromising an Alliance can be decreased. In this context, replay attacks and denial-of-service attacks are discussed. Also, the influence of certificates on the Alliance model is investigated. The chapter concludes with some arithmetic examples on the probability of an Alliance compromise, depending on the number of migrations.

Chapter 7

This chapter investigates the simulation of programs by arithmetic circuits. In this context simulations of the most common control structures are given. The remainder of this chapter presents the implementation of a long number arithmetic. An Alliance should be able to use cryptographic primitives like a digital signature. Since these primitives are often based on long numbers, it is necessary to discuss their implementation and complexity.

Chapter 8

Chapter 8 provides some efficient applications for SMAC, namely RSA [RSA78, RSA79] and AES. In this context, side-channel attacks that could be made possible when simulating a high-level language program are identified and solutions are proposed.

Chapter 9

Chapter 9 is the last chapter of this thesis and presents the conclusion and future work. There is a particular focus on a possible asynchronous approach to the secure multi-agent protocol.

Appendix A

Appendix A gives a summary of the cryptographic primitives used in the model for secure-multi-agent computation.

Appendix B

Appendix B provides a presentation of the UC framework of Canetti [Can00]. Multi-party protocols that are proven to be secure in this model, provide unconditional security. In addition, it is guaranteed, that a concurrent execution of protocol instances does not cause new security risks.

Appendix C

As the secure multi-party protocol of Hirt and Maurer is based on the verifiable secret sharing scheme from Ben-Or et al. [BGW88], this protocol is summarised in Appendix C.

For the sake of completeness it be mentioned here, that relevant parts of the results presented in this thesis have been published in advance. They are included in the publication list located behind the curriculum vitae. It seemed to be reasonable to publish the research results at an early stage, since the security of mobile agents is a very vivid research field.

Chapter 1

The Mobile Agent Paradigm

This chapter introduces the most important terms connected to the mobile agent paradigm. It starts with the presentation of mobile agents and their properties. Mobility is one of the most intriguing properties of agents. Therefore, it is discussed in depth by presenting three different classes of mobility. The chapter continues with a comparison between the mobile agent paradigm and the client/Server paradigm. This is of importance, since usually one of the first arguments in discussions with opponents of the mobile agent paradigm is that the latter is fully satisfactory and that there is no need for agents. The agent-oriented part of this chapter closes with a short section on application areas that are of particular interest for the implementation of mobile agents. A last section introduces the notion of trust. This is a very important pre-condition for any system in order to become widely accepted. Although being closely related to system security, trust does not offer security but at most the possibility to believe in the security of a system. Therefore, trust found its place in this chapter and is not part of chapter 2 on host and agent security.

1.1 What are Agents?

What are these *agents* that have been controversely discussed over the last decades? Why do they pose such a serious threat to servers and, at the same time, why are they so difficult to protect? Coming to a formal definition of agents one recognises very quickly that there is no uniform definition the agent community agrees on. This problem is primarily caused by the question of the properties the *basic* agent is supposed to have. The following list of potential agent properties is neither complete nor are all of the items clearly defined. They are mainly coined by human intuition. However, the list gives an survey on the most popular approaches. An agent is defined as

- *reactive/active* if it just reacts on input of its environment/if it starts activities on it own.
- *communicative* if it is able to interact with other entities in its environment. A communicative agent has the ability to actively initiate a com-

munication as well as to react to requests.

- *flexible* in case the agent's program allows different reactions on the same action.
- *mobile* has methods allowing it to migrate from host.
- *social* if it can imitate human behaviour (within strong limitations) and interact with other agent.
- *learning* if is able to adapt to changes in its environment.
- *autonomous* if it does not require to interact with its originator until it has completed its task.
- *intelligence* if it is learning, active and autonomous.
- *charismatic* if it has a kind of character.

Now, let us return to the original question of how agents are defined. The most restrictive (but obviously indisputably correct) definition may be the one considering an agent to be just a piece of code and annotated data (see also [CHK97]). This definition is not sufficient for *mobile* agents which require a certain degree of autonomy and intelligence. The latter two terms are very often used in the same context since autonomy requires intelligence and intelligence without autonomy is meaningful only to a limited extent. Thus, some sources define intelligent agents, while other sources define autonomous agents but the differences are mostly negligible. In [Wei00] Woolridge gives a general definition of an agent as

> "*. . . a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives.*"

An agent that is active, reactive and social is then defined as intelligent. These properties imply the agent's capability to recognise its environment and to react to changes, to execute a goal-oriented behaviour and interact with other agents and human beings. However, Franklin and Grasser define in [FG96] an autonomous agent without social capabilities as:

> "*An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it in pursuit of its own agenda and so as to effect what it senses in the future.*"

Other definitions have been proposed. For instance, in Game Theory Datta [Dat03] introduces an agent as follows:

> "*An agent is an entity which can receive information about a state of the environment, take actions which may alter that state, and express preferences among the various possible states. The preferences are encoded for each agent by a utility function, a mapping from the set of all states to* $\mathbb{R}$*.*"

Such agents are able to achieve their goals without external control and are thus autonomous, too.

However designed, mobile agents are software objects which are sent to other hosts in order to be executed, and as such they need a platform to be executed on. Hosts provide this platform together with additional services. This way, a host acts as server while the agent is its client. The platform specification should include mechanisms supporting agent migration, communication with other agents and hosting of several agents in sufficiently separated areas to guarantee security for the agents.

1.2 Agent Mobility

There are different qualities of mobility. Rothermel et al. [RHR97] differentiate four approaches which can be categorised into three classes.
The most limited class is *code and data transport*. Transport in this context means that exactly one transfer takes place. The class consists of the two approaches *Remote Execution* and *Code on Demand*. Remote execution means to transfer the agent's program together with a set of parameters to one specific target computer and to execute the program there. It is not mandatory to perform the transfer during runtime, it is also possible to distribute agent programs in advance. An agent that is executed remotely is able to use this mechanism to start other agents. But, while it can initiate those agents to be transfered, it cannot transfer itself to another server. Code on demand can be considered the inverse concept. With this mechanism, the target server initiates the data transfer and executes the program. Well-known examples for this concept can be found in the form of Java applets and ActiveX controls which are very common on the Internet.
The other two classes of mobility are not limited to one transfer only, as both support *migration* of code and data. The term "migration" means multiple changes of the agent's location and can be divided into *weak migration* and *strong migration*. In order to support strong migration, a runtime environment must automatically save the execution state of an agent, and be able to continue the agent execution at this point, without requiring the agent to be programmed in a specific way. As this operation brings up difficulties, especially for multi-threaded agents, the class of weak migration has been designed. In contrast to strong migration, the agent itself has to reconstruct its execution state by aid of the annotated data. This requires the programmer to code the state of the agent in variables and to provide a function using these variables in order to allow the agent to continue its execution.

1.3 Mobile Agent vs. Client/Server Paradigm

A discussion about mobile agents with an "unbeliever" starts with two arguments: First, there is no killer application for mobile agents. This means, there seems to be no application for which the agent paradigm is the best solution, worth it the investment necessary for a change of paradigm. Indeed, to my knowledge there is no application of importance that can exclusively be implemented by using mobile agents. The second argument is the one that the well-known and popular client/server paradigm is fully satisfactory for everything one could think of. This section takes a closer look at these arguments and evaluates their

validity.
In the client/server paradigm servers offer services, the clients call remotely. From the point of view of the server, the necessary software is trusted (as far as the producing company can be trusted) since it is installed locally on the servers and not sent from an unknown source over in the Internet. The system even allows the servers to offer commercial services for which their operators demand a fee. On the other hand, all servers willing to offer a service for a specific application must install the necessary software. Also, software maintenance regularly requires working hours and and additional financial efforts. Therefore, one can expect that only those services required by a large number of clients will ultimately be offered. The more fancy the application, the lower the probability to find servers that agree on installing the corresponding service. This presents a disadvantage with respect to the software heterogeneity of the Internet. There are two more disadvantages of the client/server paradigm. First, a client/server connection is a point to point connection. If one side is not available anymore (e.g. because of a system crash or shutdown), the requested service is interrupted when there was a socket connection. Even if both sides are online again, the computation is not resumed automatically.[1] The second disadvantage is the problem of communication complexity. In case the clients require large amounts of data, the impact on the network can be significant. As mentioned before, the client/server paradigm is in heavy use nowadays, the clients go into the millions, and thus the overall volume of data that has to be transmitted is enormous.
In the mobile agent paradigm the application is sent to a server in form of a software agent. Thus, independent of the number of services that are actually executed via agents, a server only has to provide an agent platform. Besides platform maintenance, no efforts are necessary. This implies that in principle each application can be executed as long as it does not require more resources than a server is willing to provide. There is no point-to-point connection between the agent's originator and the agent's host, which means, that after having sent an agent, the originating server may cut the connection. There is no further communication necessary until the agent returns with its computation result. So far the theory. In practice, it is not that simple. On the one hand, executing an agent costs resources such as processor time. In contrast to the client/server paradigm it is not yet usual to demand a fee. This is mainly the reason because a host does not know the semantics of the agent program that is executed. Consequently, it is more difficult to fix a fee. It would be necessary to count the processor time or the number of database entries the agent needed. But it is not easy to find a general agreement within one platform which actions should be payed for and which not. Also, in case of a fee, the host should be able to prove the rightfulness of the fee that has been charged. On the other hand, accepting and executing software originating from an unknown and possibly malicious source is something a host has to be convinced to do. There are profound security issues that have to be faced. This will be discussed in section 2.1. Analogously, unknown hosts can be a threat to an agent. The host has full control over the agent and could have an interest in an attack for several reasons. On the one hand the operator of the host could have financial interest in the data, the agent has, or there could be a social interest in causing damage to the originator of

[1] A packet-oriented connection connection (`http`) does not have this problem, as it offers asynchronous long-running operations using polling.

the agent. The degree of the control, a host has over the hosted agents, is the most relevant - and unfortunately, almost challenging - problem arising in the mobile agent paradigm. Thus, the client/server looks more favourably than the mobile agent paradigm. This thesis presents a security model for mobile agents, solving this problem.
The questionable correctness of computation results is a problem both approaches have in common. Violation of the execution integrity of an agent, but also manipulated data input can lead to semantically as well as computationally incorrect results. While there are methods to determine whether the execution integrity has been violated, there is no means to prevent wrong input. It cannot even be differentiated whether the wrong input is a matter of an attack or simply of an incorrect piece of information the server has stored in its database. However, there is not a big difference between the two concepts. Mobile agents are software object and the hosts act as servers offering the service of execution and data access. This is why the mobile agent paradigm can be regarded as an aggregation of the client/server paradigm and the object-oriented design. And thus it is a kind of specialisation of the client/server concept.
Although alternatives to mobile agents can be used for each agent application, there is no single alternative to the wide range of possible applications. This has already been stated by Chess et al. in [CHK97] and seems still valid today. Provided that the security problems can be solved satisfyingly, this is more than sufficient justification for the agent paradigm to become widely adopted.

1.4 Where do we Need Mobile Agents?

It is still one of the most important questions: Where do we need mobile agents? Isn't it always possible to find another solution? It is the author's opinion that there is not a single application that actually *must* use the mobile agents paradigm. It is the variety of possible applications they could be used for. In fact, there are several areas for which mobile agents are much better suited than other technologies. The reasons lie in their power to reduce network traffic, to interact asynchronously and to be able to perform remote searching and filtering. The two important areas, namely mobile clients and e-business, are discussed in this section. Beyond this, examples for the possible implementation of agents can be found in the fields of human-computer interaction (multi-modal rooms, speech recognition), embedded systems (monitoring, management of resources), system security (monitoring, virus detection/elimination) and co-operation in distributed systems like Grids and P2P.

Mobile Clients Nowadays, mobile devices such as laptops, mobile phones and PDAs have become very common. Those devices have three characteristics which strongly speak for the use of mobile agents instead of RPCs in order to offer a service.

1. Their connection to the Internet is mostly only intermittent. Thanks to wireless technologies this has changed over the last years. But still, small devices as for example PDAs and mobile phones do not possess the same computational power as desktop computers do and cannot constantly afford costly security mechanisms. A constant connection to the Internet

is thus too dangerous and often lacks privacy. Besides, IPv6 is not yet ready to securely manage a mobile device leaving one subnet and entering another one (which maybe not IPv6 [Mas06]). At the same time, network hopping poses a major requirement when using mobile devices.

2. Mobile devices usually have a smaller bandwidth. The bandwidth of modems as well as wireless connections cannot be compared to that of current cable-oriented connections, where a bandwidth of 6 GBit is widely available, today. Thus, instead of performing communication intensive information retrieval with different servers, it is preferable to send a mobile agent which completes this work directly on those servers and returns after having finished its task.

Electronic Commerce Mobile agents are an interesting technology for e-commerce applications. They can be used to store the clients needs and preferences. Agents can be used for automatic negotiations and bidding in electronic auctions. Since they involve money, these applications are very interesting aims for attackers. Therefore, they require a high security level. Although this cannot be provided yet, the paradigm seems to be interesting enough to encourage vendors to make use of it even without solving all security issues. An example for bidding agents are the so-called *spider* agents which are oftenly used by eBay[2] clients. These spiders monitor a specific auction and give a bid in the last milliseconds of the auction. eBay has forbidden the use of spiders. Those bidding agents offered officially by eBay do not offer the same functionality. They increase the bid as soon as a the current bid has been out-bidden. If several such agents are used, the bids unnecessarily spiral.

1.5 The Notion of Trust

As mentioned in the introduction, the notion of trust is of paramount importance for the development of secure systems [DoD85, Uni93]. For example, any system for user authentication needs a trusted functionality for keeping the authentication information. Examples are given by the Kerberos authentication protocol [SNS88] for open networks or by Woo and Lam [WL92] for distributed systems. In this thesis, trust is not a prime issue. It is more regarded as a consequence of the presentation of a system for secure multi-agent computation. Especially in this field of multi-agent systems, trust plays an important role for their applicability. Normally, such a system is meant for open, unauthenticated networks. This holds a lot of threats for the agents and their owners. Financial and social losses have to be worried about. Thus, a potential user of such a system will be grateful when having a reliable measure for trust at hand. Unfortunately, the meaning of trust is hardly ever clearly defined in these approaches and left to the readers intuition. A reason for this lack of a clear definition could be, that trust is a social concept, not a technical one. Thus, it is much more difficult to give a formal description of a trusted system or functionality.
In [WSB99] an analysis of possible trust relationships between different principals is given. A principal is thereby understood as a service provider in a network, such as for example any host which is offering its hardware and software

[2] `www.ebay.de`

resources to agents. A principal cannot separate its goals from its behaviour. It will always behave in a way that allows it to achieve its goals. Therefore, a prime requisite for a user allowing it to trust a principal is to concur with, or at least, to approve of its goals. Of course, in general this will be difficult as the goals of the principal will be unknown. Therefore, the authors from [WSB99] try to gather the goals of a principal in a policy, which is a set of rules that constrains the behaviour of this principal. The policy has to be written down and made available to all other principals. Then, trust in another principal is defined as the belief that it will adhere to its published policy. In order to determine whether a certain principal can be trusted, it can be checked if the published policy is acceptable and if a motivation for the belief that the principal will adhere to its published policy can be established. Again, the latter is difficult to formalise.
There are two approaches that can be used to make principals to belief that an entity will adhere to its published policy:

1. trust based on (good) reputation
2. trust based on control and punishment

The first approach is based on the experience and evaluation of users and oftenly used. After having taken the services of a principal, the user evaluates its behaviour by certain criteria (for example response time, available CPU time, availability, etc.). These criteria allow to compute a reputation value which can be checked before interacting with this principal. From the security point of view, this approach is not satisfactory since the evaluation could be done with ulterior motives for example to destroy the good reputation of a concurring principal. The second approach offers more security but is much more difficult to accomplish. Reliable control mechanisms are complex and slow down a system. However, punishment, too, is mostly accomplished by just carpeting the malicious principal. A legal prosecution would require a burden of proof. Thus, this approach may offer good security in certain systems, but in general the possible measures that can be taken to punish a misbehaving principal will not be sufficient to prevent from misbehaviour. This may be the reason why the first approach is the one that is mostly applied.
The model presented in this thesis achieves trust through a third approach:

3. trust based on the belief in a secure system design

This approach requires a system to be designed in a way that deviations from a protocol are impossible or that they do not cause harm to anyone. The measures falling in this class can be briefly separated in cryptographic ones, measures providing robustness by redundancy, and, finally, trusted hardware. All of them are extensively discussed in chapter 2.
The second approach to trust uses control and aims at prevention by creating fear of being detected and punished. It requires reliable control mechanisms which can be expected to be very complex. I name it a *passive* approach, as the system does not actively prevent fraud. In contrast, trust based on the belief in a secure system design can be regarded as an active one. The system actively handles fraud by providing mechanisms like encryption or authentication, which make misbehaviour impossible or at least very difficult. We will see in chapter 2 that these approaches, too, are very complex. But they provide a better protection and consequently a higher level of trust.

Chapter 2

Security Issues and Related Work

Mobile code poses serious threats to hosts that have no way of verifying its originator or its functionality. Vice versa, an agent that is transfered to an unknown host cannot trust on its correct execution or the integrity of its data. We will see in this chapter that while the protection of hosts against malicious code is mainly a matter of correct system installation and user behaviour, the protection of mobile agents requires much more sophisticated measures. The main intent of this thesis is the protection of mobile agents. But for reasons of completeness, this chapter starts with a discussion on host security before it turns to agent security. The latter part consists of a presentation of possible threats for mobile agents and ends with a broad survey of published security approaches. The section starts with the presentation of security measures against single attacks. Afterwards, methods using fault-tolerance are discussed. They offer the advantage of not preventing from but being able to bear a specific amount of attacks. Then, advanced methods on the field of agent security, namely code obfuscation, function hiding and computing with encrypted data are discussed. While the first and second class aim to the protection of the agent's functionality, the latter provides data privacy. The section ends with the presentation of approaches using the concept of co-operation between agents and mutual control.

2.1 Host Security

In times of the Internet and innumerable computers that are constantly online, system security has become a very challenging task. It is not sufficient to integrate network security tools like firewalls (e.g. for demilitarised zones), virus scanners, intrusion detection systems or honey-pots.[1] A careful system design and software development is very important because most weaknesses that are finally exploited can be found in software that is faulty by design. Mostly this is caused by an unreasonable schedule during the software development process and by releasing it to early. Multi-agent platforms providing

[1] A honey-pot is a computer connected to the Internet which sole purpose is to distract possible attackers from sensitive system components.

a powerful environment for mobile agents to act in, must be extremely carefully designed and implemented to provide enough security. In [EC05], we have shown that half-hearted security integrated into an agent platform (in this case the JADE platform) is not effective. JADE is a well-known example of a free platform. Important documentation on the security plug-in was missing and security measures were only rudimentally implemented. Such circumstances cause non-expert users to believe themselves to be safe and, consequently, they could act too careless. This results in even more dramatic consequences than attacks on unprotected systems, as the agent applications in a seemingly secure system can be expected to be of much more sensitive nature. The overall evaluation from [EC05] also mirrors my opinion that system security must be part of the design process and cannot be considered as a kind of add-on, which could be taken care of later.

2.1.1 Threats

In this section, we take a look at the potential threats an operator of a multi-agent platform has to be prepared for. An agent platform with the following properties serves as a basis for the discussion. The platform should

- be public. This implies that after a preliminary registration, like at a free-mail provider, everyone can install the platform and start agents.
- offer services like for example home banking, information retrieval or a portal for auctions.
- have many users. Since a portal with only a few users cannot be regarded as representative for some security problems, the number of users should be in the thousands.

Some of the attacks can be considered a threat for all users participating in arbitrary networks like the Internet, but some of them aim at specific properties of agents and their environment. As the main focus of this thesis is the protection of mobile agents and not the protection of hosts, only a short overview is given here. For more details, the interested reader may take a look at the description in [EC05] and [Dre05].

Spoofing An agent is said to be spoofing if it takes a false identity in order to gain access to restricted data or functions. *Phishing*[2] is a current application of spoofing. It is used by attackers pretending in emails or on web pages to be legitimate organisations like for example a credit institute, thus trying to spy on private data of the victims.

Trojan Horse In this attack an agent is used to transport malicious software like a virus, worm, sniffer or similar program. Nowadays, worms and viruses are used to transport so-called *backdoor programs*, allowing an attacker to access private data of the user without being detected. A well-known example is represented by the worm MyDoomM.

[2]For more information see `www.antiphishing.org`

Weed An agent is called a weed if it has no meaningful functionality and requires only few resources in order to avoid being detected. Each software that spreads massively, but does not cause direct damage, falls in this class of attacks. In case of agents which behave inconspicuous in terms of resource consumption, but appear in large numbers, an agent system can suffer from severe damage.

Freeloader Freeloaders are parasitic agents which live in a commercial environment and try to profit from free services. This class of attacks is relatively new and became important due to the increasing automation in form of agents. As soon as a provider offers free services, which are financed by the profits from commercial services, such agents can cause serious financial damage. There are numerous examples in human behaviour: Using share-ware programs for a longer period than allowed, registering at free-mail providers without correctly answering the necessary questionnaires or by-passing commercials in software which is financed through it (like the browser Opera). Although very similar in their basic idea, a human and an electronic freeloader have different main objectives. While a human generally thinks of his own advantage and is not aware that he causes damage, a freeloader agent aims on causing the highest possible economic damage to the service. This is reached by a permanent usage of free services, without using them in a meaningful way. Examples for free services that are used by agents acting as freeloaders are cost-free sending of SMS over the Internet or free consulting services offered by credit institutes.

Flying Dutchman A Flying Dutchman is a specific kind of Freeloader but with main issue on its own continuity. It tries to avoid its termination or, at least, to reproduce itself before being terminated.

Denial-of-Service Denial-of-Service (DoS) attacks aim at overloading a system in a way that important services cannot be offered anymore. Although this kind of attack is of very general nature and one of the most powerful and damaging attacks for current computer systems, it is necessary to mention it in the limited context of agent systems, too. Because of their nature, mobile agents are a natural means for the execution of DoS attacks. Depending on the platform design, agents provide a malicious user the opportunity to anonymously attack specific servers or even the whole system. Receiving an agent, it is in general impossible to determine its functionality. In some cases, it may be possible to decide, whether malicious code is contained, but the general equivalence problem of functions is undecidable. In contrast to malicious code attached to emails, where the user has learned to be careful, a user participating in an agent system and receiving an agent usually will not hesitate to execute it. Also for distributed Denial-of-Service (DDoS) attacks, in which a victim is attacked from several locations at once, a multi-agent platform is well suited. Designing an agent as a Flying Dutchman leads to such an attack, that is much more powerful than a general DoS attack, as it can use more resources.
There already are several examples for most of the attacks mentioned above, although not designed for agent platforms. But as soon as multi-agent systems become widespread, this will change. Therefore, it is absolutely important to address these problems in advance. Threats like Weed agents or Freeloaders are difficult to detect and to combat. Others, like Spoofing or Trojan horses result

from security breaches in the platform architecture and can be avoided.
The remainder of this section gives a brief overview of the options a platform operator has in order to protect her system against the threats mentioned above.

2.1.2 Security Measures

Operating System

Modern operating systems offer a high level of security against malicious processes. The most important properties are

- There is a memory protection by catching unauthorised memory accesses.
- An advanced access control mechanism e.g. through Access Control Lists (ACL). ACLs allow to assign single processes fine grained access permissions on hardware resources like memory, CPU, printers, etc.
- Processes are encapsulated by virtual memory areas.

It is possible to execute programs (like an agent platform) on virtual computers (VMware[3]). Doing so, the program is not aware of the real computer and its resources nor of programs executed on other virtual computers on the same hardware.
Communication between processes can be made possible by special libraries like for example the Message Passing Interface (MPI). Beyond other things, it allows to cluster several heterogeneous machines to one parallel system. A successful attack thus requires to trick the operating system. Therefore, a modern operating system can be assumed to offer satisfactory security. As usual, the situation is not as simple as it seems at first glance. In order to have platform-independent applications, a programming language offering platform-independence must be used. In order to achieve platform-independence, programming languages use different approaches. Two of them are discussed now.
For a given program, the language *C* generates a platform-dependent code. This does not create new security breaches, as the resulting programs are directly executed by the operating system. Unfortunately, Java, which is currently the most popular language for the implementation of multi-agent platforms, uses a different approach. Java contains a small operating system, which offers a virtual machine (VM) for the different operating systems. As a universal interface between programs and operating systems, the VM must have extensive access rights to the computer resources, independent of the nature of the programs that are executed on it. Thus, the VM can be used to by-pass the security mechanisms of the operating system. To avoid this, Java uses so-called *Sandboxes* as additional measure. Sandboxes are program dependent and allow to define fine-granulated access rights for individual programs like servlets or applets.
Java [GJS96] is the predominant language for mobile agent systems (see for example [JAD06]). This is with respect to both, implementation of multi-agent platforms with integrated mobility, and writing mobile agent applications. First, it inherently supports code mobility by means of dynamic class loading and separable class name spaces. Second, it offers several security properties. Java seems perfect for developing an execution environment for mobile agents as it offers

[3] `www.vmware.com`

many features to ease its implementation and deployment. Since Java runtime systems are available for most hardware platforms and operating systems, agent platforms that are built on Java are highly portable. At the same time, the underlying Java Virtual Machine (JVM) [LY99] offers sophisticated compilation techniques and other optimisations. Therefore, in this thesis, the programming language aspect of host security is limited to a discussion of Java security.

Java Security

In [BR02] Binder and Roth give an analysis of Java as a foundation for secure multi-agent systems. They point out the following advantages:

1. The Java serialisation mechanism allows to capture an agent's object instance graph before it migrates. The agent can easily be resurrected at the new host.
2. Dynamic loading and linking of code through a class loader allows to isolate classes and components of the agent system, providing separated name spaces.
3. Multi-threading enables the concurrent execution of agents.
4. Java is a type safe language as the execution of programs proceeds strictly according to the language semantics. In detail, safety depends on bytecode verification, strong typing, automatic memory management, dynamic bound checks and exception handlers.
5. Java provides a sophisticated security model with flexible access control based on dynamic stack introspection.

As an example, the security model of Java 2 is presented in figure 2.1, illustrating the interaction between the individual security components. For more information, the interested reader may take a look at the book of Gong et al. [GED03]. Here, only a brief overview is given.
A drawback of Java can be seen in its weakness against Denial-of-Service attacks. Resource consumption (CPU, threads, network bandwidth) is neither accounted nor controlled. Either through poor programming or malicious intents, the system could easily be compromised. The authors from [BR02] state that this security concern will be a matter until Java evolves from an application-level runtime system into a real operating system.

2.2 Agent Security

For a long time, researchers all over the world tend to think that the protection of mobile agents cannot be accomplished. In fact, this conviction is still very widespread. How can a program be protected if it is sent to an unknown server, has to be executable, is depending on unknown resources and under full control of the server and its operator? Clearly, there are numerous cryptographic tools that could be used. But on the one hand, they are not omnipotent and cannot protect the agent against all possible threats it is facing; on the other hand the performance as well as the size of an agent using these tools are contradicting the application areas of mobile agents. Therefore, many companies are using

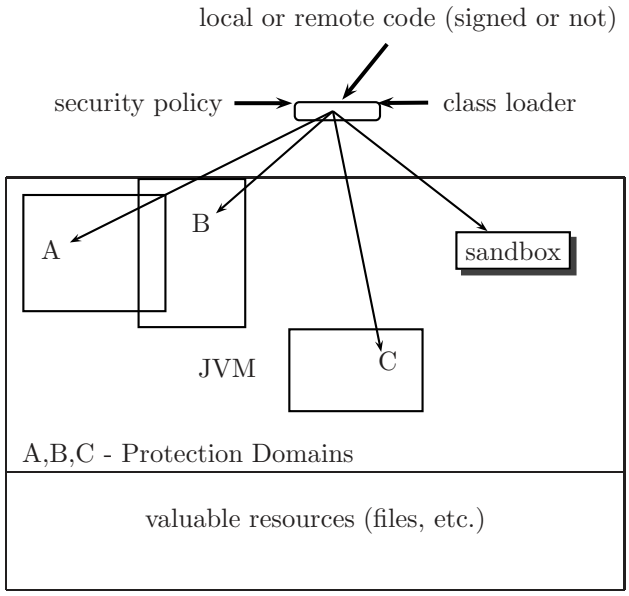


Figure 2.1: Java 2 Platform Security Model

the mobile agent paradigm without any security mechanism. The risks can be accepted if the platform is installed within an authenticated local area network, for example within a large company that is distributed over the world. But as soon as unknown outside servers are participating in any computation, a significant number of possible applications for agents, as for example electronic commerce, has to be omitted. Obviously, missing security is one of the most important reasons why the agent paradigm has not been widely accepted by the industry yet.
This section continues with a discussion of possible threats a mobile agent is confronted with in open networks as for example the Internet. Then a survey of research results in the field of mobile agent security is given. First, naive approaches, mostly using well-known cryptographic primitives are presented. Although called *naive*, they are widespread and useful as they offer acceptable performance. Then, more advanced approaches are discussed, offering a higher level of security. As usual this has to be payed with higher complexity, usually too high remain practical. The section ends with the introduction of distributed approaches, where agents protect each other by co-operation and mutual control. This directly leads to the security model presented later-on in this thesis.

2.2.1 Threats

It was mentioned above that the protection of mobile agents is challenging, if not impossible. Where does this pessimism come from? If we return to the basic definition of an agent, we consider an agent as an executable program with some annotated data. Sending this packet to another (unknown and possibly malicious) server, attacks on two levels could occur: network attacks and attacks through the malicious hosts. Network attacks happen while the agent is

transfered from one host to another. There exist

- passive attacks like spying on the agent's code and data.
- active attacks like delaying the agent for an unknown time, copying, data and code manipulation, takeover, etc.

Passive attacks are relatively easy to perform and very dangerous in case any sensitive data (private key, passwords, etc.) is transfered. At the same time they are easy to prevent by using secure channels. Nowadays, this belongs to the well-known mechanisms which are already integrated into standard libraries (see for example the `ssh` protocol). Although difficult to accomplish because they require physical network access and expert knowledge, active network attacks may happen and, unfortunately, can not be completely avoided. By using hash functions and digital signatures (see appendix A.2 and A.3), it is at least possible to detect manipulations. Encryption protects against manipulation of specific parts of the agent and against takeover, but cannot protect against delay or deleting. The latter can only be fought by security mechanisms which are based on redundancy.
Attacks performed by a malicious host are very powerful as the host has full control over the agent. The threats can be categorised into the following attack classes:

- Spying: Analysis of the agent's functionality (software piracy, violation of copyright, violation of the user's privacy with respect to his intentions and preferences) and its data (violation of user privacy, trade secrets, interaction with other agents, etc.).
- Violation of the agent's integrity: This class consists of three subclasses, namely violation of code integrity, violation of execution integrity and violation of the read-only state. Violation of code integrity aims at unauthorised changes in the agent's code and thus implies malicious changes of its functionality. Limited access to necessary resources like databases and processor time as well as unnecessary delays or ignoring (= not executing) parts of the code are violations of the execution integrity. These attacks can also be regarded as Denial-of-Service attacks. Finally, violations of the read-only state of an agent include malicious changes of the agent's data (like user preferences, account numbers or even its identity) that has been provided by the user and does not change. This may lead to incorrect computation results although the execution integrity has been preserved.
- Manipulation of variable data: Variable data consist of information the agent has collected and filtered during its journey through the network. Malicious changes of this data normally causes the agent's originator to make wrong decisions. For instance, instead of booking the cheapest flight to Paris, which has been maliciously altered, a much more expensive one is chosen. In general, significant financial losses must be feared.
- Malicious routing: Even if a predefined route belongs to the agent's read-only data, the host can decide wherever it finally sends the agent. Thus, the agent could get lost in a subnet of malicious co-operating hosts, not being able to finish its task correctly.

- Masquerading of the host: A malicious server could intercept or copy an agent transfer and start the agent by masking itself as the correct receiving host. This kind of attack can also be regarded as a network attack, because a host could function as a "host-in-the-middle[4]".

- Returning wrong results on system calls issued by the agent: By returning manipulated data to agent queries, the semantics of the agent gets corrupted. This attack can cause serious consequences as the computed results are not reliable in any way. It seems impossible to protect against attacks on the semantic level. First, it cannot be differentiated whether the host knowingly performed the attack or whether it just kept wrong information in its databases. Second, on the semantic level it is very difficult to automatically determine whether an arbitrary piece of information is correct or not.

Recapitulating, one can say that due to its mobility, an agent is faced with nearly all kinds of attack, one can think of in computer networks. At the same time, the potential damage is very high. Its whole extend depends on the authority assigned to the agent in order to act autonomously in its users name.

2.2.2 Approaches against Single Attacks

This section is dedicated to a survey on security measures for agents whose protectional power is limited to one attack type only. The means mainly consist of cryptographic primitives which can be considered standard today. But also approaches offering security by redundancy are discussed. A common nominator of the measures presented here is that they are relatively cheap in terms of computation and communications complexity. Thus, these approaches are those which are currently used in practice. For now, they seem to provide a satisfactory trade-over between effectiveness and cost for most of today's applications.

No Protection

Although "no protection" is obviously a trivial security measure,[5] it has a right to exist and to be discussed shortly. Many application areas for agents are not very attractive for attackers (for example simple web search and filtering tasks). Although there may be a loss of some kind if the agent gets corrupted, there is little profit in attacking such agents and consequently only small interest in actually performing an attack. In such cases, the cost-value ratio may allow unprotected computations. "No protection" is of particular interest for computations that allow an efficient verification of their results.

Legal Protection

Legal protection is desirable for two reasons: First, many people do not dare to misbehave in a way that is not only morally condemnable but also against some law. Second, this kind of protection has no negative influence on the complexity of the computations nor on the size of the agents. Besides protection through laws, protection through contracts are possible. For example, a server operator

[4] in analogy to the well-known "Man-in-the-Middle" attacks from cryptography
[5] if even considered as such

could be hold responsible for providing security against external attackers, for the protection of the computations against unauthorised access and to guarantee computation integrity.
The problems of this kind of protection can be summarised as follows:

- It is difficult to detect and prove breaches of contracts.
- The existence of tamper-proof log files as legal proof is necessary.
- Server operators are reluctant to take this responsibility. This is especially problematic in systems with a large number of users, as for example Internet providers are faced with.
- As the Internet is global, a standardised international regulation is required to be able to prosecute malpractice.

Trusted Networks

This class of measures contains approaches that avoid the problems by using a setting where hosts are trusted. Some of them employ a host infrastructure that is operated by a single party (see e.g. [Gen96]). This is for example given in large companies. It is also known from the field of Grid computations, where usually the Grid operator is the one in charge of detecting and punishing misbehaviour. Although in this case nothing is written down in a contract,[6] to this kind of control is often sufficient in this field. Other approaches allow a migration to trusted hosts only [FGS96]. Alternatively, they could migrate to hosts with good reputation. Rasmussen and Jansson [RJ96] first introduce the term "soft security". They claim that software agents must handle trust without human aid, but with soft means like social control. In general, the computational rules for trust and reputation must be chosen wisely in order to prevent from follow-up attacks, aiming at manipulations of trust values and destroying the reputation of honest parties.

Trusted Hardware

Another means to create trust in hosts is trusted hardware. This method does not aim at avoidance but on the impossibility of any attack and is achieved by using tamper proof hardware. This approach has been mentioned early by Yee [Yee94], Chess et al. [CGH+95] and by Palmer [Pal94]. Although originally not designed for mobile agents, the approach is of significant interest for this domain. It allows an agent to execute sensitive functions like e.g. a digital signature on an unknown host in a secured way. In [WSB99], trusted hardware is used to protect the predefined itinerary of an agent. On the other hand, Schneier states in [Sch00], that he doubts that it will be possible to prevent such hardware from physical side-channel attacks. The approach is inflexible and very costly. Also, it could open the door for monopolistic hardware producers. It must be doubted that users will accept those disadvantages in order to profit from the advantages of a multi-agent system. A well-known and recent example for the public rejection of trusted hardware can be seen in smart cards for home banking. A protected processor allows secure authentication but requires

[6] Some companies use contract-like documents to control their employees.

to buy a card reader. Users did not accept this technology for several reasons like the additional costs for the reader and being dependent on specific software. Still, they prefer to use the unconvenient authentication method using personal identification number (PIN) and transaction numbers (TAN).

Protection of Code Integrity and Read-Only State

Both, code and read-only data of an agent, do not change during its lifetime. More specifically, they are not allowed to change. Such data can be protected by adding a digital signature of the originator. At best, there is a trusted certification authority, but for most (private) applications concepts like a web of trust are fully satisfactory. A signature enables hosts to check whether code and data have been maliciously changed. As such, hosts can make the decision whether they are willing to execute an agent on this basis. The level of trust, a server operator has in an authenticated agent can be much higher than he can have in unknown code originating from an unknown source. The advantage of this tool lies in the assumption that honest hosts will inform the originator or another instance in charge about the attack. On the other hand, using digital signatures only allows an attacker to substitute parts of the agent and add her own signature. This is similar to a takeover, but at least the hosts can be relatively sure not to execute code that could be dangerous for them.

Protection of Communication Privacy and Prevention from Malicious Changes

Privacy of communication can be achieved with symmetric encryption like the Advanced Encryption Standard (AES). This also results in the impossibility of attacks which aim at specific parts of the agent or its communication. Malicious changes of encrypted data lead to the impossibility of meaningful decryption and will be detected.

Prevention from Malicious Routing

A very common mean to prevent from malicious routing is to attach a digitally signed itinerary to the agent. Yee mentions this approach in [Yee99] assuming authenticated and encrypted channels. He proposes to use a public certification chain and to attach the root certificate to the agent. This enables all hosts to check the origin of the agent and the correctness of its route. Unfortunately, this does not offer the agent the opportunity to check the identity of its host. Since the server has full control over the agent it could just by-pass the cryptography-based identity query. If there is only one malicious host on the route causing the agent to take a detour over a series of malicious hosts, the attack is detected as soon as the agent returns to an honest host of its route. Thanks to the authentication, this is possible by checking whether the former host is on the agent's itinerary. Figure 2.2 demonstrates the situation if more than one malicious host lies on the agent's itinerary. The depart from the route cannot be detected if and only if the departure starts and ends at a malicious server.
Digitally signed messages like "The agent was on host x" would at least enable the user to check whether all hosts on the route have actually been visited. But as long as there is no mathematical entanglement between the signatures, it is possible to remove specific parts and thus to pretend that the itinerary

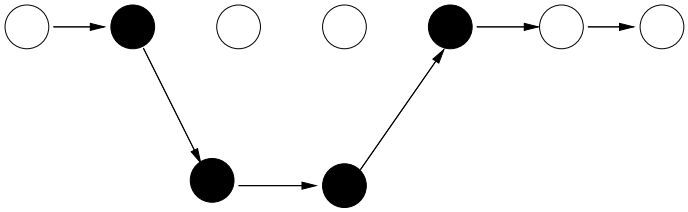


Figure 2.2: Undetected malicious routing in the presence of more than one malicious host

had been violated. In fact, there are many security approaches in the field of malicious routing. Most of them, which do not use a predefined route, require regular communication with trusted servers. This contradicts the agent's autonomy. At the same time, limiting the agent by an inflexible route causes the same contradiction. The problem of malicious routing is challenging and not yet solved.

Protection of Read-Write Data

Variable data is difficult to protect. If the data from the single hosts are not entangled in any way, it could be protected by means of digital signatures and become read-only data. In general, this will not be possible. This results in the following situation.

- An agent that has left a malicious host cannot trust its complete memory. Unfortunately, an agent will not be able to determine whether the last host was malicious or not. Consequently, it cannot trust its memory at any point in time after having left the originator's server.
- Data is only trustworthy if
  - it has been generated after having visited the last malicious host on the route, and
  - if its computation was independent from data of former hosts.

  This, too, is only of theoretical interest as it is most unlikely that the agent or its originator are able to determine which hosts have been malicious.

Works addressing the integrity of collected data can be found in [AKG98, Yee99] using hash chaining as the basic technique. As mentioned before, data must be somehow entangled. Hash chaining is such an entanglement. Basically, all collected data is stored along with a hash value computed by the visited hosts. In detail this means that each host H_i computes subsequent hash values $h_i(h_{i-1} \mid d_i)$ out of the hash value of the former host H_{i-1} and the data d_i it added to the agent. Unfortunately, data collection schemes using hash chaining are static with respect to data updates. For more dynamic scenarios like distributed auctions they are not suited. Loureiro et al. propose in [LMP01] an alternate approach, based on a cryptographic hash function and the difficulty of computing a discrete logarithm (the so-called *Diffie-Hellman Assumption*). They also compute hash values of the collected data, but define a set hashing which allows to make

authenticated updates. In addition, a verification of the data is possible at any time. The protocol is of particular interest as the integrity verification is independent of the sequence of data transmissions. The complexity of the verification is not computationally intensive (one exponentiation) and almost independent from the number of visited hosts.

Tracing

Tracing aims at the detection of violations against execution integrity. In [Vig98] Vigna proposes to force hosts to log all queries the agents issues to the server. As in general this protocol will be too large to add it to the agent's data, the host is obliged to store it for a specified time and to protect it against subsequent changes by providing the agent with a signed hash value that has been computed over the log file. In case the originator suspects malicious behaviour it can ask for the file and check its correctness. The approach is not satisfactory as the originator only receives the computation result and the hash values. Thus, it is impossible to find out which one of the visited servers was possibly malicious. To check all protocols per default is out of the question as it is as expensive as the computations themselves.
Vignas approach can be improved using techniques from the domain of probabilistic proofs. Biehl et al. state in [BMW98] that the data volume necessary to verify the protocol can be drastically decreased. The protocol must be transformed into a specific certificate, also called a *holographic proof*, which can be verified through a constant number of its bits (independent of its length). Until now, the problem suffers from the absence of a an efficient transformation from conventional log files into holographic proofs.

Fault-Tolerance

Although a suitable combination of cryptographic primitives is able to protect an agent against several attacks, this approach is not satisfactory. There are too many possible attacks, significantly diverging in their nature, to be able to offer protection against all of them. Therefore, it is worth it to take a look at an alternative approach already used in hardware design: fault-tolerance. In this context, fault-tolerance means redundancy only, co-operation of agents is left out and discussed in section 2.2.4 as it promises to be a very promising approach.
Minsky et al. did an exemplary work in [MvRSS96]. They faced the problem of performing a reliable distributed computation in the presence of malicious servers. They did not consider an agent scenario. Nonetheless, their method could also be used for the protection of agent computations. Their idea is basically to divide a computation in several phases, where in each phase identical computations are performed on n servers instead of one. After each phase, the servers send their intermediate results to each server of the next phase. Those can make a voting on the received results and determine the data with which to proceed. The method is illustrated in figure 2.3 for three servers in each phase. The model tolerates all errors/attacks, as long as a majority of the servers in each phase is honest and able to transmit their intermediate results to the servers of the next phase.
Obviously, in this example, two co-operating servers of two different phases could

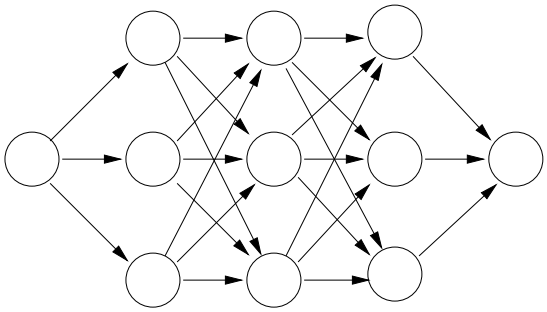


Figure 2.3: Server replication with voting

pretend to be servers of the last phase and trick the receiver of the computation results. The realisation of this attack is illustrated in figure 2.4.

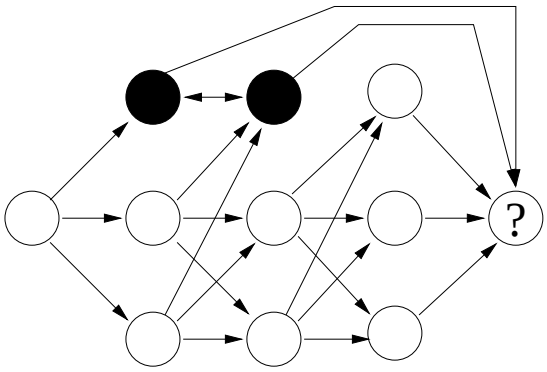


Figure 2.4: Attack on the server replication approach

Minsky et al. propose to use a secret sharing scheme to distribute intermediate results in order to prevent from such an attack. Still, the model is unrealistic as each replicated server must have the same information. In addition, server replication mostly results in very similar hardware and software. In case, one server gets compromised, all of them are at risk. This is a contradiction to the fault model of [MvRSS96], which assumes that replicated servers fail independently. In an overall evaluation, this approach is not convincing. However, the way of evaluating functions in parallel can be considered as a prequel to the model for secure multi-agent computation presented later-on.
An approach to a fault-tolerant collection of trade offers can be found in [Yee99] based on an agent replication model. Yee limits the number of malicious hosts on the agent's predefined route S to one and excludes Denial-of-Service attacks. The model is based on the secure channel model and meant for agents that are supposed to collect trade offers. If all these assumptions are kept, malicious changes of the collected offers are either detected by the originator of the agent, or a betrayal does not cause financial harm. The main idea of this approach is to send two agents A_1, A_2 to visit all servers on route $S = s_1, s_2, \ldots, s_n$, where A_1 starts at server s_1 and ends with s_n, while agent A_2 starts at server s_n and

ends at s_1. The approach is illustrated in figure 2.5 with one malicious server.

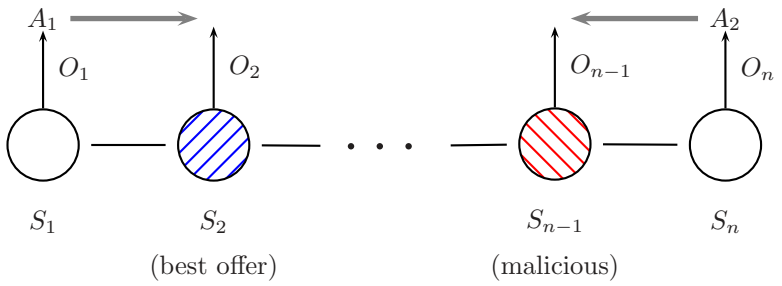


Figure 2.5: Agent replication in presence of one malicious server

Host s_{n-1} acts maliciously. A_1 already collected the best offer O_2 at server s_2 when reaching s_{n-1} which implies that it will either change the memory of A_1 or adapt its own offer accordingly. On the other hand, s_{n-1} has no interest in changing the memory of A_2 as it did not get the best offer yet, when reaching s_{n-1}. After having returned, the originator can compare all offers and in case he detects differences, he knows that an attack took place. Although functioning, this approach is too limited in its application area and demands too strong and unrealistic preconditions.

Summary

The naive approaches offer security against very specific threats. Unfortunately, mobile agent applications tend to be very security sensitive as they aim to act in the originator's name. So, in this field, the simple approaches cannot provide a satisfactory security level. In order to increase security, several of them have to be combined. A proof of security is very difficult in this case as the combination of cryptographic protocols can cause new security breaches. In addition, the more protocols are used, the higher is the risk of erroneous implementation. Approaches offering fault-tolerance seem promising but waste a lot of hardware resources. Their security is only statistical and strongly depends on network assumptions. On the other hand, they do not require as much computational power as cryptographic methods.

2.2.3 Advanced Approaches

Code Obfuscation

In the field of agent security, not only the protection of data is of interest. Depending on the application, it could also be necessary to keep the algorithm that is implemented in an agent private. One reason could be that parts of the algorithm are copyrighted and should not be open to an analysis. Another reason is the prevention from meaningful manipulations.
Code obfuscation means messing up a computer program in a way that it becomes unreadable. It is hard to read uncommented code. Even if it is written by one's own hand and seems (while programming) to be well structured it may give the impression of a complete chaos looking at it some weeks later.

This could be considered a kind of natural incompatibility between computers and the human brain. Thus, intentionally obfuscating programs immediately suggests itself as a solution to keep code secret.
Code obfuscation is done by so-called *mess-up* algorithms. Given a program P, arbitrary input x and output $P(x) = y$, a mess-up algorithm generates another program P' with the same property: $P'(x) = y$. This means, P' computes exactly the same result as P, input and output are in clear-text, but P' looks different.
In [Hoh98] Hohl applies the mess-up approach to mobile agents by defining a black-box property:

Definition 2.1. *If at any point of time*

1. *neither the agent's code nor its data can be read and*
2. *neither the agent's code nor its data can be manipulated,*

an agent is called black-boxed.

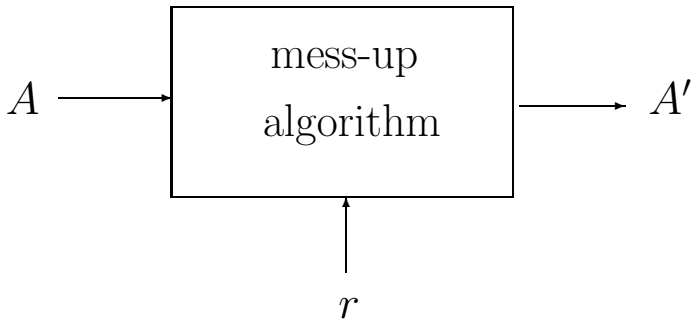


Figure 2.6: Generation of blackbox agents

As illustrated in figure 2.6, the mess-up algorithm gets the input agent A as well as a random parameter r. The latter guarantees an indeterministic behaviour of the black-box, which means that feeding the agent specification A into the black-box twice does result into two different obfuscated agents A' and A''.
Key ingredients of mess-up algorithms are

- variable recomposition
- conversion of control flow elements into value-dependent jumps
- deposited keys

Obviously, it is not possible to prove that the analysis of a messed-up program falls into a specific complexity class. The duration of an attack strongly depends on the computational power of the attacker, the kind of obfuscation that has been used and the attackers programming experience. However, until now those algorithms are not strong enough to protect a program for an unlimited time. Therefore, Hohl suggests the the so-called *time limited black-box security*. The definition is the same as in definition 2.1 with the restriction that the properties

1 and 2 must be kept for a predefined time interval t instead of for unlimited time. In this setting, each agent is assigned an expiration date that is signed by a trusted authority. As soon as the expiration date has expired the agent and its data must be considered corrupted. Servers should reject to execute such agents since they could contain malicious code (like a virus or worm).
Compared to cryptographic solutions this approach can be regarded as efficient. Black-boxed agent code is larger than "normal" code, efficient library calls cannot be used, but after all it is still executable code which does not require any costly encryption or decryption. Nonetheless, code obfuscation has two security flaws:

1. The determination of the time interval t during which the agent is supposed to be uncorrupted is difficult.

2. Mess-up algorithms must be very carefully designed, also taking into account that there are tools for decompiling code. We did some experiments showing that compiling and decompiling messed-up code returns a much better structured and readable program that may break under analysis much earlier than the messed-up one.

Actually, the security of such black-boxed agents is more a matter of believe than a matter of mathematical proof. Besides these more practical problems, code obfuscation faces some serious theoretical problems. In 2001 Barak et al. formalised the notion of circuit (program) obfuscation [BGI+01] via the *virtual black-box property*. It states that any predicate that can be computed in polynomial time from the obfuscated circuit can also be determined by an input-output analysis of the original circuit (i.e., given black-box access to the circuit). Barak et al. showed that for specific classes of functions there cannot exist an obfuscating compiler that works as a black box (such as mentioned in the approach of Hohl). In 2005, Wee [Wee05] shows under various complexity assumptions that for the class of point functions, which consists of all Boolean functions of the form $I_x(y) = 1$ if and only if $x = y$ an obfuscator exists. In the same year, a more negative result was published by Goldwasser and Kalai [GK05] proving that wide and natural classes of functions (filter functions, pseudo random functions [GMR86], secure encryption algorithms [GM84])are not obfuscatable with respect to auxiliary input. Their impossibility results are unconditional and do not require any intractability assumptions (such as the existence of one-way hash functions).

Function Hiding

In some cases it might be possible for a user A to send its input x to another server in order to run a secret algorithm applied to x and to get the result returned. This is not possible if x contains data that is legally forbidden to be given to an unauthorised party, for example in case of medical data. This problem can be solved by having methods at hand to "change" code in a way that an analysis of its functionality (for instance by performing an input-/output analysis) is fruitless.
In contrast to code obfuscation, *Function Hiding* is based on an encryption function E which is used to obfuscate a function f that is to be evaluated. Finding such methods is challenging since the result of $E_k(f)(x)$ for an input x

must be reconstructable in polynomial time using the secret key k. in addition, it must be computationally impossible to gain information on k when knowing $E_k(f)(x)$.
Function hiding allows two parties, Alice and Bob, to execute a protocol in which Alice gets to know her function f as well as the result $f(x)$, while Bob only knows his input x. The protocol is illustrated in figure 2.7.

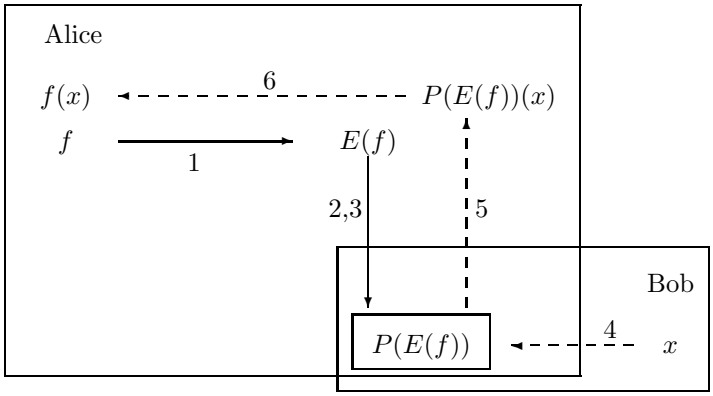


Figure 2.7: General approach to function hiding

Alice encrypts the function F and creates a program $P(E_k(f))$ that implements $E_k(f)$. This program is sent to Bob in order to be executed on Bobs local input x. Bob sends $P(E_k(f))(x)$ to Alice, who decrypts it and obtains $f(x)$. Bob can neither understand nor manipulate the computation as it is encrypted. Function hiding does not primarily aim at mobile agents, an thus it only covers one migration. A generalisation to multi-hop agents is possible [ACCK01, CCKM00]. Let $H_1, \ldots, H_n$ be the hosts on the route of the agent. Then, the originator creates an encrypted program P_i for each host H_i having the current computational state x_i and the host's data y_i as input and computing the computational state X_{i+1} as output.
As mentioned before, the design of suitable encryption functions E is challenging. One of the major obstacles is the necessity of being homomorphic. Otherwise, Alice could not reconstruct $f(x)$ out of $P(E_k(f))(x)$.
There are only few approaches on function hiding. They can be categorised into *encrypted functions* and *encrypted circuits*. For both classes, exemplary publications are discussed.

Computing with Encrypted Functions (CEF) The problem that is dealt with in this section was was first introduced by Abadi et al. in [AFK89]. It focuses on hiding data from an oracle, which means computing with encrypted data. Based on this, Abadi and Feigenbaum presented in [AF90] a first two-party protocol to secure circuit evaluation, which offers circuit encryption as well as confidentiality of data. The drawback of this protocol is the large number of interactions between the parties. In [Baz98] Bazzi formulates the first non-interactive protocol. It produces clear-text results and is based on the evaluation of a binary decomposition of all possible terms of a polynomial. The result of

the function can be computed by selecting the results corresponding to the components of the function.
Another non-interactive protocol is described in [ST98c, ST98b, ST98a] by Sander et al. using a homomorphic encryption function based on the encryption scheme of Goldwasser and Micali [GM84] to encrypt polynomials.

1. Alice encrypts f and sends the resulting Program $P(E(f))$ to Bob.
2. Bob executes $P(E(f))(x)$ on his private input x and sends the result y to Alice.
3. Alice decrypts y with E^{-1} and gets $f(x)$

The system could also be used for software distribution which would imply that Alice returns $f(x)$ to Bob in step 3.
It is easy to see that this approach requires a homomorphic encryption function E as Alice must be able to apply E^{-1} to the encrypted output. There are cryptographic functions such as RSA [RSA79] that are either additively or multiplicatively homomorphic, but it is unknown whether a strong cipher exits that is both. In fact, the property of a homomorphic function is to preserve the structure of the underlying mathematical object (ring, field, etc.). A function being a homomorphism of rings, which means, it is additively *and* multiplicatively homomorphic, would probably preserve too much information to be a good cipher. Until now, no-one has found such a cryptographic function.
Sander and Tschudin try to improve the situation by substituting multiplicative homomorphy by *mixed multiplicatively homomorphy*:

Definition 2.2. *Let R and S be rings. An encryption function $E : R \to S$ is called* mixed multiplicatively homomorphic *if there is an efficient algorithm* `MIXED-MULT` *that computes $E(xy)$ from $E(x)$ and y without revealing x.*

This limitation enabled Sander and Tschudin to propose encryption schemes using encryption functions $E : R \to S$ for the class of polynomials in n indeterminates over a ring R. Let $p = \sum a_{i_1 \ldots i_s} X_1^{i_1} \cdots X_s^{i_s} \in R[X_1, \ldots, X_s]$. Then Alice creates a program P for p as follows:

1. Each coefficient $a_{i_1 \ldots i_s}$ is replaced by $E(a_{i_1 \ldots i_s})$.
2. The monomials of p are evaluated on the input $(x_1, \ldots, x_s)$ and stored in a list $L := [\ldots, (x_1^{i_1}, \ldots, x_s^{i_s}), \ldots]$.
3. The list $M := [\ldots, E(a_{i_1 \ldots i_s} x_1^{i_1}, \ldots, x_s^{i_s}), \ldots]$ is produced by `MIXED-MULT` for the elements of L and the coefficients $E(a_{i_1 \ldots i_s})$.
4. The elements of M are added up by calling `PLUS` (as it is known from the ring S).

Unfortunately, there is an information leakage since only the coefficients of the polynomial are encrypted. The skeleton remains the original one. There are applications that allow to recover the whole polynomial, for example when using the RSA function $m \mapsto m^d$ that has only one non-zero coefficient. A known-plaintext attack is then feasible.
As a last example for non-interactive function hiding, the approach of Loureiro et al. [LM99] which uses a public key cryptosystem based on error correcting

codes be mentioned. The security of such a cryptosystem relies on the difficulty of decoding or finding a minimum weight codeword in a large linear code with no visible structure. It is proven in [AM88, Gib91, Gib95] that using Goppa codes for a public-key cryptosystem results in computational security. Since encryption and decryption are less complex as for example [ST98a] the protocol is briefly presented here and illustrated in figure 2.8. Let G be a generating matrix for

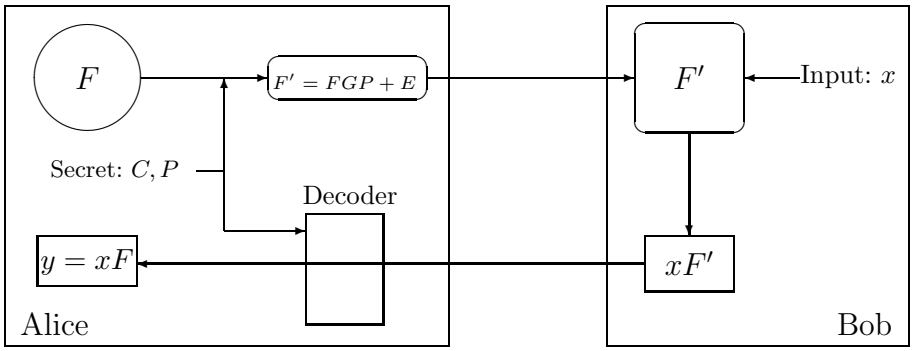


Figure 2.8: Function hiding with error correcting codes

an $[n, k, d]$ Goppa code C and P a random $n \times n$ permutation matrix. Let E be a random $k \times n$ matrix, where at least $n - t$ columns exist of the null vector. G, P, E are kept secret by Alice. Let $F \in \mathbb{F}_2^{k \times k}$ be the matrix representing the function f. Then, the protocol is as follows:

1. Alice encrypts f by computing $F' = FGP + E$ and sends it to Bob.
2. Bob evaluates F' at $x \in \mathbb{F}_2^k$ by $y' = xF'$ and sends the result to Alice.
3. Alice decrypts $y_1 = y'P^{-1}$, and uses the secret decoding algorithm C to retrieve the clear-text $y = xF$ from $y_1 = xFG + xEP^{-1}$ (where xEP^{-1} is a correctable error vector).

Encrypted Circuits Encrypted circuits offer the possibility the secure evaluation of functions that can be represented as an arithmetic circuit on an untrusted host. Yao [Yao86] has first presented a technique for circuit encryption. The method is complex but may be useful for sensitive applications that have a small circuit representation. Therefore, Algesheimer et al. [ACCK01] make use of this technique in their security model. The approach is based on a minimally trustworthy service provider T, a third party that is constantly online and guarantees the secure execution of the agent's code. The protocol requires that T cannot gain any information on the computations and the computations should cause as little interaction between the parties as possible. The expression *minimally trustworthy* results from the necessity that T could be an arbitrary service provider with the only restriction that it is not allowed to have common interests with either the originator or the hosts. In general this is a strong assumption and cannot be guaranteed. Thus, beyond the high complexity required to generate an encrypted circuit, the applicability must be considered as limited.

Computing with encrypted data – CED

The problem of computing with encrypted data (CED) can be described as follows: Alice has a private input x and would like to remotely compute a function f on her input without revealing it. Bob has an algorithm that computes f. Alice should not be able to learn anything substantial about the algorithm by the result $f(x)$.
Brassard and Crépeau presented in [BC87] a protocol where Alice could convince Bob of good results achieved by a Boolean circuit simulation, without revealing her inputs. The approach uses zero-knowledge interactive proofs and provides unconditional security for Alice's input. This protocol does not provide circuit hiding.
Abadi and Feigenbaum [AF90] presented a protocol for secure circuit evaluation which allows a party to evaluate its data on another party's boolean circuit, while preserving the confidentiality of its data. Although it was originally intended for data confidentiality, it also deals with the problem of function hiding. In fact, Abadi and Feigenbaum have pointed out the following relationship between computing with encrypted data and computing with encrypted functions:

> *"It is also clear from this description that the distinction between 'data' and 'circuits' is unnecessary. If [Alice] has the ability to hide a circuit, then [she] can also hide some private data, simply by hardwriting it into the circuit. Conversely, in protocols in which [Alice] has the ability to hide data, [she] can also hide a circuit through a detour: [Alice] can run the protocol, take the circuit for f to be a universal circuit, and use an encoding of the circuit [she] wants to hide as an input."*

one drawback of the protocol in [AF90] is a large number of interactions between the two players, since it requires several round of message exchanges between Alice and Bob.
In fact, this is a problem of both approaches discussed here. This is a contradiction to the agent methodology, even more so as both have a high computational complexity.
Sander and Tschudin state in [ST98c] that while the statement of Abadi et al. is theoretically true, it is computationally infeasible to reduce CEF to CED. This suggests that CEF may be a much harder problem than CED.

2.2.4 Security through Co-Operation

As shown in sections 2.2.2 and 2.2.3, there is a variety of approaches in the domain of agent security. But they suffer from one or more of the following disadvantages:

- They protect only against single attacks.
- They are too inefficient.
- They require a trusted party.

Although approaches to fault-tolerance seem to be the right direction, they are not really satisfactory because of their inefficiency. It must be considered a big plus that they offer robustness and thus can handle malicious environments.

Therefore, it seems promising to follow this line of research while aiming at more efficient methods. It is obvious that the degree of redundancy can be lowered in case the agents co-operate with and control each other. Co-operation allows regular security checks during the agent's journey and helps to avoid follow-up faults. This increases the efficiency of the agent's computations but, at the same time, increases the communication complexity, too. This is contradicting the mobile agent paradigm which demands a reduction of network traffic. But on the other hand, as long as no trusted authority is required, the merits countervail this disadvantage.

Making agent computations more robust is a relatively new research area and Roth [Rot99] was in 1999 one of the first (and is still one of the few) researchers going in this direction. His approach aims at the protection of the agent route. He proposes to send two agents provided with redundant information on their routes. As soon as one of the agents detects a deviation of its data from the other agent's data, it informs the originator and terminates itself. It is also possible to hide data when using a threshold scheme, if it is possible to publish the data on demand (cd. electronic money). A negative side-effect is the liability for Denial-of-Service attacks that can easily terminate the computations of the agents. It is likely that Roth also saw this problem, as he states that an extension to more than two agents would be possible. But he did not implement this suggestion.

Repeatedly, the idea of implementing protocols for secure multi-party computation for robust and distributed computations between n parties has been mentioned [ST98c, ACCK01, Yee99] as a possible solutions to mobile agents computing with hidden data. No-one did ever realise this idea. Protocols for secure multi-party computation are cryptographic protocols providing strong security guarantees. It is a quite young research field and the approaches published in the 1990s suffered from a complexity between $O(n^4)$ and $O(n^6)$, depending on the assumed network. Eventually, in 2001 the first protocol [HM01] was published offering a communication complexity of $O(mn^2)$ for an n-party circuit evaluation with m multiplication gates. This was the first one worthy to be considered for implementation. Until now, no-one has accomplished this task. So, this thesis is the first publication discussing it in detail, focussing on a detailed discussion of a synchronous multi-party protocol.

Chapter 3

SMAC in a Nutshell

Chapter 2 gave a broad survey on security measures which can be used to protect mobile agents in a multi-agent system. At the same time, it made clear that all these approaches are not satisfactory. The simple approaches for single agents merely allow to detect manipulations. Although some approaches might be suitable to avoid meaningful manipulations of code or data, a random manipulation can neither be avoided nor repaired. The situation is difficult as the mobile agent is actually located on a malicious host which could simply delay its execution in order to have more time for an analysis or manipulation. Heuristic approaches seem useful from the complexity point of view but do not offer any security guarantee. The latter is indeed provided by the cryptographic approaches which have an inacceptable complexity, the necessity of a trusted party or they are only designed for limited classes of functions. Regarding this situation, it seems promising to proceed with research on an approach achieving security by co-operation. Co-operation of mobile agents is not limited to information exchange but can also be used to achieve distributed and robust computations. Such an approach does not aim to prevent from attacks but can tolerate a specific number of corrupted agents. This is much more realistic than trying to protect single autonomous agents against all possible attacks. The costs resulting from the redundancy involved in this approach are justified, even more so as costly encryption mechanisms become obsolete.
In this thesis, a security model using secure multi-agent computations (SMAC) is presented which makes use of groups of co-operating agents. In detail, this means,

1. A user-defined function F is jointly evaluated by a group of n agents.

2. As long as no more than a specific number $t(n)$ of agents provide malicious input to the function, an output is produced which is guaranteed to be correct.

3. At any time during the evaluation, input provided by the agents and intermediate results remain private. Also, the output cannot be understood by an attacker which is controlling not more than $t(n)$ agents.

3.1 Introduction

Multi-agent systems are distributed systems. Why not use this property for the design of a group of co-operating agents? The main idea is to generate n agents which are sent over the Internet and are hosted on different servers. These agents execute an n-party protocol, called *protocol for secure multi-agent computation*. This protocol offers the agents to securely share their computational state and to perform robust and private computations. This group of co-operating agents is called an *Agent Alliance*.

3.1.1 Alliance Generation

For a given functionality F, the originator of an Alliance generates an agent offering F as she is used to do (for example as a Java program). This agent is called a *virtual agent* as it is never sent into the Internet. Instead, an interface, called the *protocol compiler* transforms the virtual agent into an Alliance of n agents that implements the same functionality as the single agent did. Each member of the Alliance has the same program, that is represented an arithmetic circuit. The original knowledge of the virtual agent is divided into n so-called *data shares*. This is done by using suitable methods like secret sharing (see section 4.2), in order to generate data pieces which reveal the original information if and only if a specific minimum number of shares are available. The data shares are distributed among the Alliance members so that only a majority of agents is able to reconstruct the data. At the same time, up to $t(n)$ co-operating corrupted agents cannot gain any information on this data. Each agent owns a small operating system that allows it to coordinate the distributed computations, an arithmetic circuit representing the user-defined functionality F, and one data share.
Figure 3.1 illustrates the generation of an Alliance, using the protocol compiler. As a consequence of this definition, the members of an Alliance do not act independently of each other, but follow a shared agenda which is the robust, private and fair computation of the functionality of the virtual agent from Figure 3.1. This means:

1. *Robustness:* Even if some of the Alliance members get corrupted and are controlled by an adversary, the result of the Alliance computation remains correct.

2. *Privacy:* Input and output data as well as intermediate results are private in the sense that only a minimum number of co-operating Alliance members is able to reconstruct data. Single hosts or other parties cannot gain any information.

3. *Fairness:* Corrupted agents which prematurely leave the protocol cannot gain any information on intermediate results or the final result.

As the adversary has full control over corrupted agents and knows everything they know, the data distribution must be done with suitable methods. In detail, this means that it is necessary that an adversary controling up to $t(n) < n$ agents cannot gain any information. At the same time, the data must be transformed in a way that still allows to evaluate the function on the resulting distributed input.

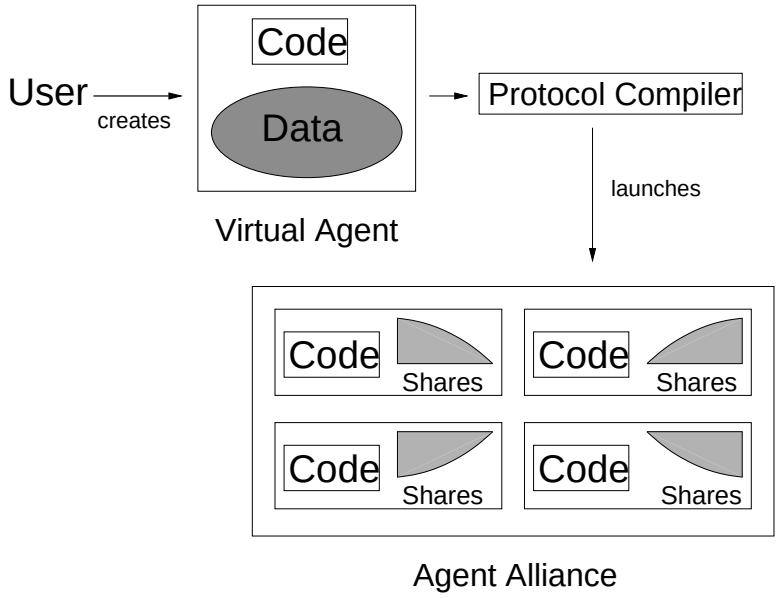


Figure 3.1: From agents to agent Alliances

All requirements mentioned above can be met when using a protocol for *secure multi-party computation* (SMPC). Protocols for SMPC are cryptographic protocols which allow a group of n untrusted parties to correctly and privately compute a function on n inputs. In order to give a proof of security, each protocol must assume that some predefined assumptions are met during protocol execution. These assumptions vary (network type, strength of the adversary, etc.) but all of them have in common that they guarantee correct computation results if an adversary corrupts at most $t(n)$ parties. The value $t(n)$ is from the set $\{\lceil n/5 \rceil - 1, \lceil n/3 \rceil - 1, \lceil n/2 \rceil - 1\}$, depending on the level of security that should be provided. Usually, protocols offering cryptographic security provide a redundancy of $\lfloor n/2 \rfloor$, while unconditional security requires not more than $\lfloor n/3 \rfloor$ to be malicious.
Protocols for SMPC lack one property which is necessary for their use in multi-agent systems: They do not provide mobility to the parties. In an agent system the behaviour of an agent must be identified with that of its host. This is because the host has control over the agent. As soon as a host is malicious, the agent must be considered corrupted. Regarding a phase during which all agents are hosted on different hosts and no migration takes place, the hosts are therefore regarded as the parties involved in the SMPC protocol. The introduction of migration poses new threats:

- Once an agent is corrupted and no additional means are taken, we have to assume a corrupted agent to stay in this condition for the rest of its life. In a naive approach, corrupted agents migrate to other hosts and cause their infection. Thus, the number of corrupted hosts in the network as well as the number of corrupted agents increases over time. Consequently, the probabilities that honest agents migrate to malicious hosts and that the

upper bound t is exceeded increases over the time.

- If different members of the same Alliance visit the same host, the adversary must be modelled as a mobile one that could gather information about more than $t(n)$ shares although it controls at most $t(n)$ hosts at any point in time. In this case, the security of the protocol would be broken although the assumptions of the SMPC protocol are kept. The model for secure multi-agent computation is built upon secure multi-party computation and should be secure under the same assumptions. Therefore, there must be adequate mechanism like a regular data re-sharing to prevent from this problem.

- The migration process cannot be realised within the SMPC protocol as such protocols are designed for n fixed parties. Agent migration thus requires to invoke a new protocol instance for the n new hosts. This opens up two security problems: On the one hand it must be guaranteed that the former host is somehow forced to leave the protocol or at least to have its knowledge about the agent data invalidated. If this is not the case, a host could collect more than $t(n)$ data pieces over time (by hosting different Alliance members) and reconstruct intermediate results. On the other hand, it has to be ensured that the new host really gets the agent and data it is supposed to get.

Because of these reasons a protocol for SMPC alone is not suitable for an agent Alliance. Additional (cryptographic) methods must be introduced. The resulting protocol which is presented in chapter 5 is then called a protocol for *secure multi-agent computation* (SMAC).
Independent of the SMPC protocol that is ultimately used, the following measures must be taken when implementing SMAC:

- One assumption that is not one of all SMPC protocols is the availability of an authenticated network. On the one hand, this allows a reliable authentication of the hosts and prevents from spoofing attacks. On the other hand, it allows us to use very efficient protocols.

- The degree of redundancy is depending on the sensitivity of the application as well as on the hostility of the network. This implies that the user should choose the size n of his Alliance accordingly. He should also keep in mind that there is a dependency between protocol complexity and n that is at least quadratic. Consequently, the user needs to fix a trade-off point between complexity and security.

- To prevent from malicious routing, the Alliance makes a common decision on where an agent is supposed to migrate. In order to maintain the same security guarantees as the underlying SMPC protocol, this decision should also be $t(n)$-robust.

- It is desirable to protect honest hosts by having means at hand to detect and eliminate corrupted agents before they migrate to another host. Detection will not be possible in general, but the nature of SMPC allows an Alliance to securely generate new agents in a distributed computation. This results in an ability to self-repair and has a major impact on keeping the number of corrupted agents below $t(n)$.

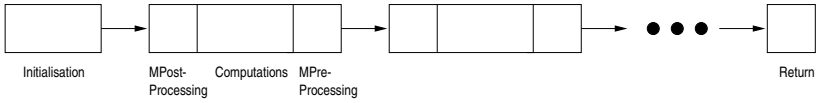


Figure 3.2: Lifetime of an Alliance member

- Similar to the last point, the Alliance computations should be protected by ensuring the integrity of a migrating agent data. This can be done by using SMPC to generate new distributed data at a migration.

- There should be a maximum time for which an Alliance member can be executed on one specific host. Otherwise, an agent could be maliciously hindered from migration, while the host waits for the moment, that more than $t(n)$ agents are hosted by co-operating malicious hosts. Suitable methods are required to ensure that a migration can be enforced and a host can be removed from the distributed Alliance computation.

In general, it will be difficult for a user to determine the degree of hostility of the network, especially when assuming the Internet. The authentication requirement may prevent some people from fraud. But although being able to detect incorrect data, the parties of an n-party protocol cannot determine which party was responsible. In the best case, they can determine a subset of two parties in which at least one party behaved maliciously. This is not sufficient for a legal prosecution. Most probably, this measure will rather keep those parties from fraud that are already tending to honesty than those having serious malicious intentions. Nevertheless, an authenticated network can be considered an advantage for secure multi-agent computations. For most applications the residual risk is tolerable, the more so as some of those applications significantly profit of the autonomous computations of an Alliance.
For the presentation of the protocol for secure multi-agent computation as well as its sub-protocols in chapter 5, it is therefore assumed, that $t(n)$ as a maximum for the number of corrupted agents is guaranteed as long as each server hosts at most one member of the same Alliance.

3.2 Lifetime of an Alliance

It has been mentioned in the introduction of agent Alliances that protocols for SMPC, as they assume n fixed parties, cannot handle mobility when being applied to multi-agent systems. Fortunately, one can divide the lifetime of an Alliance member into two kinds of phases: static and dynamic ones.
Dynamic phases consist of the actual data transfer of an agent migration. Static phases are defined by the absence of migrations. There are three types of static phases: Initialisation, task fulfilment and result return. The initialisation phase includes not only the generation of the Alliance but also the originators decision about where to send the agents initially. It can be considered as the "birth" of the Alliance. Result return consists of the agent to return to its originator and the reconstruction of the computation result. Most part of its lifetime the agent is in task fulfilment phases. Such a phase consists of three sub-phases, namely

1. *migration post-processing phase*
2. *computation phase*
3. *migration pre-processing phase*

Figure 3.2 illustrates the lifetime of an Alliance member structured by the single phases.
During a computation phase, the agent follows its agenda and takes part in the distributed computations of the Alliance, necessary to fulfil its task. This phase is completely secured by a protocol for SMPC.
The situation is different for the migration pre-processing phases. This phase is a mixed phase with respect to multi-party computations, as distributed computations take place as well as others. New hosts must be determined and potentially negotiated with. The latter requires the interaction of more than n parties and is thus not possible without leaving the current instance of the SMPC protocol. Other measures must be taken to protect this kind of computation. I will use majority decisions which are in the same complexity class as the SMPC protocol and, at the same time, do not require stronger assumptions on $t(n)$. The computation of new hosts can be done using SMPC. When having determined the set of new hosts, the Alliance distributedly computes a certificate stating the identities of the current as well as the future hosts. Distributed computations are also used for a necessary re-sharing, in which new data shares for each Alliance member are computed. The old shares are disabled automatically and cannot cause any harm if misused later-on. As a final step, the certificate, the agent and the shared data are sent to the new host. The end of the migration pre-processing phase means the end of the host's involvement in the SMAC. It is supposed to delete all knowledge (agent and data) connected to the Alliance and to have no further interaction with the remaining hosts in matters of the Alliance. Thereby, the static phase is completed.
The migration post-processing phase is for the agent initialisation on the new host. This requires the host to accept incoming data on the agent and its knowledge from hosts that are mentioned on the certificate, to reassemble that data and to start a new instance of the SMPC protocol.
Migration post-processing and migration pre-processing are phases requiring a lot of communication between the Alliance members and other servers. In order to maximise the efficiency of secure multi-agent computations, these phases should be minimised. This can be achieved by a suitable migration policy. In [EM03] we allowed the migration of single agents. In his diploma thesis [Mie03] Mie improves this, demanding several agents, may be even all of them, to migrate at once. As this significantly reduces the overhead costs, this measure should be taken, choosing a reasonable maximum for the allowed execution time on each host.
The life cycles of an Alliance member represented as a finite state machine is presented in figure 3.3.

3.3 Summary

Protocols for secure multi-party computation offer an opportunity to define a protocol for secure multi-agent computation which offers strong security guarantees. The advantage of this approach is the robustness of the distributed

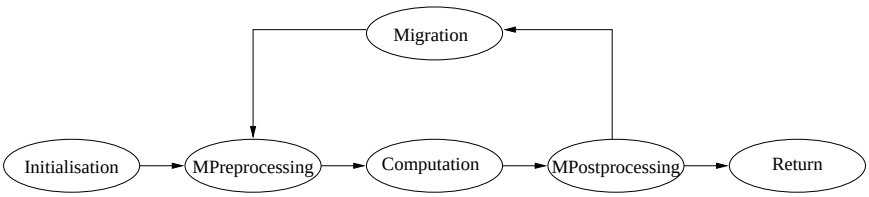


Figure 3.3: Lifetime of an Alliance member

computations. It is not necessary to know in advance which kind of attack one must protect against. In fact, attacks do not pose any problem as long as a specific majority of the agents is uncorrupted. As SMPC protocols are cryptographic protocols they provide security properties which are guaranteed as long as the specific protocol assumptions are met. This is fortunate, but aiming at a implementation in real-world systems, this must be investigated carefully. Some protocols require unrealistic assumptions a real network could not provide. Thus, it is not only of importance to take a close look at the overall complexity, but also at the protocol assumptions.
Mobility of agents is not covered by SMPC protocols. Therefore, it is essential to find suitable methods supporting this important ability. As in SMPC protocols, these methods should not require too strong assumptions. In the best case they work under the same assumptions as the chosen SMPC protocol. In addition, their complexity should lie in the same class as that of the SMPC protocol, with the constant factor as small as possible.
The methods I decided to use as well as the protocol I selected to implement secure multi-agent computation are presented in chapter 5, which is dedicated to the realisation of agent Alliances. But first, secure multi-party computation, along with its properties, potential and limitations will be presented in depth. This is the topic of the following chapter.

Chapter 4

Secure Multi-Party Computation

Secure multi-party computations allow a set of n fixed parties to evaluate a function with n inputs and n local outputs in a secure manner. The input is provided by n parties as they compute the n outputs. If those parties were to trust each other they could sent their inputs to one of the trusted parties in order to evaluate the function and finally send the outputs to the other parties. Secure multi-party computation aims on the emulation of the trusted party by the untrusted n parties themselves. Although indeterministic functions are possible, this presentation is limited to deterministic ones.
This research field is relatively young. Yao [Yao82] first mentioned a protocol for secure two-party computation in 1982. Since then, numerous protocols have been proposed, meanwhile being designed for n-party computations. For several years, all protocols assumed synchronous networks as they provide guaranteed message delivery within a certain time bound. Then, in the 1990s the first asynchronous protocols were published. While synchronous protocols are always based on secret sharing schemes [Sha79], asynchronous ones rely on several cryptographic primitives like homomorphic encryption, trapdoor functions etc. What all protocols have in common is the requirement to represent the function as an arithmetic circuit which is then somehow "commonly"[1] evaluated. The protocol execution itself always follows the Commit-Compute-Reveal (CCR) paradigm. This means

1. Each party commits itself to its input for the circuit.
2. The circuit is evaluated via distributed computations.
3. After circuit evaluation, the parties reveal their local results to each other or another pre-defined instance.

Circuit evaluation follows a specific protocol and demands the parties to perform local computations and to interact with the other parties by exchanging messages over the network.

[1] Depending on the protocol, it is for example possible that all parties evaluate the same circuit in a synchronous fashion. An efficient asynchronous method is to divide the circuit in several parts that are evaluated by subsets of the parties.

Multi-party computation uses a lot of cryptographic sub-protocols. Besides zero-knowledge proofs [GMR89, GMW86], probabilistic encryption [GM84], data representation with error correcting codes [BGW88] and oblivious transfer [Rab81, Kil88, CK88], secret sharing schemes and computing with suchlike shared data are fundamental in order to achieve SMPC (also discussed in [Gol97]). To describe all these cryptographic primitives goes beyond the scope of this thesis. For the implementation of agent Alliance, one specific protocol for SMPC [HM01] has been selected, providing the lowest communication complexity of all protocols currently known. Protocols with high communication complexity are contradicting with the autonomous agent paradigm. Therefore, the protocol of Hirt and Maurer is a good choice for an implementation in such a scenario. It is a synchronous protocol which causes the presentation in this chapter to focus on a detailed introduction of the (verifiable) secret sharing primitive.
This chapter is organised as follows: First, possible protocol assumptions are introduced and explained. These are for example, the adversary and the network models, the underlying security model and the type of security that is finally reached. After these general remarks, section 4.2 provides an introduction into early secret sharing approaches, that finally led to modern verifiable secret sharing (VSS) schemes. The chapter ends with the presentation of the protocol for secure multi-party computation from Hirt and Maurer [HM01] which is then used for the protocol for secure multi-agent computation presented in chapter 5.

4.1 How to Define a Security Model

The security of cryptographic protocols is always proved with respect to some protocol assumptions as well as the properties the protocol should provide. A proof of security does only provide the guarantee that in case the protocol assumptions are kept during the protocol execution, the functionality of the protocol (and with it, the protocol properties) is given. Therefore, a protocol that has been proven to be secure under specific assumptions, may suddenly became a threat when making the slightest change in these assumptions. This if an important aspect for the secure implementation of cryptographic protocols. Therefore, this section provides an overview on the most important definitions that come into play as soon as on speaks about *protocol security*.
Security proofs are not discussed in this thesis. It shall only be mentioned here, that the most important approach uses a simulator using a so-called *ideal functionality* that works within an environment. This ideal functionality represents a trusted authority receiving the input of all parties, executing the protocol, and finally delivering the output. The environment represents the adversarial entity. It is responsible for the message exchange between the parties of the protocol. As such it can delay or even delete messages. But it cannot gain information about the contents of the messages, neither does it know which input the parties provide to the protocol, nor the protocol output. If one can show that a protocol run in the real-life model cannot be differentiated from a protocol run within the simulator, the protocol is proven to be secure. This field of security proofs requires very deep theoretical knowledge on the different security models and goes beyond the scope of this thesis.

4.1.1 Protocol Properties

As mentioned above, the definition of a protocol requires to fix the properties, the protocol should achieve. These properties allow to introduce the notion of security as presented in section 4.1.5. The presentation here is limited to the most important possible properties, especially those, that are finally provided by the protocol for secure multi-agent computation in chapter 5.

- *Robustness:* A protocol is called t-robust if it guarantees correct results even if an adversary has corrupted up to t parties.
- *Fairness:* A protocol is called fair, if a malicious party leaving the protocol prematurely cannot gain any information on the computation result. In addition, it must be guaranteed, that the remaining parties are able to complete the protocol and to reconstruct the correct result.
- *Privacy:* Given an adversary, that corrupted at most t parties, the computations and intermediate results remain secret at any time during the computation phase.

4.1.2 Attacker Models

As already mentioned, secure multi-party computation aim at the robust evaluation of a function through n parties which are not trusted. This implies, that the correctness of the distributed computations must still be guaranteed if some of the parties behave arbitrarily instead of following the protocol. In more abstract terms, this means that the computations must be robust against an adversary controling up to $t(n)$ parties. This adversary is called the *super-adversary*. It is illustrated in figure 4.1.
This abstraction causes no loss of generality as:

- In worst case, all malicious hosts co-operate and can thus be understood as one adversary.
- Any set of separately working adversaries cannot cause more damage to or gain more information about the Alliance as a whole than the super-adversary.

Once the adversary has corrupted a party, it gets to know all input and output as well a the whole history of the party. In case messages were sent encrypted, the adversary gets to know the clear-text. In addition, the adversary fully controls the behaviour of the corrupted party. It can read all messages which will be sent to the party in the future and send messages in the party's name. The situation is similar to a complete brainwashing of the party.
Now, we discuss the different properties an attacker may have. The presentation is limited to the most important attacker types, but it be explicitly pointed out that there are many other facets an attacker. A more precise notation is given in [Gol97].
An adversary is called *passive* if it looks honest to others. This requires it to follow the protocol. However, a passive adversary can spy out data and do computations beneath the actual protocol. It is also possible that it does not use real random when necessary, hoping to gain an advantage. Because of this,

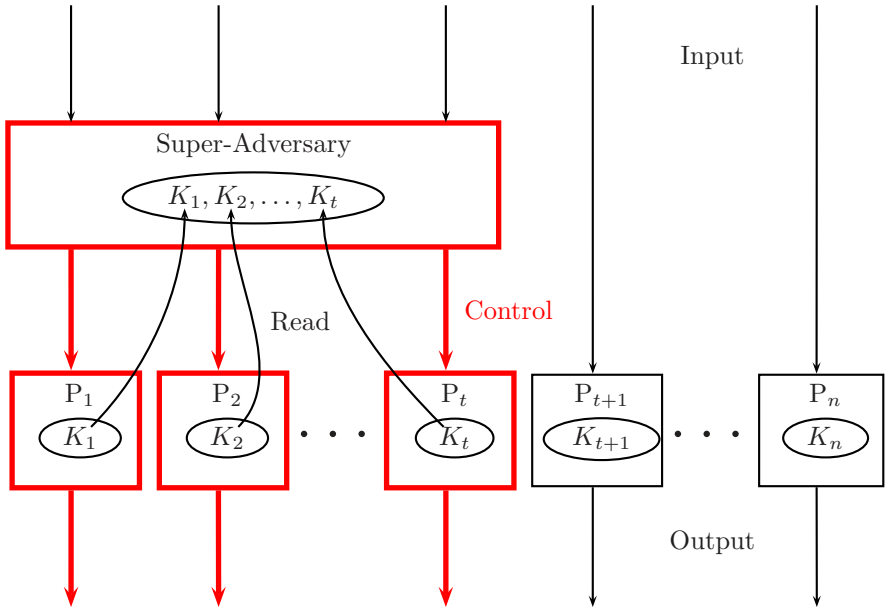


Figure 4.1: Model of the super-adversary

such an adversary is also called *malicious-but-honest-looking*. Passive adversaries are relatively simple to deal with as they usually cannot influence the protocol significantly. In contrast, an *active* adversary is openly malicious and acts as it thinks is beneficial for it. The behaviour of such an attacker must be regarded as arbitrarily or, in other words, *Byzantine*. The output of parties that are corrupted by such an adversary, is not reliable in any way. Because of the variety of possible fraud, an active adversary is difficult to prevent from.
The strength the attacker is provided with determines whether and how fast it can corrupt any party. There are mainly two different types: an attacker can be computationally unbounded or polynomially bounded. The first attacker model is mainly of interest for theoretical considerations as any real attacker will not have indefinite resources and capabilities. In case a protocol is secure against an unbounded adversary, it provides information theoretic security. A polynomially bounded attacker has limited computational power. This means, by doing polynomially bounded computations during the protocol execution it does not get more information than it does through the protocol execution itself. This type of adversary is the realistic one. In the field of secure multi-party computation both attacker types are known and used.
Not only the computational power of an adversary can be used for a classification, but also at which time the adversary has to decide, which party it will attack. There are three types:

1. A *static* adversary has to determine its victims before the protocol execution starts. This does not mean that each of these parties are corrupted from the beginning. The time of the attack is not fixed. But sometime during the protocol execution it will take place. The static adversary must

make its decision without further knowledge which it might get during and, especially, through the protocol execution. This type of adversary is the one that is the easiest to fight against[2].

2. In contrast to the static adversary, an *adaptive* one may use information it collected during protocol execution, before to decide which parties it will attack. This kind of adversary is mostly assumed in protocols for SMPC.

3. The most dangerous adversary is a *mobile* one. It can corrupt parties at any time it wishes. So much the worse as this adversary can release parties and corrupt others. The only restriction is, that the number of corrupted parties is limited by $t(n)$ at any point in time. Such an attacker is able to collect data of more than $t(n)$ parties, although controlling the behaviour of at most $t(n)$.

Protocols for secure multi-party computation get by with modelling an adaptive adversary. The concept of mobile adversaries is quite new and became necessary with the propagation of mobile systems like mobile phones and PDAs. These adversaries can only be circumvented by re-sharing techniques, like the one presented in [OY91], which disable old data shares and generate new ones. Re-sharing again, is a $t(n)$-robust distributed computation. Although secure multi-agent computation will be based on secure multi-party computation, the adversary could be able to collect information from more than $t(n)$ agents of one Alliance just by "hopping" from one set of up to $t(n)$ corrupted parties to another by means of migration. This is where the mobile adversary comes into play. Consequently, re-sharing is inevitable in SMAC and must be part of the migration pre-processing phases. A concrete protocol is presented in depth in chapter 5.

4.1.3 Error Behaviour

Abstractly spoken, error behaviour is one aspect of the adversary behaviour as errors can be assigned to the adversary. It defines the way an adversary influences a protocol, the methods it can use. A categorisation is done by [HH91], where crash failures, sending omission, receiving omission, general omission failures, and, finally, arbitrary failures are differentiated. In case errors occur arbitrarily, the literature speaks of *Byzantine errors* (see for example [PBG89, CR93]. The SMAC model in chapter 5 uses a Byzantine error model as most of today's cryptographic protocols do.

4.1.4 Network Type

All cryptographic protocols are based on specific network assumptions. These assumptions are mostly idealistic and not available in practice. If the latter is the case, a suitable protocol must be used to simulate the wished network behaviour. In this paragraph, first the physical properties are introduced, before a short discussion about the most important cryptographic network primitives is given.

[2]Which does not mean that it is simple!

Synchronous/Asynchronous

One possibility to categorise networks is depending on whether and when messages are delivered. There, two networks can be differentiated: asynchronous and synchronous ones. Synchronous networks are networks in which all system clocks are synchronised, i.e. run at identical rates. In addition, there is a guarantee, that messages are delivered within a certain time bound. With respect to distributed computations, this is the most prefered network model, as it significantly eases the message exchange involved in these protocols. Also the design of cryptographic multi-party protocols is much simpler if one can assume that messages of honest parties really arrive. As often, this idealised model cannot be used in practice. Although there are efforts underway to create real synchronous networks, all existing open networks that are of interest for practice (like e.g. the Internet) are truly asynchronous. In an asynchronous network, the single parties cannot assume when and even whether at all their messages are delivered. As a consequence, in multi-party computations it cannot be differentiated whether the input of a party is just delayed or if the party got corrupted and did not send anything at all. Also the system clocks deviate from each other which makes it difficult to determine whether a message is already delayed or not. Security proofs of cryptographic protocols which are based on asynchronous networks are feasible but require strong additional assumptions as for example a weaker definition of security.

Connectivity

Another way to look at a network is given by the way, the users are connected to each other. Here, too, ideal models are given, which have to be simulated in practice. Although other types are possible, the presentation is limited to the ones, necessary for secure multi-party computation: point-to-point connections and broadcast channels. The first assumes each pair of parties to be directly connected. In order to send identical messages to n receivers, n channels are used. In contrast to this, a broadcast channel allows parties to send a message to all other parties at once. Such a channel guarantees that each receiver receives the same message. This guarantee is not given if multiple point-to-point connections are used. Somewhere, all protocols for general secure multi-party computation make use of a broadcast channel. In practice, such channels are mostly not available and must therefore be simulated.

Cryptographic Network Primitives

It is mentioned above, that the categorisation of network types is done using theoretical models. The technical equipment necessary to physically implement these models is mostly not available. This is why one part of the wide field of cryptography is devoted to the simulation of these models, for example by developing cryptographic protocols for broadcast simulation. These protocols can be collectively refered to as *cryptographic network primitives*. I will shortly explain three of them, all of them important for the security of the Alliance model.

- *Authenticated channels:* In the authenticated channel model, a unique identifier is assigned to each party (i.e. user or server). This identifier

is generated and certified by a trusted authority. Before starting to communicate, the respective parties exchange their certificates in order to determine the identity of the communication partner. All messages sent by a party are signed[3] with its identifier.

Authenticated channels provide a level of security that is often sufficient for the prevention from message manipulations or spoofing. In real applications, the loss of anonymity alone in suchlike protected networks makes many users to behave in compliance with the protocol. As such, in practice this model oftenly offers more security as one could expect from the theoretical point of view.

- *Secure channels:* The secure channel model is not clearly defined. Some sources do not differentiate authenticated channels and secure channels, which means, that there is only authentication. In contrast to this, it is most common to add another property, namely data privacy, when talking about secure channels. This means, that after the mutual authentication via certificate, a so-called *session key* is generated[4] and used to encrypt all data sent from one party to the other.

- *Broadcast channels:* Primitives for broadcast channels aim at the simulation of physical broadcast channels. Although the problem is very old and related to so-called *Byzantine agreements* (see e.g. [LSP82]) it is still challenging. Research provided us with several cryptographic protocols that are more or less (mostly less) efficient. The best ones lie in $O(n^2)$ (for n the number of parties) and it seems as if this a lower bound. However, this primitive is very important for secure multi-party computation and agent Alliances as well. In fact, the complexity of these protocols currently represents a lower bound for all SMPC protocols. The interested reader may take a look at our current work from [Fre07] presenting an efficient asynchronous protocol for Byzantine Agreement.

4.1.5 The Notion of Security

After having fixed all assumptions, a cryptographic protocol must provide a reliable statement on how secure it is if all assumptions are kept at any time. Cryptographers speak of perfect, statistical and computational security. The differences between perfect and statistical security is not of interest when looking at the respective adversary model. The difference between both concepts is that there is a small probability of error in case of statistically secure protocols while perfectly secure protocols do not have an error probability. The term *unconditional* security contains both, perfect and statistical security, and is used from now-on.
A protocol provides unconditional security, when under the given protocol assumptions and in presence of a computationally unbounded adversary all protocol properties are guaranteed. This is the strongest type of security, similar to information-theoretical secure encryption. As soon as cryptographic primitives are used, a protocol cannot be unconditionally secure. These primitives are based on the difficulty of specific mathematical problems like factorisation of

[3]For details on digital signatures, the interested reader be refered to appendix A.3
[4]This is done with a protocol for secure key exchange, like the one of Diffie and Hellman.

large integer numbers. Choosing suitable instances of these mathematical problems, an attacker is confronted with a large key space. The only possibility to gain information, is to use the most efficient existing algorithms which compute solutions to the given problem. Although these algorithms have at least subexponential complexity, an unbounded adversary would be able to find the key within short time. Therefore, the strength of the attacker in a cryptographic setting must be limited to a polynomial bound. The resulting security is called to be *computationally*. This type of security is, although much weaker than unconditional security, absolutely sufficient for practice. Computationally secure protocols depend on a so-called security parameter, mostly called κ, that is normally determined by the size of cryptographic keys which are used in the protocol.
Protocols for secure multi-party computation exist for both types of security. And, while secure multi-agent computation as presented in chapter 5 is based on an unconditionally secure protocol for SMPC [HM01], it makes heavy use of additional cryptographic primitives and thus provides cryptographic security.

4.2 Secret Sharing

Secure multi-party computation offers the possibility to compute with private data. Consequently, the method used to distribute data must not only guarantee privacy but also allow to perform computations on these shared data. To be able to prove that such computations are secure, requires the term "computation" to be defined more clearly. As such, all protocols for SMPC presume the function that is to be evaluated is given as an arithmetic circuit over a finite field $\mathbb{F}$. This implies, that the possible operations are limited to addition and multiplication of shares as well as scalar multiplication of a share with a public value over $\mathbb{F}$.
In synchronous protocols for SMPC data distribution is done with so-called secret sharing schemes, namely the one of Shamir from [Sha79]. This section continues with a bottom-up presentation of the evolution of secret sharing. It starts with Shamir's approach which provides no robustness as it only presumes a passive adversary. Then, successively the definition is extended, finally leading to verifiable secret sharing schemes.

4.2.1 Unverified Secret Sharing

Definition 4.1. (t, n)*-Secret Sharing*
A (t, n)*-secret sharing protocol is a sequence* $(\mathcal{P}_1, \mathcal{P}_2)$ *of two* n*-party protocols. In* $\mathcal{P}_1$ *a party* p_0*, called the* dealer*, distributes shares of his private input* s *in a way that during the execution of* $\mathcal{P}_1$ *no subset of up to* t *parties (not including* p_0*) learns any information about* s*.* $\mathcal{P}_2$ *is a protocol for reconstruction* s *from at least* $t+1$ *shares. In case* p_0 *has not been malicious during* $\mathcal{P}_1$ *and all inputs for* $\mathcal{P}_2$ *are correct, the value obtained in* $\mathcal{P}_2$ *is the same as the original input* s*.*

Independently, Shamir [Sha79] and Blakely [Bla79] proposed in 1979 the first secret-sharing schemes as threshold access structures. Both approaches assume the passive adversary model. The one of Shamir is very intuitive and efficient, and as such the most popular one. The scheme is based on the interpolation of univariate polynomials and is a corner stone in secure multi-party computation. Therefore, Shamir's secret sharing scheme is now presented in detail.

Shamir's Secret Sharing Scheme The secret sharing scheme proposed by Shamir makes use of the following proposition:

Proposition 4.2. *For a given set of $t+1$ arbitrary data points*

$$(x_i, y_i) \in \mathbb{F}^2,\ i = 0, \ldots, t$$

with $x_i \neq x_k$ for $i \neq k$, exists a unique polynomial $f(X) \in \mathbb{F}[X]$ of degree t such that

$$f(x_i) = y_i \text{ for } i = 0, \ldots, t.$$

Proof. 1. Uniqueness:

Suppose two polynomials $f_1, f_2 \in \mathbb{F}[X]$ of degree t with

$$f_1[x_i] = f_2[x_i] = y_i,\ i = 0, \ldots, t.$$

The difference $f := f_1 - f_2 \in \mathbb{F}[X]$ is a polynomial of degree less or equal to t and at with at least $t+1$ different zeros, namely $x_0, \ldots, x_t$. This is a contradiction.

2. Existence:

Construction with Lagrange polynomials $L_i \in \mathbb{F}[X],\ i = 0, \ldots, t$:

$$L_i(X) = \frac{(X - x_0) \cdots (X - x_{i-1}(X - x_{i=1}) \cdots (X - x_t)}{(x_i - x_0) \cdots (x_i - x_{i-1})(x_i - x_{i+1} \cdots (x_i - x_t)} \tag{4.1}$$

obviously fulfils the equation

$$L_i(x_k) = \delta_{ik} \quad \text{(Kronecker symbol)}.$$

Via the L_i a solution f for the interpolation problem is directly given by the so-called *Lagrange interpolation formula:*

$$f(X) = \sum_{i=0}^{t} y_i L_i(X) \tag{4.2}$$

□

Although interpolating a polynomial with the Lagrange formula generally causes $O(t^2)$ multiplications and $O(n)$ divisions, this scheme offers an advantage in the field of secret sharing: In case the parties reuse their public value α_i the denominator of the Lagrange polynomial can be pre-computed once and reused for different secret sharing polynomials. In addition, if only an interpolation of one specific point $f(\alpha)$ is required, the nominator can be pre-computed, too. In the following protocol we will see that Shamir's secret sharing approach always uses the constant term $f(0)$ to encode a secret, and thus the interpolation complexity can be decreased to $O(n)$ multiplications (with one-time costs for the pre-computations in $O(n^2)$).

$\mathcal{P}_1$: Secret Sharing
Let $p_1, \ldots, p_n$ be the n parties that are supposed to share a secret $s \in \mathbb{F}$ and $\mathbb{F}$ be a finite field with $|\mathbb{F}| > n$. A public value $\alpha_i \in \mathbb{F}$ is assigned to each party

P_i. The dealer p_0 (depending on the application the dealer can be one of the parties) encodes s by generating a random polynomial $f[X] \in \mathbb{F}[X]$ of degree t with $f(0) = s$. In order to t-share the secret s among all parties he sends the share $y_i := f(\alpha_i)$ to P_i for all i.
If less than $k+1$ data points are available, no information about f can be gained. In fact, the missing nodes can be arbitrary elements of $\mathbb{F}$. A brute force attack would have to check $|\mathbb{F}|^i$ combinations of field elements if i nodes are missing. This estimation also shows that the size of $\mathbb{F}$ should be large enough to prevent from brute force attacks. The size of the secret s provides a lower bound for $|\mathbb{F}|$ as it is one of its elements and, in general, one can expect s to be sufficiently large. However, depending on the application, s may be binary. In any case, the dealer must take care of selecting a field with suitable size allowing to hide the secret.

$\mathcal{P}_2$: Secret Reconstruction:
In order to reconstruct s, the use of formula 4.2.1 with the above discussed pre-computation seems to be the most efficient approach.

$$s = f(0) = \sum_{i=0}^{t} y_i L_i(0) \tag{4.3}$$

In case p_0 has not been malicious during $\mathcal{P}_1$, the value obtained in $\mathcal{P}_2$ is the same as the original input s. It should be mentioned here, that Shamir's secret sharing does not allow t-private computations on shares. Ben-Or et al. presented in [BGW88] a method using [Sha79] as building block and allowing n parties to perform t-private computations on shares. A simplification can be found in [GRR98]. Although this is an important step in the direction of secure multi-party computation, the approaches are meant for passive adversaries only. Considering parties that are spread over the Internet Byzantine faults should be considered. Therefore, an improvement of the non-robust secret sharing has been defined (see e.g. [DDWY93]) offering not only privacy but also robustness.

Definition 4.3. (t, σ, n)*-Secret Sharing*
A (t, σ, n)*-Secret Sharing scheme is similar to the* (t, n)*-Secret Sharing from definition 4.1. But in this case the two protocols* $\mathcal{P}_1, \mathcal{P}_2$ *are able to tolerate up to* σ *Byzantine faults.*

A (t, σ, n)-secret sharing offers an n-party reconstruction protocol $\mathcal{P}_2$ which is robust against up to σ malicious inputs. As long as this limit is kept throughout its execution and the dealer p_0 remains non-faulty during $\mathcal{P}_1$, applying $\mathcal{P}_2$ results in the reconstruction of the original input s.
In order to introduce the definition of a verifiable secret sharing which is used for the secure multi-agent computation presented in chapter 5, we first have to include an attacker model. Assume $\mathcal{A}_L$ a passive adversary compromising a set L of listening parties. $\mathcal{A}_L$ knows the complete history of all parties in L. $\mathcal{A}_D$ be an active adversary corrupting a set D of disrupting parties.

Definition 4.4. (t, σ, n)*-Unverified Secret Sharing*
Assume L *to be the set of* t *listening parties (out of* $n-1$*) and* D *the subset of* L *of size* σ *that is disrupting the computation. If* p_0 *has an arbitrary input* $s \in \mathbb{F}$ *and remains non-faulty throughout* $\mathcal{P}_1$ *we require*

1. $\forall s' \in \mathbb{F}$: *If* $p_0 \notin L$ *then the probability distribution on the views of* $\mathcal{A}_L$*, given that* p_0 *has input* s*, is identical to the probability distribution on the views of* $\mathcal{A}_L$*, given that* p_0 *has input* s'*.*

2. *At the end of* $\mathcal{P}_2$ *every honest party outputs* s*.*

Property 1 provides the secrecy of input s against t listening parties, while property 2 offers resilience against σ disrupting parties. The problem of inconsistent input in $\mathcal{P}_2$ can be solved by aid of an error-correcting code as for example presented in [BW86]. Definition 4.4 does not handle the case of p_0 being faulty during the execution of $\mathcal{P}_1$. There is no possibility for the parties to check the consistency of the shares they are receiving. Consequently, in a t-unverified secret sharing scheme $\mathcal{P}_2$ will fail if p_0 behaves maliciously in $\mathcal{P}_1$. This is most unwanted for multi-party computation, where, first, the n parties are untrusted Internet users, and second, the dealer generally is one of them. Thus, there is a considerable probability of being concerned with a malicious dealer. This is why the concept of a *t-verified secret sharing* has been introduced. For simplification, most oftenly the secrecy and the resilience parameter are set to the same value and one simply writes t-verified secret sharing. The main improvement of t-verified secret sharing schemes is, that they offer additional correctness constraints for the case p_0 is faulty during execution of $\mathcal{P}_1$.

4.2.2 Verifiable Secret Sharing

As mentioned before, verifiable secret sharing is an essential component of any secure multi-party protocol. So far, the definitions do not include two significant properties, that should be available for SMPC: The possibility

1. to detect a faulty dealer and

2. to perform t-private and t-robust computations on shares.

The first is covered by the following definition 4.5 while the latter is an additional property, not all verifiable secret sharing (VSS) schemes offer.

Definition 4.5. *t-Verifiable Secret Sharing*
A secret sharing as given in definition 4.4 is called a t-verified secret sharing *if the outcome of* $\mathcal{P}_2$ *is uniquely determined by any subset of* $n - t$ *parties that are honest at the end of* $\mathcal{P}_1$ *and as long as at most* t *parties become corrupted while executing* $\mathcal{P}_2$*.*

Similar to Shamir's secret sharing in the presence of passive adversaries, Chor et al. [CGMA85] can be considered as a milestone for the active adversary model. Since then lots of approaches have been published, which can be categorised by two different criteria: by the kind of security they provide (unconditional or computational), by the assumptions they require (broadcast channel, secure channels etc.) and last but not least by the type of the underlying network (synchronous, asynchronous).
When this research field started to become popular in the late 1980th, most approaches assumed synchronous networks in order to have guaranteed message delivery. Alternatives like using perfectly secure message transmission [DW02, BM03, DDWY93] in asynchronous networks was and still is not practical. And,

in addition, protocol design in synchronous networks is much easier. Meanwhile, there are asynchronous (e.g [HNP05]) solutions, but their complexity exceeds that of synchronous solutions by a factor of at least n (if n parties are involved) in the best case.
The remainder of this section is devoted to an overview on existing VSS protocols. Since asynchronous approaches are not practical, the exemplary protocols below are without exception synchronous protocols. They are classified after the type of security they provide.

Protocols with Unconditional Security:

Protocols providing unconditional security do not require any limitation of the computational power of the parties and the adversary. Under this assumption, for a piece of data unknown to a set of parties unconditional security means not only that they cannot compute it within a certain time bound from what they know, but simply that it cannot be computed at all. Protocols offering unconditional security are secure in the information theoretic sense. They do not depend on any assumption about computational intractability. As such they are stronger, but at the same time they offer resilience against less malicious parties (normally $n/3$) than synchronous ones (normally $n/2$). VSS protocols with unconditional security can be classified into perfect and statistical ones. Statistical security means that there is a (normally very small) probability of error. If such an error occurs, the protocol fails. An example is the protocol of Rabin and Ben-Or [RBO89] which allows to share secrets as long as a majority of $n/2+1$ parties are honest. This is achieved by requiring broadcast channels and secure channels between every two parties. The probability of error is exponentially small. In [CCD88] Chaum et al. present an unconditional but statistical protocol. It uses authenticated channels and the Byzantine agreement from [DS82]. Error-correction is achieved by a scheme of zero-knowledge proofs which causes a round complexity of $O(\log n)$ for the multiplication of two n bit numbers. In contrast the approach of [BGW88] relies on arithmetic operations in large finite fields and provides perfect security, which means that no probability of error is allowed. In addition, multiplication of two n bit numbers requires $O(1)$ rounds. The round complexity of interactive protocols is one of the most important complexity measures. Therefore, not only the specific protocols take a look at this number, but there are also publications that investigate the round complexity protocol-independent but security model-depended. As such, Gennaro et al. [GIKR01] have proven some limitations in the unconditional setting. Here, it be only mentioned that a 3-round VSS is possible iff $n > 3t$, where the "if" direction is realised by an inefficient protocol. Whereas, for the same t an efficient VSS is possible with 4 rounds.

Protocols with Computational Security:

Cryptographic results are stronger than unconditional ones in terms of resilience. VSS based on cryptographic assumptions offers resistance against $t < n/2$ unreliable parties. Adversaries in this model are considered to be computationally bounded. Diffie and Hellman first introduced this security model in [DH76] and their approach can still be considered as a building block for VSS. Till 1988 all

published protocols provided computational security. Similar to the reasons of designing synchronous protocols, it is intuitive that cryptographic approaches are easier to achieve than unconditional ones. And, since the security level of cryptographic protocols is in general more than sufficient, the existence of such approaches is justified. Problems that are based on [DH76] rely on the difficulty of computing discrete logarithms. Other approaches like e.g. the one of Goldreich, Micali and Wigderson in [GMW87] prove the existence of secure trapdoor functions and show that every function with n inputs can be computed such that in case Byzantine faults occur, no set of parties of size $t < n/2$ can either disrupt the computation or compute additional information. This group has also proven the existence of zero-knowledge proofs for NP-complete problems, which is used in [GMR89]. In the cryptographic model the protocol of Cramer et al. from [CDN01] is the most efficient one with a communication complexity of $O(n^3)$, but $O(n)$ rounds per multiplication. This results from the fact that the evaluation of each multiplication gate requires the invocation of a broadcast sub-protocol. For the cryptographic setting, too, Gennaro et al. [GIKR01] have given an upper limit for the round complexity: With a linear threshold of cryptographic security, any function can be computed efficiently in its circuit size, requiring 3 rounds per multiplication gate.

4.2.3 Error Correction on Shares

A verifiable secret sharing scheme must provide specific methods to detect and correct a specific number k of incorrect shares. In [BGW88] a generalised Reed-Miller code (see [PW72]) code is used. Unfortunately, this implies the necessity of choosing the interpolation points as powers of a primitive *n-th* root of unity in the underlying finite field $\mathbb{F}$. Although the protocol of Ben-Or et al. will play an important role later-on in section 4.3.2 for the secure multi-party computation protocol, the method from Berlekamp and Welch [BW86] is proposed as substitute for the Reed-Miller code. The approach from [BW86] does not require the use of roots of unity, causes a computational complexity of $O(t \cdot n)$ and corrects up to $k < (n - t)/2$ errors.

4.3 Secure Multi-Party Computation

4.3.1 Overview

Secure multi-party computations is nothing else than computing on shared data. As mentioned in the beginning of this chapter, all SMPC protocols consist of three phases: Commit, compute and reveal. In the commitment phase every party commits to its input in a way that the other parties do not gain any information about this input. In synchronous SMPC protocols this is done with a verifiable secret sharing scheme. These schemes handle the problems arising with inconsistent shares and missing inputs. Such data are substituted by default shares and do not cause harm to the SMPC protocol as long as not more than $t(n)$ inputs are faulty. As such, the commitment phase is along the lines of a verifiable secret sharing protocol. However, the computation phase requires arithmetic operations on shared data, particularly addition and multiplication. Multiplication of t-shared data using the homomorphic properties of polynomi-

als, results in $2t$-shared output. Consequently, the degree of the sharings must be securely decreased afterwards. Till now, all approaches to SMPC require interaction between the parties to solve this problem. Unfortunately, interaction means communication, which is not only costly and due to network delays inefficient, but also causes a strong dependency between the capabilities to defend against attackers and the properties of the underlying network. Therefore, the further survey on protocols for secure multi-party protocols will be structured after the different network types.

In synchronous networks with public channels it is possible to defend against static and polynomially bounded adversaries corrupting at most $t < n/2$ parties. This has been shown in the original paper of Yao [Yao82] as well as by Goldreich et al. [GMW87, Gol04] under the assumption that there are trapdoor permutations. Softening the notion of security by assuming that an abortion of the protocol does not hurt security, even allows an adversary to corrupt an unbounded number of parties while guaranteeing the correctness of the output, if produced. This result is presented in [Gol04].

In contrast, secure point-to-point connections in a synchronous network only allow to defend against arbitrary adaptive adversaries corrupting at most $n/3$ of the parties. I have already mentioned this result when discussing the protocol of Ben-Or et al. [BGW88], but it is also given in [CCD88]. Both protocols rely on broadcast simulation. If one can additionally assume that global physical and authenticated broadcast channels are available, Rabin et al. [RBO89], Beaver [Bea91b] as well as Chaum et al. [CCD88] have presented protocols that are secure against computationally unbounded adaptive t-adversaries with $t < n/2$. This result has been improved in [FM00] where it is shown that a broadcast channel between each subset of three parties is sufficient to reach these properties.

The situation is more difficult in networks with point-to-point connections in which the adversary can delete messages (asynchronous networks). While in case of finite time message delays the parties could wait for input, in case of deleted messages the parties cannot determine whether the sending party got corrupted or is just delayed. This implies, that it is unclear how many honest parties still have to send their input before the computations can be continued. As it is necessary to have at least $3t + 1$ inputs available if t parties got corrupted, the input of all honest parties must arrive. Otherwise, the protocol fails. Ben-Or et al. [BOCW93] limited the adversary to corrupt at most $t(n) < n/4$ parties to be able to additionally face a certain amount of network errors. If one does not require error-free protocols but allows an exponentially small probability of error, Ben-Or et al. show in [BOKR94] that also $t < n/3$ is possible.

The practicability of SMPC protocols strongly depends on the given computation and communication complexity. Nowadays, mostly the later is the most limiting factor. After the publication of the first protocols in the 1980s, which focused on the presentation of the feasibility of SMPC and did not look at efficiency, subsequent protocols tried to reduce the number of rounds as well as size and number of message exchanges. In the 1990th these efforts focused on synchronous networks only [BBB89, BFKR90, FKN94, FY92, GRR98], as the guaranteed message delivery in such networks simplified the design and correctness proofs of the protocols. The approaches from [BBB89, BFKR90, GRR98] aimed on a reduction of the round complexity but had to deal with a high communication complexity and a small robustness parameter $t(n)$. In [FY92, GRR98] the

authors show that the situation can be improved if the computations are firstly done in a non-robust but private manner. Only in case of faults the computations are repeated using an inefficient but robust protocol. Although, at first look, this seems to be a promising idea, one must finally recognise that such an approach opens the door for an adversary to constantly slow down the protocol by manipulating just a single party. We will see later-on in this section, that Hirt and Maurer used this idea in [HM01], too, but limited to the preparation phase of the protocol. Additionally, it is possible to detect and eliminate players that have behaved maliciously. Over the time, this decreases the probability of needing the more costly protocol. As such, this protocol, although already published in 2001, is still the most efficient one. It causes a communication complexity of $O(mn^2)$ for the common evaluation of a circuit with m multiplication gates by n parties.
This protocol is used as a sub-protocol for the distributed computations of an agent Alliance (as introduced in chapter 3). It is presented in the remainder of this chapter.

4.3.2 The Protocol of Hirt and Maurer

Although originating in 2001, the protocol of Hirt and Maurer [HM01] is still the only practical SMPC protocol that has been published till now. Therefore, the agent Alliances are currently designed using this protocol. But, by their design, Alliances could use other SMPC protocols as well. Since the understanding of an Alliance requires detailed information about how it performs its distributed computations, this section is dedicated to a survey of the protocol.

Assumptions Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the set of n players connected by bilateral synchronous reliable secure channels. The members of $\mathcal{P}$ wish to correctly compute an agreed function without demanding each player to be honest. The function is specified as an algebraic circuit over a finite field $\mathbb{F}$ ($|\mathbb{F}| > n$) with m multiplication gates. To each player P_i a unique public value α_i is assigned (his zero for the secret sharing scheme of Shamir [Sha79] which the protocol is based on). The protocol allows a secure function evaluation although an active and computationally unbounded adversary may corrupt up to $t < n/3$ players, which means that the computed result is correct although t players can (mis)behave arbitrarily. The security of the protocol is unconditional with an arbitrarily small probability of error. Later-on, we will see that in practical use, the main problem is to keep the number of corrupted players within the upper limit t. As long as this upper bound is not exceeded, the security of the protocol is perfect.
The protocol is divided into two phases. Initially, a preparation phase takes place, in which m random triples (a, b, c) with $c = a \cdot b$ are t-shared among the players. In this phase, the players generate and distribute random numbers $a, b \in \mathbb{F}_p$. Then, a t-shared product is computed. This is done along the lines of [BGW88] (see section C) and [GRR98]. These protocols are non-robust which is why after the triple generation, a fault detection and localisation takes place. Malicious parties get excluded from the preparation phase as soon as they are detected. The second phase of the protocol is the computation phase in which the players provide their shared input to the circuit, which is then evaluated gate by gate, where for each multiplication gate one random triple is used for

Preparation Phase	
1	triple generation
1.1	computation of t'-shared random numbers $a, b \in \mathbb{F}$
1.2	computation of t'-shared $c = a \cdot b$
1.3	increase degree t' to t
2	verification of the degree
2.1	fault detection I
2.2	fault localisation I
3.	verification of sharing
3.1	fault detection II
3.2	fault localisation II
Computation Phase	
1	input sharing
2	computation phase
3	output reconstruction

Table 4.1: Protocol phases

means of blinding. The phase ends with the output reconstruction. Table 4.1 offers a detailed survey on the contents of each phase.
In a mobile agent scenario the preparation phase is not required since the originator of an agent Alliance could securely produce and distribute random triples before sending his Alliance in an un-trusted environment. Nevertheless, for some tasks it may be complicated or impossible to know an upper bound of multiplications in advance so that it could be useful to provide an Alliance with the property to produce new triples when necessary.

The Preparation Phase

As for each multiplication gate of the arithmetic circuit one random triple is spend in order to preserve the privacy of the input and output of the gate, the generation phase must produce at least m random triples. During the generation, each party $P_i \in \mathcal{P}$ receives shares of each component of the triples $(a^{(i)}, b^{(i)}, c^{(i)})$ of the ith block, $i = 1, \ldots, n$. The generation itself follows the protocol of [BGW88] and is non-robust. It is divided into blocks of length $l = \lceil m/n \rceil$. At the end of each block the consistency of the triples of one block is verified. This verification consists of two phases of fault-detection and fault-localisation, where in each phase n random triples are spend to provide privacy of the first l triples during the verification process. Thus, an overall amount of $l + 2n$ triples must be available. If a block contains inconsistent triples this will be detected with an overwhelming probability as we will see later-on in this section. In this case, the whole block is discarded and two players accusing each other of cheating are identified and excluded from the further preparation phase. This is done using the player elimination framework from [HMP00]. It is necessary to eliminate two parties as the origin of a fault cannot be localised uniquely. Consequently, in one player elimination process, the number n' of parties currently participating in the preparation phase is reduced by one, while the number t' of currently malicious parties is reduced by at least 1. Thus, it is guaranteed that at most t blocks fail and at most $n + t$ blocks have to be generated (initially given $t = t'$

and $n = n'$). The player elimination provides the inequality

$$t' < \lfloor \frac{n'}{3} \rfloor$$

as an invariant of this phase. This is a necessary assumption for the correct execution of all sub-protocols of the preparation phase[5].

0. Set $\mathcal{P}' := \mathcal{P}$, $n' := n$ and $t' := t$
1. Repeat until n blocks succeed:
   - 1.1 Generate $l + 2n'$ triples as presented in subsection 4.3.2.
   - 1.2 Verify the consistency of the first $l+n'$ triples, using the last n' triples. This is to be done with the fault detection algorithm from subsection 4.3.2. If a fault is detected identify and eliminate a set $\mathcal{D} \subseteq \mathcal{P}'$ of two players with the fault localisation from subsection 4.3.2. At least one player in $\mathcal{D}$ is cheating. Set $\mathcal{P}' := \mathcal{P}' \backslash \mathcal{D}$, $n' := n' - 2$ and $t' := t' - 1$. Remove the used triples from the block.
   - 1.3 If the triples have been successfully checked, use the last n' triples to verify that every player shares the correct values of the first l triples (fault detection from subsection 4.3.2). In case of a fault, use the fault localisation algorithm from subsection 4.3.2 to eliminate a subset $\mathcal{D} \subseteq \mathcal{P}'$ of two players such that at least one of them is cheating. Set $\mathcal{P}' := \mathcal{P}' \backslash \mathcal{D}$, $n' := n' - 2$ and $t' := t' - 1$ and remove the last n' triples.
   - 1.4 If either step 1.2 or 1.3 failed the actual block is discarded.

Triple Generation As mentioned in the beginning of this section, t-shared random triples are generated within the preparation phase. The generation of the triples of one block can be done in parallel. In a first step, t-shared random number a, b must be computed by the players. This is done intuitively. For the ith block, the players generate local random numbers $a^{(i)}, b^{(i)}$ and t'-share these values among the players of $\mathcal{P}'$ which have not yet been eliminated. This requires the parties to choose random polynomials $\sim f_i(X), \sim g_i(X) \in \mathbb{F}[X]$ with degree t' which serve for the interpolation polynomials hiding the random numbers. The parties together then generate t'-shares of the sum of all polynomials which represent sharings of the sum of the random number generated by the single parties. The exact procedure is given in sub-protocol 4.1.
Now, the players have to compute t'-sharings of $c = a \cdot b$. Doing so, each player P_i locally computes the product if his shares $\widetilde{a}_i, \widetilde{b}_i$ as

$$e_i = \widetilde{a}_i \cdot \widetilde{b}_i \, .$$

The result must then be t'-shared among the players. This requires to set up a t'-sharing of e_i as this value is actually $2t'$-shared after the local multiplication. Using Lagrange interpolation as presented in 4.2.1, each player reconstructs his own t'-shares of c. See protocol 4.2 for an algorithmic presentation. Because of the player elimination framework, it is possible that for some protocol instances

[5]For example, the invariant is needed for the robust broadcast simulation

The functionality of sub-protocol `Generate_ab`

Generation of t'-sharings of random numbers $a,\ b \in \mathbb{F}$ by n' parties $P_j \in \mathcal{P}$:

1. Every player P_i generates two random polynomials $\sim f_i(X), \sim g_i(X) \in \mathbb{F}[X]$ of degree t' and shares the random constant terms $a^{(i)}, b^{(i)}$ with the other players by sending the shares

$$\sim a_{ij} = \sim f_i(\alpha_j),\ \ \sim b_{ij} = \sim g_i(\alpha_j)$$

to player $P_j, 1 \leq j \leq n'$.

2. Each player P_j computes his shares of a, b as

$$\sim a_j = \sum_{i=1}^{n'} \sim a_{ij},\ \ \sim b_j = \sum_{i=1}^{n'} \sim b_{ij}.$$

Consequently, the sharing polynomials are defined as

$$\sim f(X) = \sum_{i=1}^{n'} \sim f_i(X),\ \ \sim g(X) = \sum_{i=1}^{n'} \sim g_i(X)$$

with $\sim f(0) = a, \sim g(0) = b$.

Protocol 4.1: Part 1 of the Triple Generation Phase

of `random_ab` and `share_c` the current robustness valuet' is smaller than t. In this case, the players have to lift the degree of the sharings of a, b, c to t. This is done by generating three random polynomials of degree t with constant terms zero. Adding sharings of these polynomials to the shares $\widetilde{a}_j, \widetilde{b}_j, \widetilde{c}_j$ causes a randomised increase of the degree of the resulting sharing polynomial. The whole procedure is given in protocol 4.3. It is easy to check that a_j, b_j, c_j are indeed t-shared now. The polynomial $f(X) = \sim f(X) + X \cdot \bar{f}(X)$ provides

$$f(0) = \sim f(0) + 0 \cdot \sum_{i=1}^{n'} \bar{f}_i(0) = a_j$$

and

$$f(\alpha_j) = \sim \alpha_j + \alpha_j \cdot \sum_{i=1}^{n'} \bar{f}(\alpha_j) = \sim \alpha_j + \alpha_h \cdot \sum_{i=1} n' \bar{a}_{ij}.$$

The analogous argumentation is valid for $g(X) = \sim g(X) + X \cdot \bar{g}(X)$ and $h(X) = \sim h(X) + X \cdot \bar{h}(X)$

As the triple generation is private but non-robust, each generation of a block of $l + 2n'$ triples is followed by a verification phase. It includes the verification of the degree of the shared triples as well as the verification that for $k = 1, \ldots, l$ all players P_i share the correct products $\tilde{e}_i^{(k)} = \tilde{a}_i^{(k)} \cdot \tilde{b}_i^{(k)}$ (from sub-protocol

The functionality of sub-protocol `Share_c`

Generation of t'-sharings of the product $c = a \cdot b \in \mathbb{F}$ by n' parties $P_j \in \mathcal{P}$:

1. Every player P_i computes his product share $e_i =\sim a_i \cdot \sim b_i$. Then he generates a random polynomial $h_i(X) \in \mathbb{F}[X]$ of degree t' with $h_i(0) = e_i$ and shares his e_i by sending $e_{ij} = h_i(\alpha_j)$ to $P_j, 1 \leq j \leq n'$.

2. Having received the values $e_{1j}, \ldots, e_{n'j}$, each player P_j reconstructs his t'-share of c as

$$c_j = \sum_{i=1}^{n'} \omega_i e_{ij}, \text{ with } \omega_i = \prod_{j=1, j\neq i}^{n'} \frac{\alpha_j}{\alpha_j - \alpha_i}.$$

Protocol 4.2: Part 2 of the Triple Generation Phase

The functionality of sub-protocol `Lift`

Lifting the t'-sharings of the product $a, b, c \in \mathbb{F}$ to t-sharings:

1. Every player P_i generates random polynomials $\bar{f}(X), \bar{g}(X), \bar{h}(X)$ of degree $t-1$ and sends the shares

$$\bar{a}_{ij} = \bar{f}(\alpha_j), \bar{b}_{ij} = \bar{g}(\alpha_j), \bar{c}_{ij} = \bar{h}(\alpha_j)$$

   to player $P_j, 1 \leq j \leq n'$.

2. Each player P_j reconstructs his t-shares a_j, b_j, c_j of a, b, c as

$$a_j = \tilde{a}_j + \alpha_j \sum_{i=1}^{n'} \bar{a}_{ij},\ b_j = \tilde{b}_j + \alpha_j \sum_{i=1}^{n'} \bar{b}_{ij},\ c_j = \tilde{c}_j + \alpha_j \sum_{i=1}^{n'} \bar{c}_{ij}.$$

Protocol 4.3: Part 3 of the Triple Generation Phase

`share_c`). In case, faults are detected, a fault localisation and subsequent player elimination is performed. The sub-protocols which implement these functionalities are presented now.

Fault Detection and Localisation I After having generated a block of $l + 2n'$ random triples, it has first to be verified that the degree of all sharings it t. By construction, the degree it at least t, which means, by cheating, a player could have caused t' to be larger than t. The basic idea is to use (and finally discard) n' triples for the verification of the remaining $l + n'$ triples. The verification must not leak information about the sharings, which is achieved by verifying the degree of a blinded linear combination of the sharing polynomials of each share of $a^{(k)}, b^{(k)}, c^{(k)},\ k = 1, \ldots, l + n'$. Obviously, the degree of a linear combination of polynomials is the same as the maximum degree of the single polynomials. Consequently, a sharing with degree $t' > t$ would cause the

linear combination to have degree $t' > t$. For the computation of the random linear combination each player P_i sends a random challenge vector $r^{(i)} \in \mathbb{F}^{l+n'}$ to the other players. The corresponding linear combinations of each involved polynomial is then reconstructed towards the challenging player who checks that the resulting polynomial is of appropriate degree. To preserve the privacy of the involved polynomials, for each verifier P_v one additional blinding polynomial is added to the linear combination. The sub-protocol is given in protocol 4.4.

The functionality of sub-protocol `Fault-detection I`
Verify the degree of all sharings in one block:

1. The verifier selects a random vector $[r_1, \ldots r_{l+n'}] \in \mathbb{F}^{l+n'}$ and sends it to each player $P_j \in \mathcal{P}'$.

2. Every player P_j computes and sends to P_v the following corresponding linear combinations and the share of the blinding polynomial for $1 \leq i \leq n'$:

$$\begin{aligned}
\tilde{a}_{ij}^{(\sum)} &= \sum_{k=1}^{l+n'} r_k + \tilde{a}_{ij}^{(}k) + \tilde{a}_{ij}^{(}l + n' + v), \\
\bar{a}_{ij}^{(\sum)} &= \sum_{k=1}^{l+n'} r_k + \bar{a}_{ij}^{(}k) + \bar{a}_{ij}^{(}l + n' + v), \\
\tilde{b}_{ij}^{(\sum)} &= \sum_{k=1}^{l+n'} r_k + \tilde{b}_{ij}^{(}k) + \tilde{b}_{ij}^{(}l + n' + v), \\
\bar{b}_{ij}^{(\sum)} &= \sum_{k=1}^{l+n'} r_k + \bar{b}_{ij}^{(}k) + \bar{b}_{ij}^{(}l + n' + v), \\
\tilde{c}_{ij}^{(\sum)} &= \sum_{k=1}^{l+n'} r_k + \tilde{c}_{ij}^{(}k) + \tilde{c}_{ij}^{(}l + n' + v), \\
\bar{c}_{ij}^{(\sum)} &= \sum_{k=1}^{l+n'} r_k + \bar{c}_{ij}^{(}k) + \bar{c}_{ij}^{(}l + n' + v).
\end{aligned}$$

3. P_v verifies whether for each $1 \leq i \leq n'$ the shares $\tilde{a}_{i1}^{(\sum)}, \ldots, \tilde{a}_{in'}^{(\sum)}$ lie on a polynomial of degree at most t'. The same is done for the shares $\tilde{b}_{i1}^{(\sum)}, \ldots, \tilde{b}_{in'}^{(\sum)}$ and $\tilde{c}_{i1}^{(\sum)}, \ldots, \tilde{c}_{in'}^{(\sum)}$, $1 \leq i \leq n'$. Also P_v verifies whether for each $1 \leq i \leq n'$, the shares $\bar{a}_{i1}, \ldots, \bar{a}_{i1}$ as well as $\bar{b}_{i1}, \ldots, \bar{b}_{i1}$ and $\bar{c}_{i1}, \ldots, \bar{c}_{i1}$ lie on a polynomial of degree at most $t - 1$.

4. P_v broadcasts[a] one bit according to whether all the $6n'$ verified polynomials have degree at most t', respectively $t - 1$, or at least one polynomial has too high degree.

[a] With a suitable broadcast simulation sub-protocol

Protocol 4.4: Fault detection I

If one verifier detects that the degree of at least one of the linearly combined polynomials is too high, he broadcasts this information to the other players and all triples of the actual block are discarded. Also, the sub-protocol `Fault-localisation I` from protocol 4.5 is run to determine and eliminate a set $D \subseteq \mathcal{P}'$ of two players, from whom at least one is corrupted. Although it is possible, that only one of the players in D is actually corrupted, he is also eliminated as his innocence cannot be proven. Obviously, the sub-protocol is run if and only if one verifier has broadcast a complaint in step 4 of sub-protocol `fault-detection I`. This verifier is denoted with P_v in the fault-localisation

protocol. In case, there were several verifiers sending a complaint, the one with smallest v is selected to be the verifier in protocol `Fault-localisation I`, protocol 4.5.

The functionality of sub-protocol `Fault-localisation I`
Elimination of a subset $D \subseteq \mathcal{P}'$ of two players of whom at least one is corrupted:

1. The verifier P_v selects one of the polynomials of too high degree and broadcasts[a] the location of the fault, consisting of the index i and the information which sharing is concerned $(\tilde{a}, \tilde{b}, \tilde{c}, \bar{a}, \bar{b}, \bar{c})$. Without loss of generality, it is assumed now, that the fault was observed in the sharing $\tilde{a}_{i1}^{(\sum)}, \ldots, \tilde{a}_{in'}^{(\sum)}$.
2. The owner P_i of this sharing (i.e., the player who acted as dealer for this sharing) sends to the verifier P_v the correct linearly combined polynomial $\tilde{f}_i^{(\sum)} = \sum k = 1^{l+n'} r_k \tilde{f}_i^{(\sum)}(X) + \tilde{f}_i^{l+n'+v}(X)$.
3. P_v finds the smallest index j such that $\tilde{a}_{ij}^{(\sum)}$, received from player P_j in step 2 of the fault-detection, does not lie on the polynomial $\tilde{f}_i^{(\sum)}(X)$ (received from the owner P_i from step 2 of this protocol), and broadcasts j among the players in $\mathcal{P}'$.
4. Both P_i and P_j send the list $[\tilde{a}_{ij}^{(1)}, \ldots, \tilde{a}_{ij}^{(l+n')}, \tilde{a}_{ij}^{(l+n'+v)}]$ to P_v.
5. P_v verifies that the linear combination $[r_1, \ldots, r_{l+n'}]$ applied to the values received from P_i is equal to $\tilde{f}_i^{(\sum)}(\alpha_j)$. Otherwise, P_v broadcasts the index i, and the set of players to be eliminated is $D = \{P_i, P_v\}$. Analogously, P_v verifies the values received from P_j to be consistent with $\tilde{a}_{ij}^{(\sum)}$ received in step 2 of the fault-detection, and in case of failure broadcasts the index j. In this case the subset $D = \{P_j, P_v\}$ is eliminated.
6. P_v finds the smallest index k such that the values $\tilde{a}_{ij}^{(k)}$ received from P_i and P_j differ, and broadcasts k and both values $\tilde{a}_{ij}^{(k)}$ from P_i and $\tilde{a}_{ij}^{(k)}$ from P_j.
7. Both P_i and P_j broadcast their value of $\tilde{a}_{ij}^{(k)}$.
8. If the values broadcast by P_i and P_j differ, the the localised set is $D = \{P_i, P_j\}$. If the value broadcast by P_i differs from the value broadcast by P_v, then $D = \{P_i, P_v\}$. Else, $D = \{P_j, P_v\}$.

[a] Using a sub-protocol for broadcast simulation

Protocol 4.5: Fault localisation I

If at least one of the involved sharings in any of the $l + n'$ triples has too high degree, this is detected by any honest verifier with probability at least $1 - 1/|\mathbb{F}|$.

Fault Detection and Localisation II Now, it must be verified that in each triple $(a^{(k)}, b^{(k)}, c^{(k)},\ k = 1, \ldots, l$, of a block every player P_i shared the correct product share $\tilde{e}_i^{(k)} = \tilde{a}_i^{(k)} \cdot \tilde{b}_i^{(k)}$. It is already verified that the sharings of all factor shares are of degree t'. Therefore, it is sufficient to verify that the shares of $\tilde{e}_1^{(k)}, \ldots, \tilde{e}_{n'}^{(k)}$ lie on a polynomial of degree at most $2t'$. This polynomial is uniquely defined by the shares of the at least $n' - t' > 2t'$ honest players. In analogy to the fault-detection I, fault-detection II (as presented in protocol 4.6) again uses n' of the remaining $l+n'$ triples of the current block in order to verify l triples. Those triples are then discarded. Every player P_v distributes a random challenge vector, and the corresponding linear combination of the polynomials plus one blinding polynomial is reconstructed towards P_v. The protocol can be invoked concurrently for each verifier $P_v \in \mathcal{P}$.

The functionality of sub-protocol `Fault-detection II`
Verify that all players share the correct product shares:

1. The verifier P_v selects a random vector $[r_1, \ldots r_l] \in \mathbb{F}^l$ and sends it to each player $P_j \in \mathcal{P}'$.
2. Every player P_j computes and sends to P_v the following corresponding linear combinations and the share of the blinding polynomial for $1 \leq i \leq n'$:
$$\tilde{e}_{ij}^{(\sum)} = \sum k = 1^l r_k \tilde{e}_{ij}^{(k)} + \tilde{e}_{ij}^{(l+v)}.$$
3. P_v verifies whether for each $1 \leq i \leq n'$ the shares $\tilde{e}_{i1}^{(\sum)}, \ldots, \tilde{e}_{in'}^{(\sum)}$ lie on a polynomial of degree at most t'. If so, he verifies, whether the secrets $\tilde{e}_1^{(\sum)}, \ldots, \tilde{e}_{n'}^{(\sum)}$ of the above sharings lie on a polynomial of degree at most $2t'$. P_v broadcasts one bit, to whether all polynomials have appropriate degree.

Protocol 4.6: Fault detection I

As in `Fault-localisation I`, `Fault-localisation II`, too, is executed if and only if one verifier has reported a fault in Step 3 of the above protocol. Again, if there are several such verifiers, the one with smallest v is selected for the protocol `Fault-location II` in protocol 4.7.
Every honest verifier will detect the fault with probability at least $1 - 1/|\mathbb{F}|$, if at least one of the involved shares has degree higher than $2t'$.

Analysis As mentioned before, the protocol of Hirt and Maurer assumes an unbounded active adversary. I have presented in section 4.1.2 the different attacker models. There it is also mentioned that an active adversary may be static or adaptive. The difference of both models lies in the point of time the attacker has to decide which party she wants to attack. Now, an analysis of the error probability of the preparation phase is given with respect to the two models.
First, I discuss the case of a static adversary. We are interested in the probability that a bad triple is inserted into a block and this is not detected by the honest

The functionality of sub-protocol `Fault-localisation II`

Elimination of a subset $D \subseteq \mathcal{P}'$ of two players of whom at least one is corrupted:

1. If in Step 3 of `Fault-detection II` the degree of one of the second-level sharings $\tilde{e}_{i1}^{(\sum)}, \ldots, \tilde{e}_{in'}^{(\sum)}$ was too high, then P_v applies the error correction to find the smallest index j such that $\tilde{e}_{ij}^{(\sum)}$ must be corrected. Since the sharings have been verified to have correct degree, P_v can conclude, that P_j has sent the wrong value $\tilde{e}_{ij}^{(\sum)}$. P_v broadcasts j and the set $D = \{P_j, P_v\}$ has to be eliminated. In this case, the protocol stops here.

2. Every player P_i sends to P_v all his factor shares $\tilde{a}_i^{(1)}, \ldots, \tilde{a}_i^{(l)}, \tilde{a}^{(l+v)}$ and $\tilde{b}_i^{(1)}, \ldots, \tilde{b}_i^{(l)}, \tilde{b}^{(l+v)}$

3. P_v verifies for every $k = 1, \ldots, l, l+v$ whether the shares $\tilde{a}_1^{(k)}, \ldots, \tilde{a}_{n'}^{(k)}$ lie on a polynomial of degree t'. If not, P_v applies error-correction and finds and broadcasts the smallest index j such that $\tilde{a}_j^{(k)}$ must be corrected. The set of players to be eliminated is $D = \{P_j, P_v\}$. The same verification is performed for the shares $\tilde{b}_1^{(k)}, \ldots, \tilde{b}_{n'}^{(k)}$ for $k = 1, \ldots, l, l+v$.

4. P_v verifies for every $i = 1, \ldots, n'$ whether the value $\tilde{e}_i^{(\sum)}$ computed in Step 1 is correct, i.e., whether

$$\tilde{e}_i^{(\sum)} \stackrel{?}{=} \sum_{k=1}^{l} r_k \tilde{a}_i^{(k)} \tilde{b}_i^{(k)} + \tilde{a}_i^{(l+v)} \tilde{b}_i^{(l+v)}.$$

   This test will fail for at least one i, and P_v broadcasts this index. The players in $D = \{P_i, P_v\}$ are eliminated.

Protocol 4.7: Fault localisation II

players. Consequently, we must assume that in a block at least one triple is faulty. Every honest player will detect this with probability at least $1 - 1/|\mathbb{F}|$. This implies that the probability that no honest player detects the inconsistency is at most

$$1 - (1 - 1/|\mathbb{F}|)^{n'-t'} = |\mathbb{F}|)^{-n'+t'}.$$

Once a bad block is detected, at least one corrupted player is eliminated. Consequently, the attacker can try at most t times to insert a bad triple. The probability that at least one of these trials is not detected is at most

$$\sum i = 0^{t-1} |\mathbb{F}|^{-(n-t-i)} \leq \left(\frac{1}{|\mathbb{F}|}\right)^{n-2t}.$$

In the adaptive case, the corruption could happen after the challenge vector is known. Thus, a bad block passes the verification, if at least $n' - t'$ of the challenge vectors cannot discover the fault. The probability for this event is at

most

$$\sum_{i=0}^{t'} \binom{n'}{i} (1 - \frac{1}{|\mathbb{F}|})^i (\frac{1}{|\mathbb{F}|}^{n'-i} \leq (\frac{3}{|\mathbb{F}|})^{n'-t'}.$$

As in the static case, the adversary can try t times to introduce a bad block. The overall error probability is then

$$\sum_{i=0}^{t-1} (\frac{3}{|\mathbb{F}|})^{n-t-i} \leq (\frac{3}{|\mathbb{F}|})^{n-2t}.$$

This probability can be decreased by repeating the fault-detection with new blinding triples. Unfortunately, this implies the costly broadcast primitive to be heavily used. Consequently, in practice, the security gain will most oftenly not justify the additional expenses.

Computation Phase

The actual computation consists of two phases. First, the input provided by the players to the circuit is t-shared. Second, the actual circuit evaluation takes place. The evaluation goes along the lines of [Bea91a] with slight modifications, which are necessary because the upper limit t' on the number of corrupted players does not need to be the same as the degree t of the sharings. The protocol of Hirt and Maurer requires t-sharings as all players, independent from having been eliminated or not, are allowed to provide input to the circuit and to take part in the evaluation. In the end, all players must receive output from the computation.
From the preparation phase, the players in $\mathcal{P}'$ have m t-shared random triples $(a^{(i)}, b^{(i)}, c^{(i)})$ with $c^{(i)} = a^{(i)} \cdot b^{(i)}$. As mentioned before, eliminated players may provide input and take part in the evaluation, but they act outside the protocol and are ignored by the players in $\mathcal{P}'$. Since the player elimination framework guarantees the inequality

$$2t' < n' - t,$$

the circuit can be efficiently and correctly evaluated by the $n' - t'$ honest players in $\mathcal{P}'$.

Input Sharing

In the input sharing phase, every player t-shares his input with the remaining players in the subset $\mathcal{P}' \subseteq \mathcal{P}$. The sharing itself is performed with the verifiable secret sharing protocol from [BGW88][6], which provides perfect security. The protocol is slightly modified to face $t \neq t'$.
Now, let us denote a dealer by P, and the secret to be shared by s. The dealer can be any party, not necessarily in $\mathcal{P}'$ or $\mathcal{P}$, that shows itself responsible to share the secret s. Normally, it will be a player in the set $\mathcal{P}$. The input sharing is done by hiding the secret in the constant term of a random polynomial $f(X, Y) \in \mathbb{F}[X, Y]$, i.e., $f(0, 0) = s$. If we represent the values $f(\alpha_i, \alpha_j)$ as a $n' \times n'$ matrix over $\mathbb{F}$, the dealer then shares s by providing each party $P_i, i = 1, \ldots, n'$ with the ith column and the ith row. Then, the players P_i send to all players P_j

[6]The protocol is presented it section C

the value $f(\alpha_i, \alpha_j)$. This way every player P_j can check, whether these values lie on the polynomial $f(\alpha_j, Y)$ and $f(X, \alpha_j)$. In case, complaints occur, the dealer broadcasts the correct points $f(\alpha_i, \alpha_j)$ and $f(\alpha_j, \alpha_i)$. This enables all players P_k to check with their polynomial $f(\alpha_k, X)$. If at most t' complaints are broadcasted every player P_k takes $f(\alpha_k, 0)$ as his share. Otherwise, the dealer must be faulty (as in this case at least one honest player sent an accusation, which must be true). In this case, the players take a default sharing. In practice, the dealer will most often be either a trusted party or one of the players, sharing his own input among the players. Then, a default sharing has no influence on the correctness of the evaluation, as there are at most t' malicious players and consequently at most t' default shares. The polynomials of at least $n' - 2t' > t$ honest players are consistent and uniquely define the polynomial $f(X, Y)$ with degree t. The whole input sharing is presented in protocol 4.8.

The functionality of sub-protocol `Input Sharing`

t-Sharing of a secret s among the players in $\mathcal{P}'$.

1. The dealer P selects a random polynomial $f(X, Y) \in \mathbb{F}[X, Y]$ of degree t, with $f(0, 0) = s$. Then, he sends the polynomials $f_i(Y) = f(\alpha_i, Y)$ and $g_i(X) = f(X, \alpha_i)$ to player P_i, $i = 1, \ldots, n'$.
2. Every $P_i \in \mathcal{P}'$ sends the values $f_i(\alpha_j)$ and $g_i(\alpha_j)$ to each $P_j, 1 \leq , j \leq n'$.
3. Every player P_j broadcasts one bit according to whether all received values are consistent with the polynomials $f_j(Y)$ and $g_j(X)$.
4. The secret sharing is finished if no player broadcasts a complaint. In this case, the share of player P_j is $f_j(0) = f(\alpha_j, 0)$. Otherwise, every complaining player P_j broadcasts a bit vector of length n', where a 1-bit in position i means, that one of the values received from P_i was not consistent with $f_j(Y)$ or $g_j(X)$. The dealer answers all complaints by broadcasting the correct values $f(\alpha_i, \alpha_j)$ and $f(\alpha_j, \alpha_i)$.
5. Every player P_i checks whether the values broadcast in step 4 are consistent with his polynomials, and broadcasts either confirmation or accusation. The dealer answers every accusation by broadcasting both polynomials $f_i(Y), g_i(X)$ of P_i, and P_i replaces his polynomials by the broadcast ones.
6. Every player P_i checks whether the polynomials broadcast by the dealer are consistent with his polynomials, and broadcasts either a confirmation or accusation.
7. If in Steps 5 and 6, there are at most t' accusation, every P_i takes $f_i(0)$ as his share of s. Otherwise, the dealer is faulty and the parties take default sharings.

Protocol 4.8: Input Sharing

Circuit Evaluation

In an arithmetic circuit 3 kinds of gates may be differentiated:

1. addition
2. scalar multiplication
3. multiplication

In the following subsections we define $u,\ v \in \mathbb{F}$ as t-shared values and

$$p(X) = \sum_{i=1}^{t} c_i X^i + u,\ q(X) = \sum_{i=1}^{t} c'_i X^i + v$$

the secret polynomials. The value $c \in \mathbb{F}$ be publicly known. The following paragraphs clarify, that both addition and scalar multiplication on shares are local operations while multiplication requires communication.

Addition:
Obviously, $p + q(0) = u + v$ and $\deg(p + q) = t$. Furthermore, $p(\alpha_i) + q(\alpha_i) = p + q(\alpha_i)$ (linearity of p). This implies that each party p_i can compute its share of $u + v$ locally.

Scalar Multiplication:
The value $c \cdot u$ can be shared by the polynomial $cp(X) = \sum_{i=1}^{t} c \cdot u + c \cdot c_i X^i$. Furthermore, each party p_i can compute its share of $c \cdot u$ locally because of $p(c \cdot \alpha_i) = c \cdot p(\alpha_i))$ (linearity of p).

Multiplication:
Multiplication of two t-shared values u, v cannot be done locally because

$$\deg(p(X) \cdot q(X)) = 2t$$

which implies that the product $u \cdot v$ would be $2t$-shared and a reconstruction would be impossible as soon as more than $\lceil n/6 \rceil - 1$ malicious parties are involved into the computation.

Therefore, a multiplication gate is evaluated according to the circuit randomisation of [Bea91a] using one t-shared triple (a, b, c) with $c := a \cdot b$ as generated in the preparation phase. The product $u \cdot v$ can be written as follows:

$$u \cdot v = ((u - a) + a)((v - b) + b) = ((u - a)(v - b)) + (u - a)b + (v - b)a + c$$

It is computed in the following steps:

1. Local computation of the differences $d_u = u - a$ and $d_v = v - b$. This can be done since $u,\ v,\ a,\ b$ are t-shared.
2. Reconstruction of d_u and d_v. No information about u and v is lacking in this process as a and b are random numbers.
3. Now, $d_u * d_v$ are publicly known and shares of the product $u \cdot v = d_u d_v + d_u b * d_v a * c$ can be computed by each party through scalar multiplication and addition, respectively.

The multiplication protocol requires two secret reconstructions. This implies all parties to send the respective shares to the other parties. Consequently, the communication costs per multiplication gate are $2n^2$.

Secret Reconstruction

In order to enable a party P_i to reconstruct a secret, each party sends its share to P_i. In a first step P_i runs the error-correction from section 4.2.3 to correct up to t incorrect or missing shares. Then, it uses the interpolation algorithm from section 4.2.1 to reconstruct the secret. The reconstruction towards one player requires n shares to be sent to this player.

Chapter 5

Secure Multi-Agent Computation

In chapter 4 the protocol for synchronous secure multi-party computation from Hirt and Maurer [HM01] has been presented. This protocol enables us to design a protocol for secure mobile multi-agent computation which does not rely on a trusted third party. As already mentioned in the context of general secure multi-party computation in the introduction of chapter 4, an Alliance of co-operating agents simulates and substitutes a trusted authority. This approach is the first of its kind in the field of mobile agents. We have firstly introduced it in [EM03], based on the protocol for SMPC from Ben-Or et al. [BGW88] which is secure in the UC framework[1] of Canetti [Can00]. The resulting SMAC protocol was mostly of theoretical interest as the protocol from [BGW88] must be regarded as impractical. We made several refinements of protocol components aiming on a complexity reduction of the SMAC protocol, including the incorporation of the efficient Hirt-Maurer protocol. These results are presented in the Diploma thesis of Mie [Mie03]. After all, our protocol achieves a complexity that is quadratic in the number of parties. As SMPC protocols require broadcast channels which is in general achieved by broadcast simulations which cannot be done below a quadratic complexity, this result must be considered optimal. The basic ideas of SMAC as well as the connection between SMPC and SMAC have been described in chapter 3. This chapter describes SMAC in depth.
In order to enable an Alliance of co-operating agents to securely fulfil its task, it must be provided with mechanisms allowing it to

1. manage data in a way that its privacy is protected against an adversary, and

2. securely perform its computations.

We have seen in chapter 3 that using a protocol for secure multi-party computations allows to protect the computation phases of an Alliance. As no migration takes place in those phases, the parties of the SMPC protocol can be identified with the hosting servers of the Alliance. This identification requires as an

[1] The UC framework is a security model that provides unconditional security for protocols. For more information see Appendix B

assumption for the SMAC protocol, that all Alliance members are hosted on different servers at any time. I have already discussed the reasons for this assumption in chapter 3. Then, using the protocol of Hirt and Maurer [HM01], all computations remain secure as long as at most $t(n) = \lceil n/3 \rceil - 1$ Alliance members are located on malicious hosts. Since this protocol assumes a synchronous network, additional synchronisation techniques (e.g. that of Awerbuch from [Awe85]) are necessary when using the protocol in an asynchronous setting. As a consequence, an attacker could stop the computations but it is not capable of hurting the correctness and privacy. We have seen in section 4.1.4 that in general, this situation is similar to the one in an asynchronous protocol. At the same time, in comparison to asynchronous protocols, the efficient Hirt-Maurer protocol offers the possibility of a practical implementation.
Migration requires special handling. While one protocol instance has to be terminated on the current hosts, a new one has to be started on the new hosts. This requires interaction between $2n$ hosts. Consequently, the computations cannot be part of the running instance of the n-party SMPC protocol. Other means must be taken. The transition from one instance to the next must not leak any information to the adversary. Therefore, the final step of the SMAC protocol requires the old hosts to delete all knowledge about the Alliance. In case of malicious hosts, one can assume that they do not follow this step of the protocol, hoping to be able to exploit their knowledge at some future time. Data shares represent a part of this knowledge which is of particular sensitivity as their manipulation can cause the computations to become incorrect. Therefore, it is part of the protocol that the shares are reliably disabled in each migration pre-processing phase, before starting the migration post-processing phase which is the new protocol instance. This is done by using a randomised re-sharing technique, like the one from Ostrovsky and Yung [OY91], which allows to compute shares of new shares by means of n-party computations on the old hosts. These shares are then sent to the new hosts and reconstructed there. The re-sharing protocol is described in depth in section 5.2.5.
The remainder of this chapter is organised as follows: First, in section 5.1, the security model for the SMAC protocol is introduced. Then, the structure of an Alliance and its members is presented. After this preparatory part, section 5.2 describes the functionality of the SMAC protocol in an informal way, focussing on the initialisation phase and the migration pre- and post-processing phases. Section 5.3 presents the protocol from the point of view of the host. This is necessary because in contrast to SMPC the protocol for SMAC cannot identify the host with the agent at all times. There are specific tasks within the dynamic phases that have to be fulfilled by the host, independent from its hosted agent instance. Last but not least, section 5.4 finally presents the protocol for secure multi-agent computation.

5.1 The Security Model

Before presenting the protocol for secure multi-agent computation it is necessary to define the security model that is assumed. This includes for example the definition of the protocol components, error types and network type. As long as these presumptions are kept, the protocol provides unconditional security (with a small error probability).

5.1.1 The Attacker Model

While the Hirt-Maurer protocol is secure against an adaptive adversary which is computationally unbounded, the model for secure multi-agent computation requires a different attacker model. SMAC makes use of cryptographic primitives like authentication and encryption. As discussed in section 4.1.2, this makes it necessary to limit the adversary's computational strength and to assume a polynomially bounded one. I have also discussed the consequences of migration and the possibility to collect shares over time resulting from it. Therefore, the SMAC model has to assume a mobile adversary that is able to hop from one set of corrupted agents to another. This can only be prevented by using randomised re-sharing techniques as presented in section 5.2.5.
Caused by the necessity of a broadcast simulation, SMPC as in [HM01] limits the number of parties the adversary is capable to corrupt simultaneously to $t(n) = \lceil n/3 \rceil - 1$. SMAC additionally makes use of majority decisions, which would allow an adversary to corrupt up to $t(n) = \lceil n/2 \rceil - 1$ agents without loosing security. However, as SMAC partially consist of SMPC, we have to demand the more restrictive limit of $t(n) = \lceil n/3 \rceil - 1$ for the overall SMAC model.

5.1.2 Error Behaviour

For now, we identify the error behaviour of the model with the behaviour of the adversary. As a Byzantine adversary is the usual model today and mirrors the reality best, it is also assumed for the SMAC model. In section 4.1.3 a more detailed classification of errors is given, which mainly aims at network errors. In theory, these can be also be assigned to the adversary. For matters of simplicity, I do not integrate those errors in the security model. Nonetheless, aiming at a practical implementation, it is important to take a close look on the underlying network and existing network attacks, because these could cause the SMAC to fail (but not to be incorrect) although the condition of less than $\lceil n/3 \rceil - 1$ corrupted agents is kept at any time. Therefore, an analysis of the SMAC model under real network assumptions is given in chapter 6.

5.1.3 Network Type

I assume an asynchronous network since this thesis aims not only at the presentation of a theoretical security model, but also on an analysis of the practicability of it. This requires the model to assume open and asynchronous networks like the Internet. The adversary controls the network and the message delivery with it. Although in principle the secure channel model (encrypted and authenticated channels) is not required, the protocol uses encryption and authentication in order to increase the tolerance against network attacks. This is discussed in depth in chapter 6.
The SMPC protocol from [HM01] assumes synchronous networks. Nevertheless, in this thesis, it is used in an asynchronous setting. This poses the problem of having no guaranteed message delivery. In order to get a satisfactory solution to this problem, I will introduce an artificial synchronisation, using timers. As one cannot differentiate network errors from message delays caused by a corrupted agent, it is necessary to classify an agent whose messages do to arrive in time

as corrupted. The problems arising from the absence of guaranteed message delivery and synchronism of the parties are handled in the given model and discussed in chapter 6.

5.1.4 Overall Security

The protocol of [HM01] provides unconditional security with a small probability of error. This is not kept for the SMAC model. For matters of efficiency, some sub-protocol providing only cryptographic security are used. In addition, cryptographic measures are inevitable when it comes to the security of agent migration which is not covered by [HM01]. Consequently, the overall security is cryptographic which is absolutely satisfactory for real applications where all attackers are computationally bounded.

5.2 How Alliances Work

We have seen before that the advantage of an agent Alliance lies in its ability to securely evaluate a user-defined function in a possibly malicious environment. The robust nature of the computations can tolerate up to $t(n) = \lceil n/3 \rceil - 1$ wrong or missing inputs without loosing correctness or fairness. Thereby, it does not matter in which way the agents got corrupted and how this corruption manifests itself. The only thing that really matters is to keep $t(n)$ at any time. This property is influenced by diverse network factors, but also by the distribution of the agents. Locating several agents of the same Alliance on one server is in this context not reasonable. The identification of the parties in SMPC with the actual hosts of an Alliance is not meaningful anymore. In principle one could do so, but having n agents hosted on $k < n$ hosts implies $t(n) = \lceil k/3 \rceil - 1 < \lceil n/3 \rceil - 1$ which is no better than running a k-party protocol. The latter provides the same redundancy with lower complexity. Consequently, the SMAC model expects all agents of one Alliance to be hosted on different servers.
Nonetheless, a server is allowed to host members of several Alliances. In this case, it must be ensured that the single agent instances are sufficiently encapsulated. In section 2.1 I have discussed the security measures of operating systems and programming languages, especially of Java. We have seen that encapsulation of processes can be considered standard, today. For agent Alliances this is a prime requisite and should be checked carefully before running a protocol instance. If not guaranteed, an attacker could send its own Alliance to other hosts in order to spy out another Alliance. Consequently, the privacy of computations would get lost even though $t(n)$ is not exceeded. The correctness would still be given. However, there is another threat: By such an espionage, the attacker could get the private key of the Alliance and abuse it. For example, it could trick the Alliance's originator by generating a substitute Alliance that cannot be differentiated from the original one as is able to authenticate itself correctly, but returns arbitrary results.
The hosting servers have to provide an Alliance with an interface allowing the Alliance members to communicate with each other (and possibly with other Alliances). This is necessary to perform distributed computations. I assume an authenticated network to increase security. This requires users to authenticate at a trusted authority (not necessarily a Certification Authority). The private

keys of the servers are then used for the communication between the Alliance members. I find this requirement necessary, even if uncomfortable for the users, because of two reasons:

1. Users that have been somehow authenticated will tend to honest behaviour.

2. Spoofing or misleading Alliance messages can corrupt an Alliance even though less than $t(n)$ members are corrupted. This point will be subject of further discussion in chapter 6, where network attacks are analysed.

The major goal of this thesis is the design of a *completely* secured agent Alliance. This means, the whole functionality defined by its originator has to be preserved as long as no more than $t(n)$ agents are corrupted at the same time. For reasons of efficiency, one can demand this property for some sensitive sub-functionalities only. This is discussed in chapter 8 for some exemplary sub-functions. However, the general model which is presented in this chapter aims at the protection of the whole functionality and all data.

5.2.1 Protocol Components

At any point in time, distributed computations are performed by a fixed number of n parties $P_1, \ldots, P_n$. But there are some more parties involved in the protocol. First, the trusted originator O, which is the owner of the agent Alliance and as such responsible for its correct initialisation and the reconstruction of the shared computation result. Consequently, the originator appears only twice: At the start of the protocol and at result return, when the agents return home with the shared results of their common computation. The other parties are the so-called *migration targets*, which represent the servers that have been determined via a distributed computation to be the new hosts of the Alliance. While in the role of being the migration target, these parties are never involved into distributed computations. But, if everything goes all right, the old host leaves the protocol and the migration target becomes one of the regular n parties.
The n parties, again, are separated into

- non-corrupted parties which can be trusted, and
- the adversary, consisting of up to $t(n)$ corrupted parties, attacking the protocol execution.

The *adversary* is an important entity in the model. It controls the behaviour of the corrupted parties and possesses their knowledge. Therefore, it is necessary to know its exact power and behaviour. In the Alliance model, we have a mobile adversary that is computationally bounded.

5.2.2 Errors and Corruption

A host is supposed to execute an agent, so it must have full control over the code and potentially over the agent data. Thus, an agent must be considered to be corrupted as soon as its host behaves actively maliciously. This means that it somehow deviates from the protocol. These deviations are not always detectable. At the same time, it may be that it only seems as if the agent or its

host behaves maliciously. This could happen in case of network errors causing a delay of messages. In general, the Alliance cannot differentiate between those error classes. Therefore, as soon as it seems as if an agent does not follow the protocol, it is assumed to be corrupted. As the agent's state (corrupted, uncorrupted) is most probably a consequence of a manipulation done by its host, the host is considered as malicious if and only if the agent behaves strangely. Consequently, we amalgamate the agent and its host to one entity during a computation phase. Thus, although the agent is an active and autonomous identity with a functionality that is independent from its hosts, it is *not* considered as a party in the SMPC protocol. Instead, the host is. So, from now on, the hosts are called the *parties* while the agents participating in the common computation are called *members of an agent Alliance*.

5.2.3 Structure of an Alliance Member

Let A_i be a member of an Alliance $\mathcal{A}$ consisting of n mobile agents, which has been designed for the joint fulfilment of a task T. The agent program is divided into two parts: a public part and a private one. Without loss of generality, the public parts of the agents are assumed to be identical. The public part consists of a program and some public data. We can divide the program into an application-related part which realises the user-defined functionality T and a part that acts like an operating system. The first is given by an arithmetic circuit which is evaluated gate by gate as defined in the SMPC protocol. We can assume this part to be identical for each Alliance member. As we use a synchronous protocol for SMPC, we have to introduce synchronisation techniques which must be organised and controlled outside the SMPC protocol. Furthermore, communication primitives and the organisation of majority decisions and migration are required. All this is part of the agent's operating system.
Public data, although seemingly unsecured and even unknown in the SMPC protocol, is necessary in the Alliance model. An Alliance member must be able to provide its host with information that is needed for synchronisation, migration and communication. This cannot be avoided, as on the one hand, the agent depends on information like that from a system clock, and on the other hand the host requires knowledge about for example the locations of the other Alliance members. This public data poses no new threats as it is either secured by regular majority decisions (see section 5.2.6) or, if static, digitally signed by the originator. The following list gives a survey on the public data components of an Alliance member:

- $ID_{\mathcal{A}}$: Each Alliance needs a unique Alliance identifier $ID_{\mathcal{A}}$ for situations requiring the agent to prove to be a member of a specific Alliance. In particular, on result return the originator must be able to determine to which set of shares the results of one specific agent belong. But also the hosts must be able to determine to which Alliance an agent belongs. As the code and thus its signature may be the same for different Alliances, the hash value is not suitable to differentiate different Alliances of the same originator. Another identifier is necessary. As we want to provide an Alliance with a private key, the identifier can be chosen as the corresponding public key $pub_{\mathcal{A}}$.
- ID_{A_i}: The agent identity ID_{A_i} allows to authenticate single agents. It is

used for example when an Alliance performs a majority vote.

- We have seen in chapter 4 on SMPC that the underlying secret sharing scheme assigns a point α_i to each party P_i. This value remains the same during the whole execution of the SMPC protocol. The situation is the same for the SMAC protocol. But here, each Alliance member must know all points. Otherwise, it would not be able to share new knowledge with its co-operating agents.

- n: The size n of $\mathcal{A}$ is primarily required for majority votes which again are used for the migration process.

- O: The originator's identity O is of interest for two reasons:

  1. Thanks to the authentication, a black list containing misbehaving servers can be used on the agent platform. Also, if an originator is found guilty of having sent malicious code, it may be put on this list.
  2. On result return, the Alliance can authenticate the receiving server. No spoofing attack is possible.

  As we assume authenticated networks, the identity O is just the public key pub_O of the originator.

- Loc^{c_m}: A location list Loc^{c_m} is used to identify those servers currently hosting the Alliance. This is necessary to determine who is authorised to contact an Alliance member and to provide it with (shared) input. The location list is implemented as an array with n entries, storing the location of agent A_i in the ith entry. For reasons of security the location list is depending on the migration counter c_m which is explained below.

- t_{ex}: A maximum execution time t_{ex} for one agent per host. This allows the Alliance members to decide whether an agent has been corrupted. Here, the corruption can consist of not executing an agent, keeping it forever etc. As soon as t_{ex} is exceeded, the agent has to start its migration phase together with its co-operating Alliance members. If the migration phase is not initiated in time, the agent is assumed to be corrupted.

- t_{rep}: A maximum reply time t_{rep}. This is essential if the SMPC protocol is a synchronous one such as [HM01]. The Internet is highly asynchronous. Therefore, it is not guaranteed that messages are delivered within a specific time interval. It is even possible that they get lost. This is why a synchronisation is done within the SMAC protocol. An agent, regardless of whether it really is, is assumed to be corrupted as soon as it does not send its messages in time.

- t_{exp}: The expiration date t_{exp} defines when the Alliance is supposed to return home at the latest. After the expiration date, the originator assumes the Alliance as compromised.

In addition to the public parameters, an agent carries the following shared parameters:

- $c_{rep(serv)}$: Counters $c_{rep(serv)}$ for the number of agents which witness the reply time of a host *serv* or, during the migration pre-processing phase, that of a candidate server which has been sent an execution request to be exceeded.

- c_m: A migration counter c_m that counts the number of migrations of the agent.

- c_{ex}: Each agent must be provided with a counter c_{ex} which counts the number of agents which witness the expiration of the permitted execution time t_{ex}.

These counters are kept shared because it is a local operation to increment a counter. This means, it is an efficient operation and, at the same time, the counters cannot be manipulated by an attacker.
To enable an Alliance to determine whether the counter $c_{rep(serv_l)}$ exceeded the maximum, it starts a timer t_{rep} after sending an execution request and at the beginning of a computation phase. As soon as this timer signals a timeout, the agent broadcasts this information to the other Alliance members. After receiving such a message, the agent checks whether it has previously received an identical message from this agent and, if not, increments the respective counter by one. As the agents rely on their hosts to be able to run the clocks, manipulations are possible. Assuming at most $t(n)$ messages concerning the same counter as being manipulated (which is conform to our assumption that at most $t(n)$ hosts are malicious), an agent reliably knows about the expiration of the respective counter after at least $t(n) + 1$ identical messages. This is given at the latest as soon as at least $2t(n) + 1$ messages have arrived. Thus, an Alliance does not have to wait for delayed or even missing inputs of all members before being able to react. Having successfully determined that $c_{rep(serv_l)}$ exceeded t_{rep} the Alliance starts to determine a new migration target. In case c_{ex} exceeded t_{ex}, a migration pre-processing phase is induced. In the model presented here, a migration is always a consequence of exceeding t_{ex}. Other solutions are possible but require a complex organisation of the migration requests of single Alliance members. We have presented this approach in [EM03].
As mentioned above, the location list Loc^{c_m} contains very sensitive knowledge and manipulations could endanger the whole Alliance. Unfortunately, the list is dynamic and cannot be protected by the originator's signature. This problem could be solved by a distributed signature through the Alliance itself at each migration. This has been proposed in [Mie03]. Meanwhile, we did an exact analysis on the complexity of such a signature. The results were disillusioning. The analysis is given in [Amm07]. For practical purposes it makes more sense to use the means of majority decision for this data, too. This requires each party to send its version of the latest location list to each of the new servers before stopping the current instance of the protocol. Then, the new hosts can determine a correct location list in analogy to the checking of counters as explained above.
In the beginning of this section, I have introduced an Alliance member as consisting of public and private parts. The public part has been discussed in depth above. Now, I turn to the private one. This part consists entirely of shared data. Shared data originates from two sources: Either it has been initially produced by the protocol compiler, or an agent shared its new knowledge during on-going computations. However, all shared data is generated by using Shamir's secret

sharing scheme from [Sha79]. This means, that single shares do not provide any information on the original data to an adversary. If and only if at least $t(n)+1$ correct shares are available, the information can be reconstructed. Each Alliance member A_i shares the Alliance knowledge and organises it in a list S_i. Examples for shared knowledge are user-preferences or the private key of the Alliance.

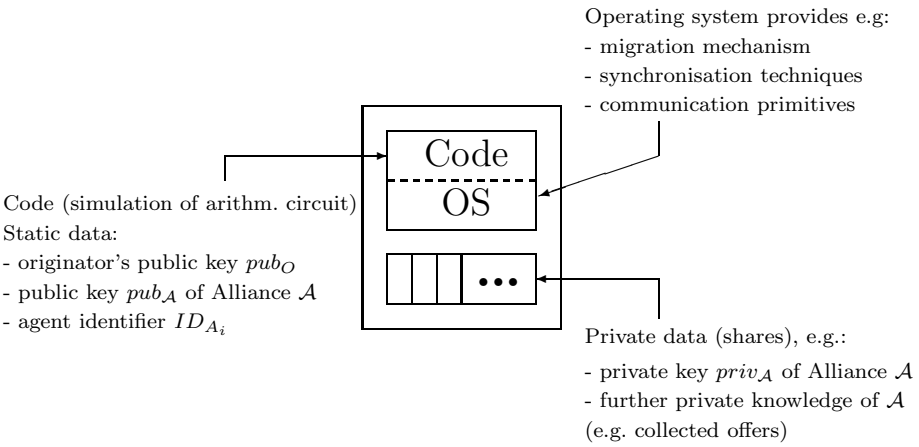


Figure 5.1: Structure of an Alliance member

Figure 5.1 illustrates the structure of Alliance member A_i. The list S_i is located under the code/operating system part of the agent. It is organised as a linearly addressed memory, in which the position $S_i[j]$ is used to store the jth $t(n)$-shared value s_{ij}. Thus, the lists $S_1, \ldots, S_n$ represent the distribution of the Alliance's state $S_{\mathcal{A}}$, consisting of the data s_j for $j \geq 1$.

5.2.4 The Initialisation Phase

One goal of the security model presented in this thesis is to bother the originator of an agent as little as possible with the underlying protocol. The originator should be able to create his agent with the tools he is used to. The only decision he must make is to fix the grade of redundancy. The choice of $t(n)$ depends on three factors, mainly:

1. The sensitivity of the application: The higher it is, the higher the redundancy should be chosen.

2. The network hostility: One has to estimate the number or agents that could get corrupted at the same time. This leads to results which strongly deviate, depending on the type of network (open, closed, authenticated, trusted, untrusted).

3. The costs of communication: Redundancy is expensive[2]. So, the price of communication significantly influences the choice of $t(n)$.

[2]Here: in terms of communication overhead.

The number of distributed multiplications necessary to fulfil the task T is also important, as multiplications require communication (see chapter 4). But this should not influence the decision as much as the other factors.
Having fixed the redundancy parameter $t(n)$, the originator is supposed to simulate the preparation phase. In the preparation phase the random triples, which are necessary for the distributed multiplications, are generated. In the protocol for SMPC from [HM01] this is part of the distributed computations. The generation itself is relatively efficient, but in case of wrong triples, a complex fault detection takes place. As we can assume the originator as trusted with respect to his own Alliance, he can simulate this phase by securely generating the necessary triples through local computations only. This requires a random number generator and a good estimate on the number of required distributed multiplications. In principle, an Alliance can be designed to generate new random triples during the computation phases, too.[3] This feature could be of interest if the user-defined functionality does not allow to reliably rate the number of multiplications. As such, this ability can be regarded as a necessary measure with respect to the practicability of an Alliance.
After the triple generation, the originator feeds the function that the Alliance should evaluate into a protocol compiler which creates an arithmetic circuit with the same functionality. In other words, the compiler creates a circuit which simulates the originator's program. This simulation is not trivial and will be discussed in chapter 7. Not only the originators program has to be simulated, it must be extended by the operating system mentioned in section 5.2.3. The agent database must be transformed into $(n, t(n))$-shared data. This can be easily done with Shamir's secret sharing scheme. For reasons of efficiency, n should always be chosen equal to $3t(n) + 1$. Other choices do not increase redundancy but increase communication and computation complexity. In a final step, $3t(n) + 1$ Alliance members are generated, each of them with unique shared data packages stored in the list S_i.
After generation of $n = 3t+1$ agents, the originator determines n different hosts. It is necessary to select different hosts for two reasons:

1. As all agents run the same code, no additional information can be gained from the host databases if more than one agent is located on the same hosts. In contrary, they would compete for the same resources.

2. If the protocol does not forbid servers to host more than one agent of the same Alliance, there could be an information leakage. By chance, a host could collect enough different shares, namely $t+1$, and so reconstruct sensitive information like a private key. Computations, too, are compromised in case $t + 1$ agents are hosted simultaneously by the same host.

This is a real threat and therefore it is among the model assumptions, that each migration of an Alliance results in a set of n different hosts. After having determined n suitable migration targets, the originator sends a request for execution to the servers. If a server does not respond to the request in time (within t_{rep}) or a server does not agree, the originator determines an alternative server. Only if n servers agreed, O provides the Alliance with a signed location list and sends it out. This is the last interaction between him and his Alliance before it will finally return with the computation result.

[3] This is mentioned in [HM01]

5.2.5 Migration Pre-Processing Phase

The migration preparation phase mainly consists of two tasks:

1. Fixing a location list, containing the identities of all new hosts.

2. Computing of new shares for the new hosts (re-sharing).

Both can be done securely, as the Alliance is still hosted on the current n hosts. All necessary computations can be done either distributedly or secured by majority decisions.
The migration pre-processing phase starts for two reasons only. First: if the agent got informed about the expiration of t_{ex}, it forwards this message to the other Alliance members in order to invoke a migration sub-protocol. Until the new hosts are determined and the actual migration takes place, the Alliance could continue concurrently with the circuit evaluation. Second: if the expiration date t_{exp} of the Alliance expires, the host informs its agent about this. The agent forwards the information to the other Alliance members. The distributed computations are stopped and the hosts return their agents to the originator.

Fixing the Location List

The process of computing a new location list includes two steps that may be executed repeatedly. First, n candidates have to be determined. The Alliance can do this securely, using an arbitrary selection function. This function could determine the servers randomly or depending on the application. It has been mentioned before that thanks to the authentication of the servers, a black list could be made available to the selection function. It could contain hosts which have been proven to behave maliciously, but also application dependent information. It makes no sense to send agents to hosts that cannot provide information that is necessary to fulfil their task. Thus, such hosts could be explicitly excluded. One could also separate the list into two parts, one containing servers that are to be excluded from the selection function, the other containing a set of prefered servers. This short discussion already makes clear that a black list depends on many preferences of the originator. It is therefore not part of the default agent structure as introduced in section 5.2.3. However, it can be easily added if desired. The black list should be in clear-text and signed by the originator O of the Alliance. Although this implies the list is static, it is a reasonable decision. The advantages of distributed storage and possibly regular updates is disproportionate to the communication and computation complexity this would cause.
After having determined a candidate, each Alliance member sends an execution request to it. This request must be answered withing the predefined time interval t_{rep}. If there is no answer withing t_{rep} or a negative answer arrives at a majority of the Alliance, the selection function determines a new candidate. Although it seems reasonable to design the selection function as a function which outputs one candidate only, one should keep in mind that it is computed distributedly. This means, that it causes communication and should therefore be called as seldom as possible. This is why the selection function is assumed here to output i candidate servers at once. To speed up the process of host determination, an Alliance could send requests to more than n candidate servers in parallel. In

case too many agree, the Alliance could select the most prefered n ones and reject the remaining.
An execution request contains the public parts of an Alliance, namely code, static data, and operating system (see figure 5.1 from section 5.2.3) and the current location list Loc^{c_m}. Sending the remaining public data has two reasons:

1. It allows the receiving server to check the integrity of the agent before it agrees to the execution.

2. In case the candidate server agrees on the execution only the private data of the agent remains to be transmitted.

It has been mentioned before, that each Alliance member sends an execution request to the candidate. The data it contains is signed by the originator of the Alliance and can be checked easily. As soon as the public data of a request is verified, the candidate can store this data. Nonetheless, it is required that at least $t(n)+1$ Alliance members send the request, as a subset of up to $t(n)$ agents could mislead the candidate to believe in a non-existent migration.
If it is honest, the candidate replies to each and every execution request concerning one Alliance. At this stage, it does not have to differentiate, whether a former host maliciously sent the request[4] or whether the request was sent by an authorised host. Later-on, when it comes to the transmission of the new shares and the location list, an authentication of the current hosts, proving that they really are hosts of the same Alliance, is indispensable. This is part of the second step of the migration pre-processing phase and will be discussed next. As soon as the candidate server agrees on the execution of the Alliance, and the Alliance has verified this by a majority decision, the candidate is stored in the new location list Loc^{c_m+1}.

Re-sharing

Having fixed the location list Loc^{c_m+1}, the second step of the migration pre-processing phase starts. The Alliance has to securely inform the new servers about the future locations of the Alliance members by sending the new location list. As the new servers do not know the identity of the current servers, they cannot distinguish messages sent by authorised from those sent by unauthorised servers. Therefore, it is not sufficient to just let each agent send its new location list to each new server, which in turn performs a local majority decision on the arriving messages. At the same time, the correctness of the location list is of major importance as it is used to authenticate messages of the hosts of one Alliance. This implies that additional means must be taken to protect it. Although the Alliance could use the protocol of Hirt and Maurer to distributedly compute a signature of the list, this is not advised here. The data volume necessary to transfer over the network for such a signature is outrageous (see our analysis in [Amm07] and a cross-reference in section 8.2 on page 123). Fortunately, there are protocols especially for threshold signatures which are much more efficient than general multi-party computations. One current example is given by Fischlin in [Fis03]. But a distributed signature by [HM01] could also be considered when limiting the underlying circuit to a finite ring. It is shown in section 8.2

[4]This is possible, since the public part of the Alliance members does not change and a malicious host could just store and re-use it

that with this restriction [HM01] allows for a very efficient signature. Signing the location list is not only advantageous for authenticating the current hosts to the future ones. It also allows the future hosts to check whether they have really been selected as new hosts, and that the shares they receive are correct.
After having secured the new location list, the next task is to compute shares for the new servers and to disable the current ones. Ostrovsky and Yung have faced a similar problem in [OY91]. There, not a mobile agent scenario was assumed but a virus infecting server after server, thereby collecting information. This is the original setting for the definition of a mobile adversary as introduced in section 4.1.2. Ostrovsky and Yung already assumed data that is secured by the use of a secret sharing scheme. Thus, their task was to re-share that data from time to time to prevent the virus from collecting more than $t(n)$ shares. Obviously, such a re-sharing requires not only to generate new shares, but also to make the old ones useless. The protocol is presented now.

The Protocol of Ostrovsky and Yung
Let $f(X) \in \mathbb{F}_p[X]$ be the polynomial of degree $t(n)$ used to hide the information s, and $s_i = (\alpha_i, f(\alpha_i)), 1 \leq i \leq n$, the interpolation points of the respective parties $P_1, \ldots, P_n$. Be $f'(X) \in \mathbb{F}_p[X]$ a random polynomial of degree $t(n)$ with $f'(0) = 0$. Then, the sum

$$g(X) = f(x) + f'(x)$$

allows a randomised re-sharing of s. This the case, because the resulting polynomial $g(X)$ has $t(n)$ random coefficients and the secret s is determined by $g(0) = s$. As polynomials over $\mathbb{F}_p[X]$ are additively homomorphic, the interpolation point of $g(X)$ can be locally computed by $\hat{s}_i = (\alpha_i, f(\alpha_i) + f'(\alpha_i))$.
The actual computation of the new shares requires the randomising polynomial $f'(X)$ to remain secret. Otherwise, a mobile adversary that collected $k < t + 1$ old shares could simply continue collecting re-shared data. By a local subtraction of $f(\alpha_i), 1 \leq i \leq n$, it could produce enough old shares to be able to reconstruct the secret s. Therefore, the sum must be computed distributedly, which actually means to compute shares of the old shares and f'. The whole re-sharing process consists of four steps. It is presented in protocol 5.1.

5.2.6 Migration

During the determination of the new location list, the candidate hosts already received code, static data and the operating system of the Alliance. Because of the certificate they were able to check the integrity of this data. As they were provided with the old location list Loc^{c_m} during the migration pre-processing phase, it is guaranteed that they only accept messages from the former hosts and no attacker is able to infiltrate the data exchange that takes part in the actual migration.
What is still missing for the execution of the new protocol instance is the result of the re-sharing as well as the new location list. Without this information, the new protocol instance cannot be invoked. The transfer of this data defines the migration phase.
In order to provide the new hosts with the re-shared data, the former hosts send the shares of shares that have been produced in step 3 of the re-sharing

Sub-protocol `re-share`

Implements a randomised re-sharing of the Alliance state.

1. Generating and sharing of randomising polynomial f':

   All parties exchange random bits in order to generate randomising polynomials $f'_1, \ldots, f'_n$ with $f'_i(0) = f'(\alpha_i)$. That means, the f'_i interpolate a random polynomial f'. Each party P_j receives one interpolation point (share) of $f'_1, \ldots, f'_n$:

   $$r_{1j} = f'_1(\alpha_j), \ldots, r_{nj} = f'_n(\alpha_j).$$

2. Sharing of current shares:

   In order not to leak information on the current shares, they have to be shared before being added to the random polynomial. Doing so, the parties generate, in analogy to the first step, n shared random polynomials $f_1, \ldots, f'_n$ with degree $t(n)$ and $f_i(0) = s_i = f(\alpha_i), 1 \leq i \leq n$. Each party P_j receives one interpolation point (share) of $f_1, \ldots, f_n$:

   $$s_{1j} = f_1(\alpha_j), \ldots, s_{nj} = f_n(\alpha_j).$$

3. Computing of shares of new shares:

   After having received all shares from step 1 and 2, each party P_j computes shares of the new shares by

   $$\hat{s}_{1j} = s_{1j} + r_{1j}, \ldots, \hat{s}_{nj} = s_{nj} + r_{nj}.$$

   Each P_j then sends the share $\hat{s}_{ij}, 1 \leq i \leq n$, to party P_i.

4. Reconstructing of new shares:

   Each party P_i has received n shares $\hat{s}_{ij}, 1 \leq i \leq n$, and is able to interpolate its new share s_i.

Protocol 5.1: Sub-protocol `re-share`

protocol to the new hosts. Those then perform step 4 of the protocol in order to reconstruct their relevant data shares. The correctness and secrecy of this process is ensured as long as less than $t(n) = \lceil n/3 \rceil$ Alliance members send manipulated data. This is given by the SMAC protocol assumptions. For the secure transfer of the location list, other means must be taken. As mentioned before, it would be too costly to secure this list by a distributed signature of the Alliance itself. On the other hand, the security of the protocol execution heavily depends on the security of the location list. Therefore, the Alliance uses a majority decision as security measure. This implies that all Alliance members send their local copy of Loc^{c_m+1} to all new hosts, which in turn count the same versions until there is one version with a majority of at least $n/2 + 1$. This procedure guarantees that a correct list is computed, as long as less than $n/2$

corrupted agents sent manipulated data. As the general protocol assumption is that at least $2n/3 + 1$ agents are honest, this is always the case.

5.2.7 The Migration Post-Processing Phase

Now that the Alliance has been securely transfered, the old instance of the SMAC protocol has to be stopped and a new instance has to be started. Stopping an instance requires the former hosts to stop all processes and communication that are related to the Alliance. Furthermore, all knowledge, namely data shares and static data, must be deleted.
At the same time, the new hosts reconstruct their data shares and thus share the current state of the Alliance. All shares used for reconstruction must originate from a server that is listed in Loc^{c_m}. After reconstruction and start of the agent execution, the hosts immediately start the local timer t_{ex}. This action is the first asked by the agent and defines the start of the new protocol instance. In a last step, the operating system of the agent starts the application specific multi-party protocol.

5.2.8 The Computation Phase

After migration and initialisation on the new hosts, the Alliance continues work, which means the evaluation of the arithmetic circuit can be resumed. At this point, an arbitrary protocol for secure multi-party computation can be used. Later-on, for the analysis of program simulation and the complexity of specific applications (see chapters 7 and 8) the Hirt-Maurer protocol, presented in chapter 4, is used.

5.2.9 Result Return

After the circuit has been evaluated, the Alliance returns to its originator. This requires the hosts to send all Alliance members back to the server of the originator that is part of the signed public part of the agents. The originator then applies the reconstruction from [HM01] (see chapter 4, page 75). to reconstruct the computation result.

5.3 Host View on a Protocol Execution

An execution request for an Alliance member reaches a server S in form of up to n single requests from the hosts of the Alliance. The request contains the public (signed) part of the agent (i.e. $ID_{\mathcal{A}}, ID_{A_i}, t_{exp}$), the current public local location lists Loc^{c_m}, and the shared migration counter c_m. As up to $t(n)$ requests could originate from corrupted Alliance members, the server has to determine a correct version of the location list by counting consistent lists. If there is a majority of at least $n/2+1$ consistent lists, it must be the correct one (as $t(n) < n/3$). The migration counter can be and is reconstructed, as soon as all requests have arrived so that all shares are available. Then, the server performs the following checks:

1. Do the requests concern the same Alliance identifier $ID_{\mathcal{A}}$ and agent identifier ID_{A_i}?

2. Is the integrity of the signed public data given?

3. Did the messages originate from different servers in Loc^{c_m}?

4. Has there already been an execution request concerning this Alliance and agent with the same value of the migration counter c_m, although t_exp is not yet reached?

Each request failing at least one of these checks is considered corrupted. If at most $n/2$ requests exist that have passed the tests, the server refuses the execution of the respective agent and informs all hosts in Loc^{c_m} that have passed the test about its decision. It then deletes the request and refuses any further communication requests related to this Alliance as the Alliance has obviously been compromised.
If at least $n/2 + 1$ requests have passed the tests, the server may (after some additional considerations) agree to the execution. In a first step it stores the migration counter c_m and the Alliance identifier $ID_{\mathcal{A}}$ until the expiration date is reached. This is necessary to allow the detection of replay attacks as done in step 4 of the test series. Then, S sends its positive decision to all hosts in Loc^{c_m} that have sent correct requests. At the same time, it allocates an encapsulated address space and stores the public part of the agent there. Thenceforward, S accepts messages concerning this agent if and only if they originate from a host in Loc^{c_m}. Initially, it will receive its new shares of the Alliance's state and the new location list Loc^{c_m+1}. After checking whether it is mentioned in Loc^{c_m+1}, S proceeds with the reconstruction of its shares. It then starts the actual agent execution.
In a first step, the agent asks S to start the counter t_{ex} and to inform it as soon as it expires. If so, S sends a broadcast message to the hosts of Loc^{c_m+1}. The receiving hosts inform their agents about it, which causes the agents to invoke a sub-protocol that counts respective messages and increments the local counter c_{ex}. If this counter reaches the value $2t(n) + 1$, the application-specific part of the protocol is continued concurrently to the determination of new hosts by the migration sub-protocol. The phase is synchronously started by the Alliance members, as all honest hosts have the same value of c_{ex}. S supports these preparations by offering a secure re-sharing sub-protocol for the Alliance's state and sending the resulting shares of shares to the respective new hosts. This is the last action server S performs for the Alliance. Afterwards, it deletes all data that is related to the Alliance, except $ID_{\mathcal{A}}, c_m$ and t_{ex}.

5.4 The Protocol for SMAC

In this section the protocol for secure mobile multi-agent computation is presented in depth. The SMAC protocol offers the possibility to design the whole functionality of the Alliance as a $t(n)$-robust function. It is built upon two main blocks: First, the protocol for secure multi-party computation from [HM01] handling the computation phases. Second the sub-protocols involved in the migration pre-processing phases. The latter make use of majority votes and additional cryptographic primitives like digital signatures, and encryption. As the computation phases do not require special handling, the description focusses on the transition from one computation phase to another, which means the

sub-protocols of the migration process. For efficiency the current version of the SMAC protocol, which is presented here, does not allow single agents to migrate. A migration process is very costly in terms of communication complexity, as it requires the execution of a re-sharing protocol. Our original framework [EM03] offers more flexibility by using a semaphore to serialise migration requests of single Alliance members. Besides the high complexity, a migration is a time consuming distributed procedure, which in the worst case could result in spending most of the execution time for computations related to migration instead of evaluating the user-defined function.

5.4.1 Sub-Protocols and Sub-Routines

Before going into the details of the protocol, it is necessary to review the nomenclature and variable identifiers.

Abb.	definition	type of data
$\mathcal{A}$:	an Alliance	public, signed
A_j:	member of $\mathcal{A}$, $1 \leq j \leq n$	public, signed
O:	originator of the Alliance	public, signed
c_m:	migration counter	shared
Loc^{c_m}:	locations of $A_1, \ldots, A_n$	public
t_{ex}:	execution time on each host	public, signed
t_{rep}:	maximum time for servers to reply	public, signed
t_{exp}:	expiration date of $\mathcal{A}$	public, signed
$c_{rep(serv)}$:	counts witnesses for exp. of t_{rep} for server	shared
c_{ex}:	counts witnesses for expiration of t_{ex}	shared
H_0:	initial host; trustworthy	
H_{jc_m} :	c_mth host visited by agent A_j	

The protocol consists of several sub-protocols. They are presented in this section using pseudo code notation. But first, two interfaces, necessary for the interaction between the agents and for invoking sub-protocols, are introduced.

1. `deliver`:

   As distributed computations require communication between the parties, it is necessary to provide a function which sends messages to specific subsets of $\mathcal{A}$ defined in a subset Loc'^{c_m} of the location list Loc^{c_m}. The signature of this interface is

   $$\texttt{deliver}(Loc'^{c_m}, m).$$

   Since all agents migrate at once, `deliver` can consider the locations in Loc'^{c_m} as valid. In case single agents were allowed to migrate (see [EM03]), the routine would have to check, whether the target agents have already asked for migration and stopped working. In this case, the messages had to be buffered until new hosts are determined for these agents. Given a parallel migration of all agents, this sub-routine is only called within computation phases, where no migration takes place.

2. `run`:

   The SMAC protocol consists of several cryptographic sub-protocols. As soon as such a sub-protocol has to be invoked, the routine `run` is called. It invokes a local instance of the n-party protocol X providing a random

input parameter r to this instance. The random parameter is necessary in order to distinguish concurrent protocol executions and to increase security. The signature of `run` is

$$\texttt{run}(Loc^{c_m}, X, r)$$

Since migration is the most challenging part of the SMAC protocol, we start with a description of the migration process and the sub-protocols involved. Consider an agent A_j that is currently executed on a host H_{jc_m} being informed by H_{jc_m} about the expiration of its maximum execution time t_{ex}. It immediately calls the function

$$\texttt{deliver}(Loc^{c_m},\text{"}t_{ex} \text{ has expired at } A_j\text{"})$$

to inform the Alliance of its state. This information causes the Alliance members to invoke the sub-protocol `count` by calling

$$\texttt{run}(Loc^{c_m},\texttt{count},r)$$

that allows the agents to count the number of Alliance members have been informed that their t_{ex} has expired. The protocol is presented in protocol 5.2.

Sub-protocol `count`

Implements a majority decision about the expiration of the Alliance execution time during a computation phase.

1. Increase the distributed counter c_{ex} by 1
2. If $c_{ex} > 2t(n)$, then
   execution of sub-protocol
   `run(`Loc^{c_m}`,migrate(`$\mathcal{A}$`),`r`)`

Protocol 5.2: Sub-protocol `count`

Protocol `count` invokes sub-protocol `migrate` which handles the actual migration pre-processing phase. It is presented in protocol 5.3.As explained in depth in section 5.2, two sub-protocols are used. First, new hosts must be determined. This is done with sub-protocol `new_hosts` that computes n new hosts for the Alliance. The protocol is not specified here, as there is an uncountable number of possibilities for its design, strongly depending on the given network properties and user preferences. One possibility is to simply select candidates at random. More advanced approaches could use blacklists, reputation, etc. However, the output of `new_hosts` is the location list Loc^{c_m+1}. The second sub-protocol is `re-share`, a randomised re-sharing protocol that computes shares of the shares of the Alliance state. In section 5.2.5 the protocol from Ostrovsky and Yung [OY91] has already been introduced as one option. Therefore, protocol `re-share` is not further discussed here. Now, all sub-routines and sub-protocols necessary for the protocol for secure multi-agent computation are available and the presentation can continue with the SMAC protocol itself.

5.4.2 The Protocol Secure Multi-Agent Computation

Initialisation Phase

Sub-protocol `migrate`

Implements the migration of Alliance $\mathcal{A}$ in case of an execution expiration.

1. Invoke `run`(Loc^{c_m},`new_hosts`,r)
2. Invoke `run`(Loc^{c_m},`re-share`,r)
3. For each $H_{ji+1} \in Loc^{c_m+1}$ call
   `deliver`($H_j c_m + 1, s_j c_m + 1$)

Protocol 5.3: Sub-protocol `migrate`

1. The originator O, located on host H_0, generates n agents $A_1, \ldots, A_n$ belonging to an Alliance $\mathcal{A}$ with identifier $ID_{\mathcal{A}}$. This is done with the protocol compiler that has been introduced in chapter 3. Consequently, the resulting Alliance members are provided with some signed public data, some shared knowledge, the same code and a small operating system.

2. H_0 computes an initial list $Loc^1 = [H_{11}, \ldots, H_{n1}]$ containing the hosts of the first protocol instance.

3. For all $1 \leq j \leq n$, the host H_0 sends the message

   $$(ID_{A_j}, ID_{\mathcal{A}}, A_j, c_m, t_{exp}, Loc_{c_m-1}, \text{“Agree?”}, O)$$

   to Host H_{j1}, whereby A_j only contains the public parts of the agent. In parallel the originator starts timers $t_{resp(H_{j1})}$ for $1 \leq j \leq n$.

4. As long as there is any $j, 1 \leq j \leq n$, with outstanding positive response:

   If any timer $t_{resp(H_{j1})}$ runs out or any H_{j1} sends a negative response:
   compute a new host H_{j1} for A_j, send
   $(ID_{A_j}, ID_{\mathcal{A}}, A_j, c_m, t_{exp}, Loc^{c_m-1}, \text{“Agree?”}, O)$
   to H_{j1} and restart timer $t_{resp(H_{j1})}$.
   Else
   If $H_{j1}, 1 \leq j \leq n$, sends “yes” then $t_{resp(H_{j1})}$ is stopped and H_0 makes an endorsement about j in Loc^1.

5. H_0 signs Loc^1 and sends it along with the shares s_{j1} to the members H_{j1} of Loc^1.

Life phase of A_j on host H_{jc_m} ($c_m \geq 1$) Starts with receipt of execution request $req_{ex} = (ID_{A_j}, ID_{\mathcal{A}}, A_j, c_m, t_{exp}, Loc^{c_m-1}, \text{“Agree?”}, H_i(c_{m-1}))$

1. If $H_{i(c_m-1)} = H_0$
   - store request;
   - check the integrity of the data;
   - check expiration date and migration counter;
   If integrity is corrupted or same request occured before
   - call *deliver*(H_0,”no, Alliance $ID_{\mathcal{A}}$ corrupted”);

- delete request (store $ID_{\mathcal{A}}, t_{exp}, c_m$ till t_{exp} expires);
set *dec*="yes";
Else
Whileless than $n/2+1$ equal execution requests req_{ex} have arrived
- store incoming requests;
- check the integrity of the data;
- check whether the expiration date has expired;
If t_{ex} has not yet expired
check whether there was a previous
request for the same Alliance identifier and c_m;
If "yes"
- delete request and ignore further messages originating
from the responsible server.
- exit loop;
If any of the integrity tests fails
set *dec*="no";
Else *dec*="yes";

2. Determine Loc^{c_m-1} by a majority decision on all received lists or set $Loc^{c_m-1} = H_0$, respectively.

3. Call *deliver*(Loc^{c_m-1},"yes");

4. Store up to n incoming data shares originating from hosts in Loc^{c_m-1}. As soon as there is a sufficient number of shares available, reconstruct the share s_{jc_m}.

5. Start the execution of A_j (as a first action start the timer t_{ex}.

6. During the circuit evaluation, the following events may occur at H_{jc_m}:

   - calls of `deliver`(H_{ic_m}, m), where m contains data necessary to evaluate a multiplication gate (cf. section 4.3.2 on page 72)
   - incoming messages necessary for circuit evaluation; forwarding to agent A_j
   - expiration of t_ex; forward information to A_j
   - expiration of t_{expt}; forward information to A_j
   - Invocation of sub-routines `run`(Loc'^{c_m}, X, r)
     for distributed computations (e.g. `count, migrate`)
   - Sharing new data with the other Alliance members (data collection)
   - Delivery of expiration messages to the Alliance with subroutine `deliver`(L, m)

7. As soon as step 1 of the sub-protocol `migrate` is finished (i.e. protocol `new_hosts` has determined a set of n new hosts), H_{jc_m} exits the event loop. Now, the remaining steps of sub-protocol `migrate` are executed.

8. Delete all knowledge connected to A_j, besides the identifier $ID_{\mathcal{A}}$, expiration date t_{exp} and migration counter c_m.

Chapter 6

Alliance Computations in Real Networks

6.1 Bad Things to do to an Alliance

As mentioned before, Alliance compromise always requires the compromise of at least $t_{max} + 1$ of the agents at any one time. In practical deployment there also exists the possibility of Denial-of-Service (DoS) attacks, with various purposes. DoS attacks can aim at or support the violation of the $t_(n)$–limit. They can also be targeted directly at Alliance sabotage. DoS Attacks are discussed in more detail in Section 6.3.

Attack trees Attack trees [Sch99] are a specialised version of general decision trees. They are used to organise complex reasoning that starts with abstract concepts and subsequently divides these concepts into more concrete sub-concepts. The variant of attack trees used here represents a more abstract attack by the incoming edge of a node and the more concrete sub-cases by the outgoing edges. Hidden assumptions and additional reasoning determining the number and nature of the outgoing edges is attached to the nodes. This additional information is given in textual form. Node numbers are used as reference. A leaf node indicates that the specific sub-attack it corresponds to is deemed unsuitable for further graphical subdivision and is discussed in textual form instead.
It is important to note that attack trees are not a method for formal reasoning in the mathematical sense. They are a very useful tool to organise informal argumentation, make it more transparent and facilitate identification of special cases. However, the analysis itself still relies on human ingenuity and is contained in the text and not in the tree. The tree just binds the argumentation together.

Analysis

The attack tree for secure mobile agent Alliances is shown in Figure 6.1. The following list gives hidden assumptions and possible attacks and countermeasures on a node-by-node basis. Many node-comments will contain forward-references to later parts of the paper.

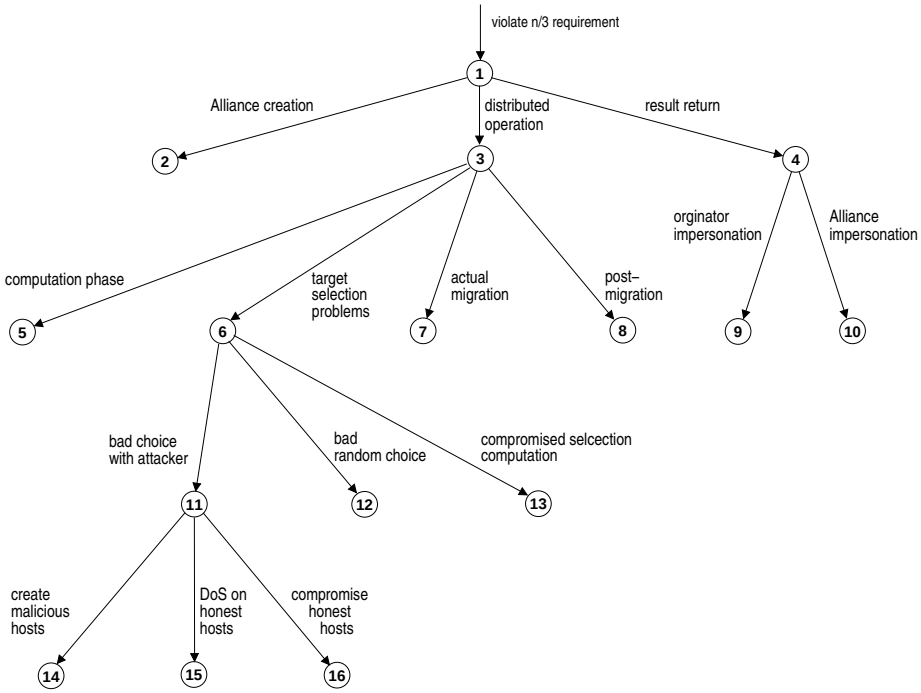


Figure 6.1: Attack tree

1. Attacks can only happen during the lifetime of an Alliance. The originator is trusted, therefore the initial migration can be treated like any later migration.

2. The originator is trusted, so no attack is possible.

3. The Alliance is required to follow the life-cycle of Figure 3.2 from chapter 3. The subdivisions are named a little differently, but represent the same steps. From a security point of view, the possible vulnerabilities in migration preparation are centred on migration target selection. Other preparation steps are just distributed computations.

4. Basically two attacks are possible in result return: To send fake data to the originator or to impersonate the originator to obtain the result the Alliance is reporting. (If both takes place simultaneously, no harm might be done.)

5. Since the distributed computation method used is believed to be secure, no compromise is possible during the computation phase, as long as there are no more than $t_(n)$ malicious agents in the Alliance. However there is not only one computation phase, but one after each migration. See section 6.5 for a discussion of the security implications.

   A second concern during the computation phase is Denial of Service. See Section 6.3 for a detailed discussion.

6. The target selection can either be bad because the computation was compromised, because of random factors or because an attacker changed the degree of the network hostility.

7. Possible attack here: Migration target impersonation. Whether this is possible depends on the host authentication mechanism used. See Section 6.4 for a detailed discussion. If enough malicious hosts can impersonate non-malicious hosts, this attack succeeds.

   Migration source impersonation is not possible. It would just create a new Alliance. Message manipulation in transit is not possible, since the transmissions are secured end-to-end. As long as both end hosts are not impersonated and not malicious, no undetected message corruption is possible. If too many hosts are malicious the scheme breaks down in other places anyway.

8. The computations done here are mainly triple generation for the distributed computations and re-sharing. The same discussion as for tree node 5 applies. See sections 6.5 and 6.3.

9. The originator can be impersonated. However, if the Alliance encrypts the result data in such a way that only the originator can decrypt it, originator impersonation degenerates to conventional DoS and does not compromise confidentiality.

10. Alliance impersonation is feasible. A possible effective countermeasure is to embed a unique, random ID in the shared (secret, dynamic) data of the Alliance. An impersonator cannot get this ID and hence Alliance impersonation fails.

11. The aim is to increase the relative number of malicious hosts in order to make a *bad choice* more likely. By *bad choice* it is meant that more than $t(n)$ malicious migration targets are selected. The possibilities are to have more malicious hosts, less honest hosts (general DoS) or convert honest hosts into malicious ones.

12. Bad random selection of the migration targets is a problem, see Section 6.5.

13. Since the distributed computations are secure, this attack is not possible.

14. This attack is possible, depending on the host authentication scheme used, i.e. the condition a host has to fulfil in order to be a valid migration target. See Section 6.4 for a detailed discussion of some options.

15. This type of attack is similar to a general DoS attack. It can succeed, yet it can be made harder, as explained in Section 6.3.

16. The possibility of host compromise depends on host security and is out of the scope of this thesis. Note however that host compromise can lead to a retroactive *bad choice* where a host was honest before and during migration and only became malicious after an agent had been migrated to it or possibly even later when agent data is not deleted securely on the old hosts after migration.

We have seen that there are several types of specific security problems that can cause Alliance compromise. Most are caused by agent and Alliance migration and are discussed in the following sections. In addition, there are some not-quite obvious ways to use DoS attacks as part in an Alliance compromise attempt.

6.2 Replay Attacks

In a replay attack, the attacker re-uses a copy of an Alliance member it hosted before or got by a man-in-the-middle attack. She then fakes an execution request (or even lots of them) and sends it to candidate servers. This attack could be used to bind the resources of the target server(s). In addition, it could aim at obfuscating the Alliance.
In the version of the SMAC protocol presented in this thesis, this attack is made impossible by providing each Alliance member with some additional signed data, namely an agent identifier, a migration counter and an expiration date. This data allows a candidate host to check whether this specific agent has already been hosted at the same migration cycle of the Alliance. In case the expiration date has not yet been reached, the execution request is a replay attack. Otherwise, the identicalness of agent identifier and migration counter happened by chance. The agent is obviously a member of another Alliance. So, the candidate host can reject malicious execution requests and avoid to be flooded with copies of such an agent.

6.3 Denial of Service

DoS attacks can be used to facilitate Alliance compromise. Of course, DoS can also be used as a direct sabotage-type attack on an Alliance. DoS attacks on one particular Alliance could be used to prevent it from obtaining a specific piece of information, from reporting something back to the originator or from executing some action. As an application of this type of DoS, consider for example a vendor that does not want to honour an offer made to an Alliance earlier.
Generally DoS attacks are very hard to defend against. Still, in many cases something can be done to make them harder or less worthwhile. In the case of mobile Alliances, one option is to detect DoS-like conditions and report them back to the originator of the Alliance. Care needs to be taken to distinguish network problems from attacks. A second option is to increase the effort needed for DoS attacks.

6.3.1 Detect and Report

Detection of network problems or insufficient local resources is easy. Timeouts on computations and communication attempts are sufficient in most cases. Resource exhaustion attacks on hosts executing the agents can also be detected in this way. An agent can assume that the environment is too hostile for further operation when the number of accumulated timeouts exceeds a certain threshold. In this case the agent could notify the originator. When the originator gets this type of notification from at least $t_(n) + 1$ honest agents, the originator knows that the Alliance has stopped to be functional. Since the number of malicious

agents in an Alliance is not larger than $t(n)$, DoS attacks where the agents running on malicious hosts claim an error condition are only possible when Alliance compromise is feasible anyway.

6.3.2 Increase DoS Tolerance

If a DoS attack directly targets individual hosts with agents on them, these hosts are rendered unusable and unreachable in many cases. In the strict model, DoS on even a single honest host allows Alliance compromise, if the maximum permitted number of hosts is already malicious. This is due to the fact that non-responding agents need to be treated as malicious, since they cannot contribute any correct data to computations and majority votes. One way to deal with this problem is to decrease the number of responding (and thereby not readily identifiable) malicious host that are allowed in an Alliance. If, for example, the number of allowed responding malicious hosts is decreased to $\frac{n}{6}$, then DoS attacks on up to $\frac{n}{6}$ of the hosts executing an Alliance can be tolerated. See Section 6.5 for more details.
A connectivity disruption DoS attack can be made more difficult. There are two variants of this attack: Disrupt connectivity between the agents in a computation phase and disrupt connectivity to target hosts in agent migration. In the simple model every agent has to have direct connectivity to every other agent and migration target. An Alliance can employ internal dynamic routing of messages that is hard to understand from the outside. It then forms a *dark net* [BEPW02]. Cutting off an agent's connectivity to the Alliance now requires cutting off its connectivity to all other agents. For the case of migration it requires cutting off the connectivity from all agents to a migration target. Depending on the concrete network infrastructure this may or may not be harder to do than just cutting of some connections.
Alliance-internal routing does not cause additional security problems. Messages are routed over an untrusted network anyway. Adding some possible untrusted members of the alliance in the chain of routers causes no new risks. An added benefit of Alliance-internal routing is that it increases the robustness against unreliable networks.

6.4 Certificates

Agents of an Alliance are executed on hosts that supply a special execution environment with standardised functionality. An important degree of freedom in the design of such frameworks is the question of certificates, i.e. how hosts running an agent framework can demonstrate their identity. Several alternatives exist. Each has a different impact on practicability of the overall system, effort needed and security level achieved.
The main use for host identification is the prevention of an attack where the attacker creates a massive number of (possibly virtual) malicious hosts that run the agent framework. Identity is also beneficial in the creation of secure channels between hosts.
The main possibilities are the following:

1. *Use Certificates.* Each participating host gets a certificate from one of possibly several well known authorities that allow the host to prove its identity

and also allows to prevent impersonation attacks when establishing secure connections to other hosts running the framework.

In addition, this option gives some possibility to detect the *malicious host creation*-attack in the certification authorities. Still, if one or more of the certification authorities are malicious, this scheme breaks completely.

The most serious drawback of this option is the need for non-distributed, trusted infrastructure. The cost and effort needed to establish and operate this infrastructure may well be prohibitive.

2. *Web-of-trust.* A PGP-like web-of-trust (see e.g. [Fei02]) can be used. This could be done by having certificates that are signed by other hosts running the framework. Whether this will prevent the creation of significant numbers of malicious hosts depends strongly on the concrete details of the scheme used.

3. *Do not use certificates.* This is a least-effort least-security solution. There may still be some weak authentication, namely by IP address, i.e. by reachability. If an attacker cannot intercept most/all packets sent between two specific IP addresses, there is the possibility to establish secure end-to-end communication between two specific IP addresses.

   While this alternative offers little protection against the creation of large numbers of (possibly virtual) malicious hosts, such an attack can be made more difficult. One possibility is to disallow the selection of migration targets with IP addresses that are close to each other or from the same subnet. In [RP02] Rennhard and Plattner describe a *collusion detection* mechanism along these lines. The reason this is effective to some degree is that physical subnets on the Internet can only be allocated in larger portions. The underlying reason is that routing in the Internet is not done on individual addresses but on target address prefixes which correspond to subnets of groups of subnets that are physically close to each other.

   With the IP address based countermeasures, an attacker can not simply take a class B subnet [Pos81] and install 65,534 malicious hosts[1]. Instead, a large number of smaller subnets has to be obtained or addresses in many subnets have to be connected to malicious hosts. This is very expensive, since either hosts physically connected to the individual subnets are needed or virtual channels have to be created from the individual IP addresses to a central pool of (virtual) malicious hosts.

   Still, attacks that create large numbers of malicious hosts are feasible if the attacker does not care about legality or about doing a huge amount of collateral damage. A worm could be used to compromise a large number of well distributed hosts that could then serve as malicious hosts within an agent framework. 100,000 and more compromised hosts in worm attacks are feasible today.

   Whether this type of attack is visible enough to be detected remains to be seen. The first attempts at 'stealth worms' that do not significantly impact host functionality or performance, and thereby limit the motivation of host operators to deal with them, have already been observed.

[1] At least two addresses are needed for network purposes and cannot be used as host addresses.

6.5 Numbers Matter

In the absence of agent migration, a secure distributed computation needs only be concerned with the number of malicious hosts in the set of hosts performing the computation. A limit on the number of malicious hosts is sufficient to model the requirement for a secure computation.
With agent migration the problem becomes more difficult. If there are enough malicious hosts in the set of migration target hosts to choose from, a *bad choice* can happen that selects more malicious targets than the scheme for secure distributed computation can tolerate. The question of Alliance compromise by malicious hosts becomes a matter of the target selection process and if it is random or has random components, a question of probability.

Bad Choice

Now, an analysis of the chances of alliance compromise by bad random selection of migration targets is presented. The problem exists in a scenario where the number of malicious hosts is constant ("attacker-less bad choice") as well as in the cases where the relative number of malicious hosts has been increased by an attacker ("bad choice with attacker"). For the analysis there is no real difference. It is just important to keep in mind that the number of malicious hosts does not need to be stable during the Alliance lifetime.
The following analysis assumes a static number of malicious hosts. In a scenario with dynamically changing malicious host numbers, the following limits can still be used as upper bounds. Special scenarios might need more specialised models.
To solve the problem for random selection, the overall fraction of malicious hosts needs to be lower than the fraction of malicious hosts permitted in the Alliance. In addition, the number of migrations m has to be limited, since the risk of selecting too many malicious hosts exists in each migration.
Let now k_m be the number of malicious hosts in the set of k participating hosts (i.e. hosts that operate an agent framework) and n the size of the Alliances as before. Then, the probability of sending exactly i agents to malicious hosts and $n-i$ agents to honest hosts is (for $i \leq n, k - k_m \geq n$):

$$P(i \text{ mal. hosts selected}) = \frac{\binom{k_m}{i} \cdot \binom{k-k_m}{n-i}}{\binom{k}{n}}$$

With the selected protocols the number of malicious hosts needs to be smaller than $t(n) = \lceil \frac{n}{3} \rceil - 1$. Hence the probability of having selected up to the maximum number of allowed malicious hosts is now:

$$P(\text{max. } t(n) \text{ mal. hosts selected}) = \frac{\sum_{i=0}^{t(n)} \binom{k_m}{i} \cdot \binom{k-k_m}{n-i}}{\binom{k}{n}}$$

For m migrations we get the probability P_t of having no more than $t(n)$ corrupted agents in the Alliance after each migration step as follows:

$$P_t = \left(\frac{\sum_{i=0}^{t(n)} \binom{k_m}{i} \cdot \binom{k-k_m}{n-i}}{\binom{k}{n}} \right)^m$$

This means P_t is the overall probability that the Alliance is not compromised because of bad host choices in migration. In each migration the agents are securely re-created and consequentially corrupted agents can not move from one host to the other, hence the argument is valid. For the case $k_m \leq t(n)$ we get $P_t = 1$ because there are not enough malicious hosts available to break the system.

Bad Choice Examples

Bad choice is a real problem, as will be illustrated now with some concrete examples. In the following figures Maple [Inc] is used to plot P_t for $k = 100$ and $k = 10,000$ participating hosts and Alliances of sizes of $n = 10$ (Figures 6.2, 6.4) and $n = 50$ (Figures 6.3, 6.5). The number of malicious hosts k_m was set to $\frac{k}{2}$, since for more malicious hosts, P_t is always very close to 0 in our examples. Choosing $k = 100$ and $n = 10$ we get $P_t \approx 0.921$ for having a non-compromised Alliance after 10 migrations if up to 10 hosts (10%) are malicious. For $n = 50$ we improve the result to $P_t \approx 0.996$ for $k_m \leq 20$ (20% malicious hosts). Even with 25% malicious hosts we still have $P_t \approx 0.7245$.

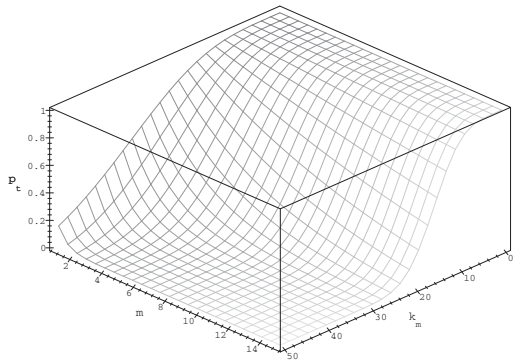


Figure 6.2: Alliance compromise with $k = 100$ and $n = 10$

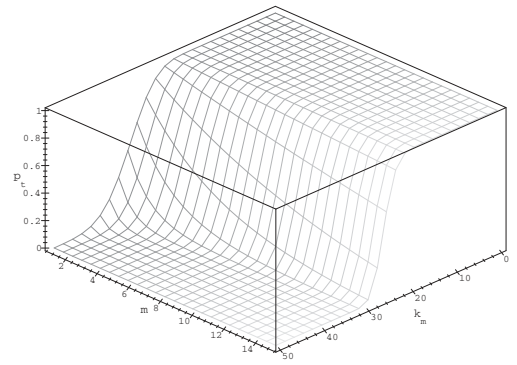


Figure 6.3: Alliance compromise with $k = 100$ and $n = 50$

As one can see in Figures 6.4 and 6.5, the results for $k = 10,000$ are a bit worse.

For instance in case of $n = 10$, $k_m = 1000$ and 10 migrations we get $P_t \approx 0.8796$. And, as expected for $n = 50$ and $k_m = 2000$ we have $P_t \approx 0.8665$. If we reduce k_m to 15% we get $P_t \approx 0.9936$.

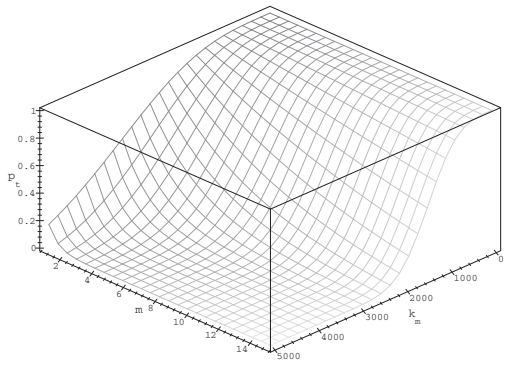


Figure 6.4: Alliance compromise with $k = 10,000$ and $n = 10$

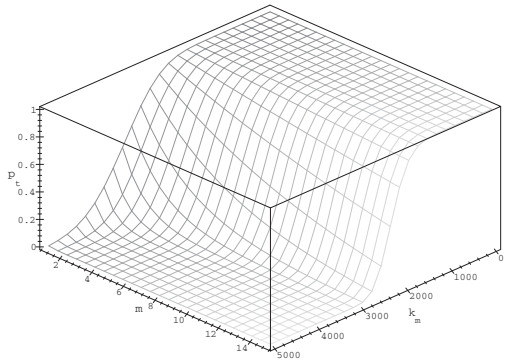


Figure 6.5: Alliance compromise with $k = 10,000$ and $n = 50$

The following tables 6.1, 6.2 and 6.3 demonstrate the necessity of choosing n big enough. Since Maple has precision issues when the values approach 0 or 1, the tables where calculated using the GNU MP [Gnu] library with a floating point precision of 1024 bits. Entries are rounded in the last digit.
As a general observation, larger Alliances are more secure, which is not surprising. It can also be seen that pretty large numbers of malicious hosts can be tolerated. These results demonstrate the practical feasibility of the Alliance approach. Random choice is the simplest method to select new hosts. If one takes into account to import a more sophisticated algorithm into the model, the results could improve considerably.

$k = 100$			
$n = 10 \;\; (t(n) = 3)$			
$m \backslash k_m$	10	20	30
10	0.9207	0.3133	0.0143
30	0.7805	0.0308	≈ 0
100	0.4378	≈ 0	≈ 0
$n = 30 \;\; (t(n) = 9)$			
10	$1-2{\cdot}10^{-5}$	0.7322	0.006
30	$1-5{\cdot}10^{-5}$	0.3925	≈ 0
100	0.9998	0.0443	≈ 0

Table 6.1: $k = 100$

$k = 1,000$				
$m \backslash k_m$	10	30	100	300
$n = 10 \;\; (t(n) = 3)$				
10	$1-1{\cdot}10^{-5}$	0.9988	0.8833	0.0135
30	$1-3{\cdot}10^{-5}$	0.9963	0.6893	≈ 0
100	$1-1{\cdot}10^{-4}$	0.9878	0.2893	≈ 0
$n = 30 \;\; (t(n) = 9)$				
10	$1-1{\cdot}10^{-15}$	$1-2{\cdot}10^{-8}$	0.9965	0.005
30	$1-3{\cdot}10^{-15}$	$1-7{\cdot}10^{-8}$	0.9895	≈ 0
100	$1-1{\cdot}10^{-14}$	$1-2{\cdot}10^{-7}$	0.9655	≈ 0
$n = 100 \;\; (t(n) = 33)$				
10	≈ 1	≈ 1	$1-2{\cdot}10^{-11}$	0.0957
30	≈ 1	≈ 1	$1-5{\cdot}10^{-11}$	0.0009
100	≈ 1	≈ 1	$1-2{\cdot}10^{-10}$	≈ 0

Table 6.2: $k = 1,000$

$k = 10,000$			
$m \backslash k_m$	30	100	300
$n = 10 \;\; (t(n) = 3)$			
10	$1-1{\cdot}10^{-7}$	$1-2{\cdot}10^{-5}$	0.9986
30	$1-4{\cdot}10^{-7}$	$1-6{\cdot}10^{-5}$	0.9957
100	$1-1{\cdot}10^{-6}$	0.9998	0.9857
$n = 30 \;\; (t(n) = 9)$			
10	$1-3{\cdot}10^{-18}$	$1-2{\cdot}10^{-12}$	$1-9{\cdot}10^{-8}$
30	$1-1{\cdot}10^{-17}$	$1-5{\cdot}10^{-12}$	$1-3{\cdot}10^{-7}$
100	$1-3{\cdot}10^{-17}$	$1-2{\cdot}10^{-11}$	$1-9{\cdot}10^{-7}$

Table 6.3: $k = 10,000$

Chapter 7

From Theory to Practice

7.1 Introductory Remarks

Secure multi-agent computations live within an arithmetic circuit over a finite field $\mathbb{F}$, knowing only addition, multiplication and multiplication with a public scalar as described in section 4.3.2. In [Gál95] it is shown that a polynomial size semi-unbounded fan-in Boolean circuit of depth d over an arbitrary finite field can be simulated by a polynomial size semi-unbounded fan-in arithmetic circuit of depth $O(d + \log n)$ (where n is the number of variables of the Boolean function f that is implemented by the Boolean circuit).

Definition 7.1. *Semi-unbounded boolean circuits A Boolean circuit is defined as a circuit with gates from the Boolean basis* $\{\wedge, \vee, \neg\}$*. A* semi-unbounded boolean circuit *is a boolean circuit with constant fan-in* $\wedge$ *gates and unbounded fan-in* $\vee$ *gates.*

For practical algorithms this is no restriction since both computational power is finite and the implemented functions should terminate and thus not have an infinite number of input variables. This implies the theoretical possibility of implementing all functions that one could think of in an agent scenario. I call it a *theoretical possibility* as secure multi-agent computations are on the one hand very secure but on the other hand also very expensive (concerning both computation and communication costs). This will be shown in detail in this section when a digital signature is realized through SMAC. So, no-one would propose to use secure multi-agent computation for each and every application. Instead, this kind of computation should be used in the following cases:

1. In a scenario that requires a very high security and confidentiality level all computations should be done distributed and all data should be shared. In such a scenario the gain is worth the expenses.

2. SMAC can be used without restriction for functions that require only a small number of multiplications since those operations cause the most communication and computation costs.

3. While designing an agent it is wise to decide which sub-tasks are the most sensitive and to implement those together with the data connected to them using SMAC. Other functions can be implemented as usual.

7.2 What Language do Alliances understand?

Secure multi-agent computation should guarantee not only robustness but also data privacy. Arithmetic circuits do not speak *Javanese*. Besides complex operations defined in libraries, Java also offers the usual control structures such as `if-then-else` or `while`. If a designer would simply substitute additions and multiplications by distributed ones, data privacy would be terribly hurt. Control structures can leak a considerable amount of information on shares that can be used by side channel attacks. In [SGE02] we have shown that a binary secret key k can be completely uncovered when evaluating a multi-variate polynomial at point k just by measuring the time necessary for evaluation. Recently, such attacks are well-known and popular and are caused by the possibility of measuring the execution time of loops and `if-then-else` structures. This allows inference on the expressions used as Boolean control mechanisms for the control structures. Consequently, this implies for secure multi-agent computations that such structures may not be used as usual. All computations and evaluations must be lift from the single-agent level to the Alliance level as illustrated in figure 7.1. On the single-agent level an Agent A makes its computations i.e. executing function f on an input x in an unsecured way. The input x can be read and manipulated by H. Instead, a functor lifts x on the Alliance level by splitting it into shares using the function "share". On the Alliance level f is turned into a distributed function Mf using only shared input x_s. Once an input is shared and processed on this level, no return is possible (from point of view of one agent). The resulting commutating diagram describes a monad from category theory. It also shows the main task arising in building Alliances: lifting arbitrary functions f to Mf. I have firstly introduced this problem in chapter 5 in figure 3.1 where the so-called *protocol compiler* takes care of this task. Actually, the design of this compiler requires to develop a new machine model, that can handle control structures using for example comparisons of variables in an efficient manner without lacking any information on the input variables. A native approach requires to break all input down into bits and to perform a bitwise comparison. This is a very inefficient approach and not suitable for practice. Besides lifting computations on the Alliance level one still has

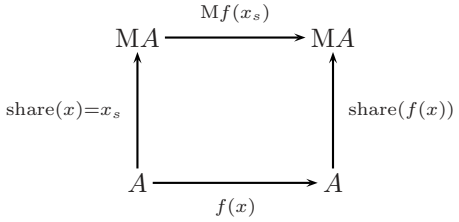


Figure 7.1: The SMAC monad

to prevent from information leakage and side-channel attacks. Additional methods must be introduced in order to simulate a normal computer program, as to say always executing all branches of an `if-then-else` structure or passing through all loops as often as a predefined upper limit requires. This, too, raises the computation and communication costs on the Alliance level in a significant manner.
Developing an (hopefully) efficient and secure machine model for all functions

that could be computed by arithmetic circuits requires methods from circuit optimisation, a field that is heavily worked on but still does not provide generic methods that would allow to simulate a program written in a higher programming language in a way suitable for the Alliance approach. Exemplary solutions will be given in chapter 8 in which I present some implementation details and a detailed complexity analysis of an RSA digital signature and of the symmetric cipher AES.
In the following sections I present a secure simulation of conditional statements and loops which are the control structures that are heavily used by higher programming languages. Furthermore, long numbers on an arithmetic circuit together with the necessary functions are defined. For all operations the number of distributed multiplications serves as complexity measure and is given as exact number.
The terminology is as follows. Arithmetic expressions that are not yet lifted to the monad and evaluated are denoted with capital letters A_i. Small letters like a, b stands for evaluated arithmetic expressions which are not distributed. A leading "M" lifts an expression to the monad. Finally, the function "Meval" evaluates distributed arithmetic expression within the monad, i.e. privately and robustly.

7.3 Simulation of Conditional Branches

There is no possibility to implement a conditional branch in an arithmetic circuit as in any evaluation process all gates are evaluated. Each gate gets input and produces output. This cannot be suppressed. In normal programming languages there are control structures which use control variables. Thus, by analysing the control flow of a program, information, e.g. about the control variables, gets lost. This must be prevented.
As an example, take a look at the following simple example:

$$\texttt{if } b \texttt{ then } x \leftarrow A_1$$

By analysing the control flow one can determine whether the statement $x \leftarrow A_1$ has been executed or not. Thus, the condition b is revealed. In case of a modular exponentiation using the square-and-multiply algorithm from section 9 the consequences are crucial. Computing an RSA digital signature, a binary representation of the private key serves as exponent for a modular exponentiation. Each bit of the private key serves as condition for an `if-then-else` statement. Thus, by an analysis of the control flow, the whole private key is revealed. To prevent from this, it is necessary to *complete* conditional statements, which means that missing branches are anyhow created.
But how can this be done? A completion is reached by statements that do not change the state of the variables. In the example above, the solution is to introduce an `else` branch with the command $x \leftarrow x$. Thinking of a normal program, this still allows a side-channel attack as the computation of a regular command will take much more time as the assignment $x \leftarrow x$. A modern compiler will even recognise and delete such meaningless statements. However, simulating the branching and having distributed variables the situation is different as one can see in example 7.2.

Example 7.2. *The conditional statement*

$$\textit{if } b \textit{ then } x \leftarrow A_1$$

is completed and lifted to

$$Mx = Meval(Mb \cdot MA_1 + (1 - Mb) \cdot Mx).$$

As x and b are distributed, the shares $\mathrm{M}x$ and $\mathrm{M}b$ are arbitrary field elements. Thus, the product $(1 - \mathrm{M}b) \cdot \mathrm{M}x)$ to be computed in the simulation of the `else` branch, is not trivial.
In general, the simulation of a conditional branch is done as in definition 7.3.

Definition 7.3. *Let b be a Boolean value and A_1, A_2 arithmetic instructions. Then, the branching*

$$\textit{if } b \textit{ then } x \leftarrow A_1 \textit{ else } x \leftarrow A_2$$

can be lifted to the SMAC monad as follows:

$$Mx = Mb \cdot MA_1 + (1 - Mb) \cdot MA_2$$

The multiplicative complexity of such a branching is

$$\mathcal{M}(\mathtt{if}_{A_1,A_2}) = \mathcal{M}(A_1) + \mathcal{M}(A_2) + 2.$$

Thereby, $\mathcal{M}(A_1),\ \mathcal{M}(A_2)$ defines the number of distributed multiplications necessary to evaluate the arithmetic expressions $A_1,\ A_2$ in the monad. The analysis and optimisation of general branching structures of larger depth is out of scope of this thesis. Solutions can be found in standard works as [GW84] in the field of compiler construction.

7.4 Simulation of Loops

A loop construct consists of a block of code and a condition. Again, the latter does not exist in an arithmetic circuit. Therefore, the simulation requires a trick. The while loop can be thought of as a repeating `if` statement without an `else` branch. If one knows an upper bound k of the number of executions a simulation of

```
1 while b do
2 |_ x ← A_1
```

is given by the sequence

$$\begin{aligned}
\mathrm{M}x &= \mathrm{M}b_1\mathrm{M}A_1 + (1 - \mathrm{M}b_1)\mathrm{M}x, \\
\mathrm{M}x &= \mathrm{M}b_2\mathrm{M}A_2 + (1 - \mathrm{M}b_2)\mathrm{M}x, \\
&\vdots \\
\mathrm{M}x &= \mathrm{M}b_k\mathrm{M}A_k + (1 - \mathrm{M}b_k)\mathrm{M}x.
\end{aligned}$$

The condition b_i contains the result of the evaluation of B in the ith iteration. A purely arithmetic representation of the resulting (reconstructed) x is

$$b_k A_k + (1-b_k)(b_{k-1}A_{k-1} + (1-b_{k-1})(\cdots(b_2 A_2 + (1-b_2)(b_1 A_1 + (1-b_1)A_1))\cdots))$$

Obviously, the number of distributed multiplications is

$$\mathcal{M}(\texttt{while}_k) \leq 2k + k \cdot \mathcal{M}(B) + \sum_{i=1}^{n} \mathcal{M}(A_i).$$

7.5 Long Numbers

As public key (PK) cryptography is always based on mathematical problems, which are (hopefully) difficult to solve. One category of cryptographic algorithms for instance is characterised by the so-called *Diffie-Hellman assumption*. All algorithms in this category can be broken as soon as there is an efficient algorithm to compute the discrete algorithm in large finite fields. Another category contains algorithms that are based on the difficulty to factorise large numbers. There are other categories as for example algorithms based on solving non-linear equations over $\mathbb{F}_q[X_1, \ldots, X_n]$, but there is one common property: all PK-algorithms require large finite rings or fields to provide a satisfactory security level.
For mobile agents it is of mayor concern to have some cryptographic functions (as e.g. a digital signature) at hand that can be securely evaluated. Therefore, chapter 8 provides a complexity analysis of RSA and AES. The first requires a definition of long numbers on the arithmetic circuit over the field $\mathbb{F}_p$.
Each n-digit long number a is represented in a B-adic form, i.e.

$$a = \sum_{i=0}^{n-1} a_i B^i, \quad 0 \leq a_i \leq B-1.$$

The results of all basic operations (addition, multiplication and its inverses) applied on two digits is smaller than or equal to B^2-1. For reasons of uniqueness of the results, it must be ensured, that during the computation no modulo reduction in the field $\mathbb{F}_p$ is necessary. Therefore, we have the following inequality for the size of B:

$$B^2 - 1 < p \Leftrightarrow B^2 < p+1 \Leftrightarrow B < \lfloor\sqrt{p+1}\rfloor$$

The largest possible basis B for a long number representation in the field $\mathbb{F}_p$ is thus $B = \lfloor\sqrt{p+1}\rfloor - 1$.
Implementing a long number arithmetics on an arithmetic circuit requires the following operations:

1. Division with remainder on short numbers:

$$\begin{array}{rccl} \texttt{div:}\ \mathcal{B}\times\mathcal{B} & \rightarrow & \mathcal{B}\times\mathcal{B} & \\ (a,b) & \mapsto & (q,r) & (\text{with } a = b\cdot q + r,\ r < q) \end{array}$$

2. Addition and multiplication of two short numbers with carry digit.

3. Comparison of two short numbers ($<, >, =$).

4. Division of a 2-digit number by a 1-digit number:

$$\begin{array}{rcl} \texttt{div}\colon \mathcal{B}\times\mathcal{B}\times\mathcal{B} & \to & \mathcal{B}\times\mathcal{B} \\ (a1, a_0, b) & \mapsto & c_1c_o \ \ (\text{with } c_1c_0 = \lfloor (a_1\cdot B + a_0)/b \rfloor,\ c_0, c_1 < B) \end{array}$$

   This operation is required to determine the carry digit and remainder after multiplication of two digits.

The following sections present several operations. First, operations on the single digits, the so-called *short numbers*, are introduced and used as a basis for the definition of the long number operations in later sections. The notion is as follows: All operations are written in teletype font (e.g. `op`). The number of distributed arguments is attached to the end of the functions name (e.g. `op1`). If it is a long number operation there is a leading `l` (e.g. `lop1`).

7.5.1 Basic Arithmetics with Short Numbers

Any operation on long numbers requires to have methods at hand to reduce intermediate results coming from operations on each digit with respect to basis B. These are div, mod, addition and multiplication of short numbers which have to be defined with respect to the underlying field $\mathbb{F}_p$. As these operations are a building block for the actual long number operations they are called very oftenly and should cause as few distributed multiplications as possible. This implies that a straight-forward implementation is out of the question. An approach using linear shift registers has also turned out to be not general enough, as they require a very specific modulus. The method which is finally used, is a function interpolation. The computations are done over a finite field and thus all functions can be interpolated by a polynomial.
These polynomials can be pre-computed by the originator of an agent Alliance. The interpolation polynomial of an arbitrary n-ary function requires the pre-computation of all possible points (as to say p^n many). Then, proposition 4.2 gives us an interpolation polynomial q of degree $d \leq p^n - 1$. Represented in Horner form and given to the agents, the evaluation of q causes $p^n - 2$ distributed multiplications.
For the implementation of long numbers, addition and multiplication are defined digit wise, so-called *short number operations*. In order to store eventual carry digits, one requires the functions div and mod to transform the computation results in B-ary form. Thus, for $a, b < B$ one requires $c_1, c_2, c_1', c_2' < B$ with:

$$a + b = c_1B + c_0, \quad a * b = c_1'B + c_2'$$

It is obvious, that $c_1 \in \{0, 1\}$ while $c_1' \in \{0, \ldots, B-2\}$. This difference in the domain size drastically influences the multiplication complexity of the interpolation polynomial of div and mod. Therefore, it is reasonable to define functions on different domains:

$$\begin{array}{rcl} \texttt{div1} : \{0, \ldots, B^2 - 2\} & \to & \{0, \ldots, B-1\} \\ x & \mapsto & x \text{ div } c_1 \end{array}$$

and

$$\begin{array}{rcl} \texttt{mod1} : \{0, \ldots, B^2 - 1\} & \to & \{0, \ldots, B-1\} \\ x & \mapsto & x \bmod B = c_2 \end{array}$$

for reductions after a short number multiplication, and

$$\begin{array}{rcl} \texttt{rediv1} : \{0,\ldots,2B-1\} & \to & \{0,1\} \\ x & \mapsto & \lfloor x/B \rfloor \end{array}$$

and

$$\begin{array}{rcl} \texttt{remod1} : \{0,\ldots,2B-1\} & \to & \{0,\ldots,B-1\} \\ x & \mapsto & x \bmod B = c_2' \end{array}$$

for reductions after a short number addition. The argument of the `remod1` is smaller or equal to $2B-1$ as the long number addition is a cascading addition and thus has not only to consider a and b but also a carry from the addition step before. All operations are defined for one distributed argument only, as one can assume the basis B to be public. This is just for cases of consistency, as the number of distributed arguments does not influence the degree of the resulting interpolation polynomial.
Unfortunately, all these functions are specific in the sense that they are not differentiable. The function `mod` n for example is linear and then suddenly falls to zero. In order to interpolate such functions the interpolating polynomial has a high degree. Most probably, is has the maximum degree $p^n - 1$ when computing over the field $\mathbb{F}_p$.
Consequently, we get the following complexities as upper bounds:

$$\begin{aligned} \mathcal{M}(\texttt{div1}) = \mathcal{M}(\texttt{mod1}) &= B^2 - 2 \\ \mathcal{M}(\texttt{rediv1}) = \mathcal{M}(\texttt{remod1}) &= 2B - 2 \end{aligned}$$

Oftenly, the computation of the quotient as well as the remainder are required. If the quotient q of a/b has already been computed, the remainder can be determined locally by $a \bmod b = a - q \cdot b$. The function computing both quotient and remainder on the domain $\{0,\ldots,B^2-1\}$ is called `divmod1`, the one on the reduced domain `redivmod1`. The number of distributed multiplication is then[1]

$$\begin{aligned} \mathcal{M}(\texttt{divmod1}) = \mathcal{M}(\texttt{div1}) &= B^2 - 2 \\ \mathcal{M}(\texttt{redivmod1}) = \mathcal{M}(\texttt{rediv1}) &= 2B - 2 \end{aligned}$$

Addition and Multiplication

Addition of two short numbers is a local operation but requires one reduction modulo B to determine the quotient (carry) and the remainder (`redivmod1`). Multiplication, if not with public argument, is itself a distributed multiplication. The result, too, requires to be reduced modulo B. This can be done with `divmod1`. Consequently, the complexity is

$$\begin{aligned} \mathcal{M}(\texttt{add2}) = \mathcal{M}(\texttt{redivmod}) &= 2B - 2 \\ \mathcal{M}(\texttt{mul2}) = \mathcal{M}(\texttt{divmod1}) + 1 &= B^2 - 1 \\ \mathcal{M}(\texttt{mul1}) = \mathcal{M}(\texttt{divmod1}) &= B^2 - 2 \end{aligned}$$

Comparison

The operators $<, >, \leq, \geq$ can be computed distributedly by means of a division by B. The corresponding arithmetic expressions are given in table 7.5.1. With

[1] For a more detailed analysis see [Amm07].

Operation	Arithmetic expression
$a > b$	$\texttt{rediv1}(B-1+a-b)$
$a \geq b$	$\texttt{rediv1}(B+a-b)$
$a < b$	$\texttt{rediv1}(1-((B+a-b))$
$a \leq b$	$\texttt{rediv1}(1-((B-1+a-b))$

Table 7.1: Arithmetic expressions for short number comparisons

$a, b < B$ the arithmetic expressions for $a > b$ and $a \leq b$ lie in the interval $\{0, \ldots, 2B-2\}$, and those for $a \geq b$ and $a < b$ in $\{1, \ldots, 2B-1\}$. So, the reduced div operation `redivmod1` can be used.
The operators $=, \neq$ can be computed by the equivalence

$$a = b \Leftrightarrow a \leq b \wedge a \geq b.$$

The costs are then

$$\mathcal{M}(=, \neq) = 4(B-1) = 2 \cdot \mathcal{M}(>, <, \geq, \leq)$$

It can also be computed by an interpolation polynomial of degree $2B-2$, requiring $2B-3$ multiplications.
These results allow to assign the generic term `cmp` to the operations in the set $\{<, >, \leq, \geq, \neq, =\}$ and

$$\mathcal{M}(\texttt{cmp}) = 2B-2.$$

7.5.2 Basic Arithmetics with Long Numbers

This section introduces the implementation of basic arithmetics on an arithmetic circuit with long numbers as well as a complexity analysis. The methods are mainly based on the standard algorithms from [Knu98, Wel05, CLRS01], with modifications to handle the information leakage problem. From now-on, a, b be long numbers in an B-ary representation.

If-Then-Else

The method from section 7.3 could also be applied to long numbers. But there is a more efficient way. For a long number $a = (a_{n-1}, \ldots, a_0)$, a condition $b \in \{0, 1\}$ and arithmetic expressions A_1, A_2 the branching

```
1 if b then
2   a ← A_1
3 a ← A_2
```

can be simulated by

$$(a_{n-1}...a_0) = b \cdot (r_{n-1}...r_0) + (1-b)(s_{n-1}...s_0)\ .$$

In this equation, the long numbers $r = (r_{n-1}, \ldots, r_0)$ and $s = (s_{n-1}, \ldots, s_0)$ are the results of evaluation A_1, A_2. The multiplications in this equation do not

cause carry digits. So, one can multiply the factors b and $1-b$ with each digit. Thus, the equation can be rewritten in the form

$$(a_{n-1}...a_0) = (b \cdot r_{n-1}...b \cdot r_0) + ((1-b)s_{n-1}...(1-b)s_0).$$

Therefore, the complexity is

$$\mathcal{M}(\texttt{lif}) = 2n.$$

Comparison

Algorithm 1 `lcomp2` computes for two long numbers a, b the distributed value (1 or 0) for $a < b$. Normally, one would start with the highest valued digit and stop as soon as there is a difference. As discussed in section 7.4 this is not possible in the SMAC monad. There, one has to start with the lowest-valued digit using using a short number comparison for each digit. Line 3 and 5 cause $\mathcal{M}(\texttt{cmp})$ multiplications. The branching costs two additional multiplications. Consequently, for n executions, the complexity is

$$\mathcal{M}(\texttt{lcmp}_n) = n(2\mathcal{M}(\texttt{cmp}) + 2) = n(4B - 2) = 4Bn - 2n.$$

```
   Input: a = (a_{n-1}...a_0), b = (b_{n-1}...b_0)
   Output: r = 1 if a < b, else r = 0
1  r ← 0;
2  for i ← 0 to n − 1 do
3  |  if a_i < b_i then
4  |  |  r ← 1;
5  |  if a_i > b_i then
6  |  |  r ← 0;
```

Algorithm 1: `lcomp2`

Addition

The method presented here is addition known from school. It is based on the operations `rediv1` and `remod1` from section 7.5.1 are used. It is presented in Algorithm 2. Obviously, one division per coefficient and one scalar multiplication

```
   Input: a = (a_{n-1} ... a_0), b = (b_{n-1} ... b_0),  a_i, b_i ∈ {0, ..., B − 1}
   Output: s = (s_n ... s_0) = a + b
1  c ← 0;
2  for i ← 0 to n − 1 do
3  |  s_i ← (a_i + b_i + c) mod B;
4  |  c ← (a_i + b_i + c) div B;
5  s_n ← c;
```

Algorithm 2: `ladd`

for the modulo operation is necessary. The overall multiplication complexity for two n-digit numbers is then

$$\mathcal{M}(\texttt{ladd}) = n \cdot \mathcal{M}(\texttt{redivmod}) = 2Bn - 2n = 2n(B-1).$$

Subtraction

Algorithm 3 computes the difference s of two long numbers a and b. The value c is the leading sign and defined as $c = 0$ if $a \geq b$, and $c = 1$ if $a < b$. Line 4 computes the actual borrow. In both lines 3 and 4 it is necessary to add the value B in order to guarantee the results to lie in the interval $[0, B-1]$. Otherwise, an underflow in the field $\mathbb{F}_p$ could occur and cause the div and mod operation to deliver wrong results. For the actual implementation of the subtraction algorithm it suffices to use the operations `rediv` and `remod` as we have

$$0 \leq B + a_i - b_i - c < 2B.$$

Thus, the complexity of subtracting two n-digit long numbers with basis B is

$$\mathcal{M}(\texttt{lsub}) = n \cdot \mathcal{M}(\texttt{redivmod}) = 2Bn - 2n = 2n(B-1).$$

Input: $a = (a_{n-1}...a_0), b = (b_{n-1}...b_0),\ a_i, b_i \in \{0, \ldots, B-1\}$
Output: $s = (cs_{n-1}...s_0) = a - b$
1 $c \leftarrow 0$;
2 **for** $i \leftarrow 0$ **to** $n-1$ **do**
3 $\quad s_i \leftarrow B + a_i - b_i - c \ mod\ B$;
4 $\quad c \leftarrow 1 - (B + a_i - b_i - c \ div\ B)$;

Algorithm 3: `lsub`

Multiplication

Algorithm 4 presents a method for multiplying long numbers which is similar to the method known from school. The school method consists of two steps. In the first step, all multiplications are done for each digit of one of the numbers. Then, the resulting numbers are added. In contrast to this, `lmul` does the multiplications and additions for each digit in parallel.
The body of the loop requires $\mathcal{M}(\text{divmod1}) + 1$ distributed multiplications. Having n and m the number of digits of the arguments, the body is executed $n \cdot m$ times. So, we get two cases:

1. Private multiplication of a distributed long number with a public one causes

   $$\mathcal{M}(\texttt{lmul1}) = mn\mathcal{M}(\texttt{divmod1}) = mn(B^2 - 2) = mnB^2 - 2mn$$

   distributed multiplications.

2. Private multiplication of two distributed long numbers costs

   $$\mathcal{M}(\texttt{lmul2}) = mn(\mathcal{M}(\texttt{divmod1}) + 1) = mn(B^2 - 2) = mnB^2 - mn$$

   distributed multiplications.

Input: $a = (a_{m-1}...a_0),\ b = (b_{n-1}...b_0),\ a_i, b_i \in \{0, \ldots, B-1\}$
Output: $w = a \cdot b = (w_{m+n-1}...w_0)$

```
1 w ← 0;
2 for i ← 0 to m − 1 do
3     c ← 0;
4     for j ← 0 to n − 1 do
5         t ← a_i · b_j + w_{i+j} + c;
6         c ← t div1 B;
7         w_{i+j} ← t mod1 B;
8     w_{j+m} ← c;
```

Algorithm 4: `lmul`

Short Division with Remainder

In the beginning of section 7.5 it is mentioned that it is necessary for an implementation of a division of long numbers to define a division of a two-digit by a one-digit number. This operation be called `shdiv` (short division). In a general long number division it is quite complex to determine how often the divisor fits into the leading one or two digits of the dividend. A short division allows to do this by a simple `div` operation. Algorithm 5 computes the quotient q and the remainder. The method equals the one known from school. The body of the

Input: $a = (a_{m-1}...a_0),\ b,\ a_i \in \{0, \ldots, B-1\}$
Output: $q = (q_{m-1}...q_0), r$ with $a = q \cdot b + r$

```
1 r ← 0;
2 q ← 0;
3 for i ← m − 1 to 0 do
4     q_i ← (a_i + rB) div b;
5     r ← (a_i + rB) mod b;
```

Algorithm 5: `shdiv`

loop is executed m times and requires $\mathcal{M}(\texttt{divmod1})$ multiplications. The overall complexity is thus

$$\mathcal{M}(\texttt{shdiv1}) = m\mathcal{M}(\texttt{divmod1}) = B^2m - 2m.$$

Since in this thesis only a reduction by the public basis B is required, the complexity analysis is limited to this case.
Now, a general long number division can be defined.

Long Number Division with Remainder

As mentioned before, the difficulty of a long number division is the necessity of estimating how often the divisor fits into a specific number.

Example 7.4. *In order to divide 355,938 by 427 the following steps are performed:*

1. *division of 3,559 by 427* $\rightarrow q = 8$ *and* $r = 133$,

2. *add one digit to r and divide 13333 by 427* $\rightarrow q = 3,\ r = 52$

3. *add one digit to r and divide 528 by 427* $\rightarrow q = 1,\ r = 101$.

Example 7.4 illustrates that a long number division can be reduced to a division of a $n+1$ digit number by an n digit number. While a human being is able to estimate the quotient q of such numbers efficiently, a computer needs help.

Proposition 7.5. *Be $u = (u_n \dots u_0), v = (v_{n-1} \dots v_0)$ long numbers with $u/v < B$. Then, the quotient q with $q = \lfloor u/v \rfloor$ can be estimated by*

$$\hat{q} = min\left(\left\lfloor \frac{u_n B + u_{n-1}}{v_n - 1} \right\rfloor, B - 1\right),$$

and for $v_{n-1} \geq \lfloor B/2 \rfloor$ we have the inequality

$$\hat{q} - 2 \leq q \leq \hat{q}.$$

Proof. [Knu98] □

In order to guarantee $v_{n-1} \geq \lfloor B/2 \rfloor$, [Knu98] proposes a normalisation by multiplying both u, v by $\lfloor B/(v_{n-1}+1) \rfloor$. This does not change the quotient of u/v and does not increase the number of digits of v. Now, a long number division can be defined. It is shown in Algorithm 6.
The complexity analysis is limited to the case of a public divisor. First, the specialities of the single steps are discussed. The number of multiplications in each line of algorithm 6 are then given in table 7.5.2.

1. "Normalise": line 1 is a local division of public values, lines 2,3 are multiplications with a public scalar.

2. "Computation of the next digit of q": the expression $Bc + a_{j+n-2}$ in lines 9,12 lies in the interval $[0, \dots, B^2)$. They are first brought into B-ary presentation and then compared.

3. "Multiplication and Subtraction": line 17 contains the assignment of a long number. This is handled as shown in section 7.5.2.

4. "Denormalise": requires `shdiv`.

Thus, the long number division results in a total of

$$\begin{aligned} \mathcal{M}(\texttt{ldiv}_{m,n}) &= mnB^2 + 3nB^2 + 7mB^2 + 6B^2 + 6mnB + 6nB \\ &\quad +28mB + 26B - 6mn - 6n - 20m - 18. \end{aligned}$$

Summary

Table 7.5.2 gives an overview on all operations needed in the context of a long number arithmetics. The complexity is measured in the number of distributed multiplications, whereby the parameters n, m define the number of digits of the long numbers a and b. The term "mult." in the second column stands for a distributed multiplication of two field elements.

```
   Input: a = (a_{m+n-1}...a_0),  b = (b_{n-1}...b_0),  a_i, b_i ∈ {0, ..., B − 1}
   Output: q = (q_m...q_0), r = (r_{n-1}...r_0) with a = q · b + r
   /* Normalise                                                          */
 1 d ← B div (b_{n-1} + 1);
 2 a ← d · a;
 3 b ← d · b;
 4 for j ← m to 0 do
       /* Computation of the next digit of q                             */
 5     q_j ← (a_{j+n}B + a_{j+n-1}) div b_{n-1};
 6     if q_j ≥ B then
 7         q_j ← B − 1;
 8     c ← a_{j+n}B + a_{j+n-1} − q_j b_{n-1};
 9     if q_j b_{n-2} > Bc + a_{j+n-2} then
10         q_j ← q_j − 1;
11         c ← c + b_{n-1};
12         if q_j b_{n-2} > Bc + a_{j+n-2} then
13             q_j ← q_j − 1;
       /* Multiply and subtract                                          */
14     (a_{j+n+1}...a_j) = (a_{j+n}...a_j) − q_j(0b_{n-1}...b_0);
15     if a_{j+n+1} = 1 then
16         q_j ← q_j − 1;
17         (a_{j+n}...a_j) = (a_{j+n}...a_j) + (0b_{n-1}...b_0);
   /* Denormalise                                                        */
18 (a_{n-1}...a_0) = (a_{n-1}...a_0) div d;
19 r ← (a_{n-1}...a_0);
```

Algorithm 6: $\texttt{ldiv}_{m,n}$

1	0	0
2	$\mathcal{M}(lmul1_{m+n,1})$	$B^2(m+n)-2m-2n$
3	0	0
4	$m+1$ times row 5 - 17	
5	$\mathcal{M}(\texttt{shdiv1}_2)$	$2B^2-4$
6	$\mathcal{M}(\texttt{cmp})$	$2B-2$
7	2	2
8	0	0
9	$2\mathcal{M}(\texttt{divmod1})+\mathcal{M}(\texttt{lcmp}_2)$	$2B^2+8B-8$
10	0	0
11	2	2
12	$2\mathcal{M}(\texttt{divmod1})+\mathcal{M}(\texttt{lcmp}_2)$	$2B^2+8B-8$
13	4	4
14	$\mathcal{M}(\texttt{lsub}_{n+1})+\mathcal{M}(\texttt{lmul1}_{1,n})$	$B^2n+2Bn+2B-4n-2$
15	$\mathcal{M}(\texttt{cmp})$	$2B-2$
16	2	2
17	$\mathcal{M}(\texttt{ladd}_{n+1})+\mathcal{M}(\texttt{lif}_{n+1})$	$2Bn+2B$
18	$\mathcal{M}(\texttt{shdiv1}_n)$	B^2n-2n

Table 7.2: **Complexity of long number divisions** given by the number of distributed multiplications in each line of algorithm 6.

Function	# short number operations	# multiplications
logical not		0
logical and/or		1
`if`		2
$\texttt{if}_{A_1,A_2}$		$\mathcal{M}(A_1)+\mathcal{M}(A_2)+2$
`div1`, `mod1`		B^2-2
`rediv`, `remod`		$2B-2$
`divmod1`	`div1`	B^2-2
`redivmod`	`rediv`	$2B-2$
`add`	`redivmod`	$2B-2$
`mul`	`divmod1`, 1 mult.	B^2-1
`cmp`		$2B-2$
$\texttt{lcmp}_n$	$2n \times$ (`cmp` + 1 mult.)	$4Bn-2n$
$\texttt{lif}_n$		$2n$
$\texttt{ladd}_n$	$n \times$ `redivmod`	$2Bn-2n$
$\texttt{lsub}_n$	$n \times$ `redivmod`	$2Bn-2n$
$\texttt{lmul1}_{m,n}$	$mn \times$ `divmod1`	$B^2mn-2mn$
$\texttt{lmul2}_{m,n}$	$mn \times$ `divmod1` + mn mult.	B^2mn-mn
$\texttt{shdiv1}_n$	$n \times$ `divmod1`	B^2n-2n
$\texttt{ldiv}_{m,n}$		$mnB^2+3nB^2+7mB^2+6B^2$ $+6mnB+6nB+26mB+26B$ $-6mn-10n-20m-18$

Table 7.3: Complexity of short/long number operations (in # distr. mult.)

Chapter 8

Applications

Obviously, secure multi-agent computations cause too much communication overhead to allow a practical implementation of a complete user-defined agent functionality. This will change with the development of new technologies providing more bandwidth to Internet users. Nonetheless, even today SMAC can be used.

1. A typical information collecting agent can be be substituted by an Alliance using a verifiable secret sharing scheme (VSS) as described in section 4.2.2 in order to keep important information secret and protected. The communication costs in this case are limited to a one-time information exchange between one agent and the rest of the Alliance. The program of such an Alliance would be written in a normal programming language, as for instance Java. If one agent wishes to distribute sensitive information among the Alliance it applies a VSS scheme and sends one share to each Alliance member. No computation on these shares is necessary, they are only a method to store data distributedly, redundantly and secretly.

2. It is also possible to design an Alliance member, that mainly consists of a program written in a conventional programming language. Additionally, they could be provided with arithmetic circuits for specific critical tasks. For instance, the agents could store distributed secret keys (private as well as symmetric ones) and use SMAC in order to sign contracts or to encrypt data in a robust way, without revealing information on the used key.

The first has been proposed by us in [EM06] for the field of file sharing in peer-to-peer systems. Along with supplementary measures this method provides a strong protection against censorship and legal prosecution. This is given by the guaranteed anonymity of clients and servers as well by distributing files over several servers in a way that makes the data of single servers looking like random data. Another application domain is given by Grid computing as we have shown in [EC04].
The second possibility mainly aims at the secure integration of cryptographic primitives into an agent Alliance. This represents a crucial step towards the practicability of fully autonomous mobile agents. This is why the following sections focus on a detailed complexity analysis of two important cryptographic primitives, namely the public-key algorithm RSA and the symmetric block cipher AES.

8.1 RSA

RSA [RSA79] is a public-key cryptosystem that can be used for data encryption as well as digital signatures. It can be considered the most popular and widespread system in use (for details see appendix A.1 and A.3). In order to have autonomous agents acting in their user's name, the possibility to carry private keys and thus to sign contracts is of great interest. At the same time, these keys can be read and possibly misused by each host, causing terrible consequences for the agent's originator one could imagine. SMAC is a solution to this problem. But what about the communication complexity of a digital signature performed via SMAC?

Modular Exponentiation with Square-and-Multiply An RSA signature requires the computation of $h^d \bmod n$ for a given hash value h of a message. Besides its efficiency,[1] the so-called *Square-and-Multiply Algorithm* (see Algorithm 9) is a natural solution to the exponentiation problem using SMAC. We will see this after a short review of the algorithm. Square-and-Multiply uses the unique representation

$$h^d \bmod n = h^{1 \cdot d_0} \cdot h^{2 \cdot d_1} \cdot h^{4 \cdot d_2} \cdot \ldots \cdot h^{2^k \cdot d_k} \bmod n$$

of h^d with $d = \sum_{i=0}^{k} d_i 2^i,\ d_i \in \{0,1\}$:

Input: $n \in \mathbb{N}, d, m \in \mathbb{Z}_n$
Output: $res = m^d \bmod n$

```
1 res := 1;
2 for i ← k to 0 do
3 |   res = res^2 mod n;
4 |   if d_i = 1 then
5 |   |   res = (res * m) mod n
```

Algorithm 7: Modular Exponentiation with Square-and-Multiply

As discussed in section 7 an implementation in traditional programming languages could open the possibility of side-channel attacks by analysing the execution of the program. Especially in case a private key is used as conditional argument for a control structure, important information could leak. Here, we have two problems:

1. By using `if` depending on the key bits, the whole key is revealed.
2. By the `for` loop the number of bits of the private key d is revealed.

As explained in section 7.3 we solve the first problem by computing both branches using the following construction: Be res_i the result after the ith loop. Then, res_{i+1} is computed by

$$res_{i+1} := res_i^2 (d_i \cdot h + (1 - d_i)).$$

[1] The number of multiplications needed to raise a number h to the power of d lies in $O(\log d)$.

The multiplication of d_i and h is a local scalar multiplication as the hash value is publicly known. The resulting product is a shared value. Thus, there are two shared multiplications, namely the computation of res_i^2 and multiplying the result with the result of the shared scalar multiplication. The resulting number of multiplication steps is in $\Theta(2k) = \Theta(2 \log_2 d)$. As the d_i are binary, they can be directly used as conditional statements for the computation of both branches. No further evaluation of arithmetic expressions is necessary. This is why the Square-and-Multiply algorithm is a natural choice for the exponentiation problem with SMAC.
Additional measures are necessary in order to hide the actual key size. For smaller d a padding is difficult as Square-and-Multiply would deliver wrong results. It is much easier to demand the private key to be of a specific binary length, as for example 1024 bit. Doing so, an attacker knows the exact length of the key and thus the highest valued bit to be 1. Consequently, the key space an attacker had to search through is reduced to 2^{1024-1} possible keys. This is one half of the original search space. For practice this is not relevant, but the problem could easily be circumvented by enlarging the key to 1025 bit. A last point to discuss in this context is the creation of the private key. Is there any problem to find an invertible number of exactly k bits in a given residue class ring $\mathbb{Z}_n$ with $n = pq$, with p, q prime numbers, and is there a serious reduction of the key space? The answer is "no", as the multiplicative group $\mathbb{Z}_n^\times$ has $(p-1)(q-1) = pq - p - q + 1$ elements. This means, the number of zero divisors is only $p + q - 1$. As all zero divisors are multiples of either p or q, the invertible elements are uniformly distributed over the whole ring. Consequently, there are as many k-bit invertible numbers in $\mathbb{Z}_n$ as smaller ones, and thus only one bit of the search space gets lost.

8.2 SMAC Digital Signature

In this section a complexity analysis for a digital RSA signature is given. We have shown in [Amm07] that an implementation of long numbers and performing a signature over an arithmetic circuit over $\mathbb{F}_p$ is impractical. For instance, an n-party RSA signature causes $21n^2$ GB data traffic! But, how could one avoid long numbers if one wants to apply public key cryptography? An intuitive but effective solution is to abandon the field property and to work in the residue class ring $\mathbb{Z}_n$ instead. This way, the modulo operations in the Square-and-Multiply algorithm are done automatically by the circuit gates and do not require communication nor processor time. Unfortunately, substituting a field with a ring is problematic as there are zero divisors in a ring that do not exist in a field. Therefore, one has to analyse the SMAC protocol as well as the application carefully to determine those parts in which divisions are necessary and whether zero divisors are a threat to the protocol.
There are two situations in which the inversion of numbers is required.

1. As shown in section 4.3.2 each multiplication requires a random triple. The triple generation phase in the beginning of the underlying SMPC protocol

from [HM01] uses Lagrange polynomials (see also section 4.2.1):

$$L_i(X) = \frac{(X - x_0) \cdots (X - x_{i-1}(X - x_{i=1}) \cdots (X - x_t)}{(x_i - x_0) \cdots (x_i - x_{i-1})(x_i - x_{i+1} \cdots (x_i - x_t)} \quad \text{fr } i = 1, ..n, \tag{8.1}$$

where x_i is the value that is uniquely assigned to agent A_i.

Those values $x_1, \ldots, x_n$ are assigned only once and can be arbitrarily chosen before the protocol starts. Thus, it is no problem to check whether the differences $(x_i - x_j)$ for $i \neq j$ are invertible in $\mathbb{Z}_n$. There is no limitation for the security parameters by doing so.

2. Secret reconstruction uses the method from Berlekamp and Welch [BW86]. This entails the necessity of solving a linear equation system using the algorithm of Gauss (for a detailed description see section 4.3.2). Here, one cannot eliminate the risk of having to invert a zero divisor a. In this case, the protocol would terminate with an error and by computing $\gcd(a, n)$ either p or q would be revealed. But, there are two arguments saying that this risk can be accepted:

   (a) Breaking RSA is equivalent to factorising the modulus n. Still, the problem of factorising long numbers is assumed to be hard enough to be used for cryptographic purposes. Consequently, it is very unlikely that a factorisation appears from nowhere just using SMAC.

   (b) Looking at probabilities, we get

   $$P(x \notin \mathbb{Z}_n^\times) = \frac{p + q - 1}{pq} = \frac{1}{q} + \frac{1}{p} - \frac{1}{pq} \approx \frac{1}{\sqrt{n}}.$$

   This is exactly the same probability as for breaking RSA by chance when guessing possible factorisations.

From the security point of view there is no reason not to use an arithmetic circuit over the ring $\mathbb{Z}_n$ for a digital signature. But what about the complexity of such a signature?

Communication Complexity Let d be a 1024 bit private key and n a 1024 bit modulus. The message m that is to be signed is initially a hash value that has 160 bit.[2] But its length changes during the Square-and-Multiply algorithm by squaring and modulo reductions. So, we assume the worst case which is 1024 bit. We know from chapter 4.3 that each distributed multiplication requires two messages exchanges between all agents.

1024 Square-and-Multiply steps × 2 distr. multiplications = 2048 message exchanges per agent

The data volume to be transfered by one agent is

2048 message exchanges × 1024 bit = 0.25 MB = 2.1 MBit

[2]This length depends on the used hash function.

Type	T1	ADSL	SDSL	Cable	Cable
Users	prof.	private	business	business	private
Downstream/s	1 GBit	16 MBit	2 MBit	20 MBit	6 MBit
Upstream/s	1 GBit	1 MBit	2 MBit	10 MBit	600 KBit
s/signature	0.0021	2.1	1.04	0.21	3.5

Table 8.1: Time required for one signature

With a realistic number of 10 agents in an Alliance (which gives us a redundancy of $t = 3$) the data volume that is to be communicated for one signature is 25 MB. Table 8.1 shows the time needed to perform one signature for different bandwidths currently in use. It is assumed that the communication of the single agents is parallel which means the total time is assessed by the time needed to transfer 2.1 MBit. As most private Internet connections are asynchronous, the upload rate (which is lower than the download rate) is used. One can easily see that all results are practical. To compute realistic time periods, the results from the tabular have to be multiplied by the network delay (`ping`) which usually takes some milliseconds. Nonetheless, the overall time required for a signature remains practical. Thus, the evaluation of this approach to a threshold signature is very positive. There are other theoretical approaches as for example [FMY98, DF92, CS00, Cac03, Fis03] which offer similar security properties, but those methods have been designed for signatures only. In contrary, SMAC can be used for every boolean function with finite fan-in gates. Concerning the time complexity of a signature, an SMAC signature can compete with those from above.

8.3 AES

The *Advanced Encryption Standard* (AES) [DR01] is a very efficient iterative block cipher used to encrypt communication channels. The algorithm is designed for different block and key sizes (128, 192 and 256 Bit). For the complexity analysis presented in this section, AES-256 is used. In this case, 14 iterations are performed. AES is defined over the field F_2^8 which means that a field element can be represented by one Byte. A data block is divided into 8 blocks of 4 field elements and stored in the following matrix:

$$A := \begin{pmatrix} a_{00} & a_{01} & .. & a_{07} \\ a_{10} & a_{11} & .. & a_{17} \\ a_{20} & a_{21} & .. & a_{27} \\ a_{30} & a_{31} & .. & a_{37} \end{pmatrix} \tag{8.2}$$

For each iteration, matrix A passes 4 transformational steps, *Bit Pattern Transformation, Row Transformation, Column Transformation* and *Key Addition.*

Bit Pattern Transformation: Each entry of matrix A is transformed by substituting each Byte by its multiplicative inverse element:

$$b_{ij} := a_{ij}^{-1}.$$

Then, a linear transformation over F_3 takes place:

$$\begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \tag{8.3}$$

The two transformation steps can be summarised into a so-called *S-Box* with 256 entries. Doing so, the overall Bit-transformation specified as follows:

$$T_B : F_{2^8} \rightarrow F_{2^8}.$$

Row Transformation: The rows of the matrix from equation 8.3 are rotated by use of circular rotates by a row-specific offset. Each byte of the second row is shifted one to the left. In the case of the 256 bit block, the first row is unchanged and the shifting for second, third and fourth row is 1 byte, 2 byte and 4 byte respectively.

Column Transformation: Each column is mapped by an invertible linear transformation:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \tag{8.4}$$

Key Expansion: The original key is expanded to a length of $256 \cdot (14 + 1)$ Bits and subsequently divided into blocks of 256 Bits. These blocks are the keys for the individual interactions. As there is an initial key addition, one more key is required. The expansion is given in algorithm 8.

Input: $Key[n_k]$ (column vectors of the key)
Output: $ExKey[n_b(n_r + 1)]$ (column vectors of the expanded key)

```
for i ← 0 to n_k − 1 do
    ExKey[i] ← Key[i];
for i ← n_k to n_b(n_r + 1) − 1 do
    temp ← ExKey[i − 1]; if i mod n_k = 0 then
        temp ← T_B(T_R(temp) + c(i/n_k);
    if i mod n_k = 4 then
        temp ← T_B(temp);
    ExKey[i] = ExKey[i − n_k] + temp;
```

Algorithm 8: Key Expansion

The value n_k represents the number of column vectors of a key block, n_b the number of column vectors of a data block, and n_r the number of iterations. In

case of AES-256 it is $n_k = n_b = 8$ and $n_r = 14$. The function $c(x)$ outputs a constant value. Finally, the transformation T_R is a permutation of the form $(a, b, c, d) \rightarrow (b, c, d, a)$ (rotate left).
AES could be implemented with an arithmetic circuit over the field $\mathbb{F}_{2^8}$ as well as over the field $\mathbb{F}_2$. Both possibilities are analysed now.

8.3.1 Complexity Analysis Using the Field $\mathbb{F}_{2^8}$

Over the field $\mathbb{F}_{2^8}$ the first transformation, namely the *Bit Transformation* cannot be computed over shares, as the single bits cannot be accessed. Furthermore, the computation of multiplicative inverse elements is impossible. As for example shown in chapter 7, page 113, an interpolation can be used to interpolate the whole transformation function. This requires 254[3] multiplications for each of the 32 Bytes. The *Row Transformation* does not require multiplications as it is only a rotation of field elements. In contrast, the *Column Transformation* costs 4 multiplications per Byte, which sums up to 16 multiplications. The permutation T_R and the simulation of the conditions and branches for $c(x)$ can be computed locally. The only operation requiring multiplications in the *Key Expansion*, is the Bit-transformation T_B. First, one has to determine the number of calls of T_B in the loop: The loop is executed 112 times. Each condition in the branch becomes exactly $112/8 = 4$ times true. This results in 28 Bit-transformations.
The overall number of multiplications of AES-256 is then

$$14(32 \cdot 254 + 4 \cdot 32) + 28 \cdot 254 = 122696.$$

Thereby, the data volume that has to be communicated is

$$245392 \cdot n^2 \text{ Byte}.$$

Considering an agent Alliance with 10 members, each member communicates $2.45MB$, which amounts to a total of 24.5 MB for the whole Alliance.

8.3.2 Complexity Analysis Using the Field $\mathbb{F}_2$

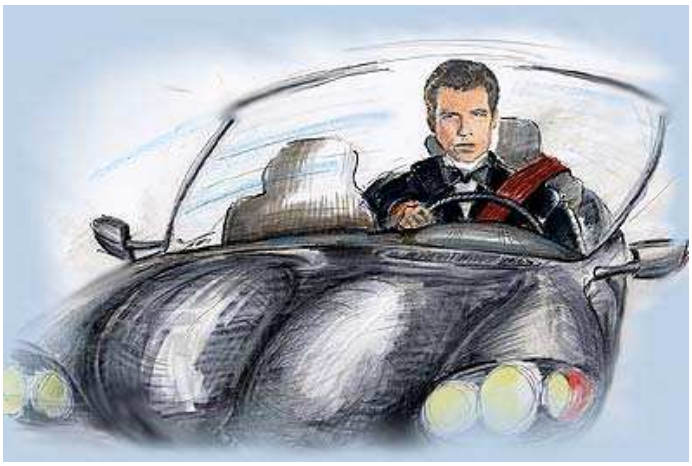
In an arithmetic circuit over $\mathbb{F}_2$ the inverse elements have to be interpolated, too. The costs for the other operations is higher than in $\mathbb{F}_{2^8}$. Although equation 8.3 could be computed locally, there is no advantage as the interpolation is still necessary. Consequently, AES implemented with an arithmetic circuit over $\mathbb{F}_2$ does not offer a better complexity.

[3] as there are 256 possible input values

Chapter 9

Conclusion

This thesis presents a security framework for groups of co-operating mobile agents, the so-called agent *Alliances* which spends its life in an open and untrusted environment. Security in this context means that the functionality of the agents can be guaranteed as long as no more than an upper limit $t(n)$ of the Alliance members get corrupted at the same time. No trusted authority is required to provide this guarantee. This is a very potent result, the more so as the Alliance model is the only security model know up to day offering this security in the context of *mobile* agents. A friend of mine illustrated this situation by the following picture of James Bond using a safety belt.



9.1 The Alliance Model

The Alliance framework defines a protocol for secure multi-agent computations which defines several phases in the life of an Alliance. The functionality of the Alliance is implemented in an arithmetic circuit over a finite field $\mathbb{F}$. Its evaluation is one of the phases, namely the computation phase. For the evaluation, a cryptographic protocol for secure multi-party computation (SMPC) is used.

This means, all computations are done on input that is shared via a verifiable secret sharing scheme. The computations themselves are distributed and fault-tolerant against up to $t(n)$ corrupted inputs. As the Alliance functionality can require the agents to migrate to other hosts, the evaluation of the circuit on one host is only partial. Consequently, a new instance of the protocol for SMPC must be invoked. This brings up a new security problem: Because of the migration, the attacker model assumed in the SMPC, an adaptive adversary, is not suitable anymore. Without migration, an adversary can corrupt at most $t(n)$ agents and cannot disturb the distributed computation. With migration, it is possible to store the knowledge from one protocol instance and combine it with knowledge from a later instance, obtained by another Alliance member. Thus the adversary could collect enough data, to compromise the Alliance, although there is no protocol instance in which she corrupted more than $t(n)$ agents. This adversarial power equals the one of a *mobile adversary*. The model of mobile adversaries is a young research field. A mobile adversary is the most powerful known and there is only one means available that can be used to protect against her: randomised re-sharing techniques. Re-sharing protocols are cryptographic multi-party protocols that re-share the computational state of the parties (here: the Alliance). In detail this means that old shares are invalidated and new shares generated and distributed among the parties.
Consequently, the lifetime of an Alliance consists of invoking the SMPC protocol after having arrived on a new host, and invoking the re-sharing protocol before migration takes place. As both protocol types offer distributed and fault-tolerant computations, the functionality in these phases of the Alliance lifetime can be regarded as secured. The actual migration cannot be done with an existing multi-party protocol. This is unfortunate as the migration of data from one set of hosts to another is sensitive to network attacks, like spoofing, replay Denial-of-Service, etc. This gap is filled by the protocol for secure multi-agent computation, which is presented in chapter 5. Cryptographic primitives like authentication of hosts and digital signatures prevent from fraud. Also majority votes are used to secure the sensitive migration phase.
The Alliance model offers cryptographic security for mobile agent computations against a mobile and computationally bounded adversary. Consequently, a challenging task is finally solved, the feasibility is shown is shown by a theoretical model.

9.2 On the Implementation of Alliances

As the Alliance model offers a theoretical framework, some concrete protocols have been proposed in order to fill the model with life. Two tasks had to be solved in this context: First, suitable protocols for secure multi-party computation and re-sharing had to be selected. Second, new sub-protocols necessary for the migration process had to be designed. The main objective was to find an efficient solution for all protocols to achieve a practical security model.
Most of the published SMPC protocols only aim at feasibility results, not at practicability. Consequently, the complexity of these protocols is outrageous. Finally, in the beginning of this decade, the attention of the researchers turned from feasibility to practicability. In 2001, Hirt and Maurer [HM01] published the first protocol that had a communication complexity of $O(mn^2)$, where n is the

number of parties and m the number of multiplication gates in the circuit. The security model of this protocol makes assumptions that can be met by a realistic network. Since then, alternative protocols have been published, either offering a better complexity, but making unrealistic assumptions, or offering a higher fault-tolerance at the cost of complexity. Therefore, the protocol from Hirt and Maurer is still the one best suited for the Alliance model. It is described in detail in section 4.3.2 on page 63.
For re-sharing the protocol of Ostrovsky and Young [OY91] has been selected, as it is proven to be secure in the UC framework of Canetti (see appendix B), in which also the protocol from [HM01] is secure. The protocol is described in section 5.2.5 on page 88.
The new protocols necessary for migration are `count` and `migrate`. Both are distributed protocols in the sense that the input of a majority of honest Alliance members is necessary to compute an output. The protocol `count` is responsible to determine, whether there is already a majority of agents available. If so, it invokes `migrate` which handles the data transfer involved in migration and invokes the re-sharing protocol.
The overall complexity of the resulting protocol for secure multi-agent computation is in $O(mn^2)$ (with m, n as above).

9.3 Network Attacks

The protocol for secure multi-agent computation is based on the assumption that at no point in time the adversary controls more than $t(n)$ members of the same Alliance. In the theoretical framework, network errors causing that an agent cannot follow the protocol, are considered as corruption by the adversary. In practical deployment, it makes sense to distinguish between the probability of network errors and the probability of corruption of agents through the adversary. Chapter 6 therefore analyses which kind of attacks could raise the probability that $t(n)$ is exceeded although the adversary does not host more than $t(n)$ agents. The analysis is given using an attack tree, which starts with an abstract attack in the root and subsequently makes the respective attacks more concrete in the outgoing edges. The attacks involved are discussed and solutions to these security problems are proposed.
Chapter 6 also gives some concrete examples for the probability of Alliance compromise assuming a random target selection in the migration process. It is shown, that the probability for a compromise increases with the number of migrations. An Alliance generally tolerates a network hostility of approximately 20%. Exceeding 25%, the risk of Alliance compromise increases significantly. This is similar to current results of investigations on the consequences of reduced network connectivity.

9.4 Feasibility Results

As mentioned above, the protocol for secure multi-agent computation has a complexity that is quadratic in the number of Alliance members. At the first glance this seems to be practical. But what about the constant factor that is invisible when only looking at the complexity class? The exact factor involved

in the implementation of individual functionalities, can only be determined by investigating the structure of the arithmetic circuit that implements this functionality. In order to do so, it must be analysed, how the respective function can be simulated by an arithmetic circuit without leaking information on the input shares. This problem is not trivial as for example conditional branches obviously leak information in case a data share is compared to a public value. This is why chapter 7 is dedicated to the secure simulation of programs.
It is shown that in general the communication complexity of a simulated program is too high for the bandwidth of today's networks. However, it is possible to limit the secure multi-agent computations to specific sub-functionalities that are of particular sensitivity, like for example a digital signature. Therefore, chapter 8 investigates two cryptographic primitives, namely an RSA signature and the symmetric block cipher AES. It turned out, that using the square-and-multiply algorithm, a digital signature using SMAC is efficient and can even compete with current protocols for threshold signatures. On the other hand, it is not advisable to use a distributed AES to encrypt communication channels. It is more likely to make use of the encryption primitive the host makes available to its agents.

9.5 Future Work

The current approach for an implementation of the Alliance framework contains synchronous sub-protocols for secure multi-party computation and re-sharing. This has two reasons. At the time of doing research on this topic there were no efficient asynchronous protocols available. Using synchronous protocols is unproblematic as there are synchronisation techniques such as from Awerbuch [Awe85] allowing to use a synchronous protocol in an asynchronous network. Unfortunately, these techniques require to introduce times to synchronise the communication and consequently slow down the Alliance computation. Meanwhile, the situation has changed a bit. There is for example the protocol of Hirt et al. [HNP05] offering robustness against up to $t < n/3$ corrupted parties and a communication complexity of $O(m\kappa n^3)$, where m is the number of multiplication gates and κ a security parameter.[1] As we have seen in this thesis, even quadratic complexity is too high for arbitrary functionalities. Therefore, asynchronous solutions are not yet suitable for practice. Moreover, the security model of these protocols only guarantees the security of the computations if the necessary messages are delivered to the parties. They do *not* solve the possible problem of delayed message delivery in an asynchronous network. Consequently, they offer no security advantage when compared with synchronous protocols. However, it is worth to take a closer look at an asynchronous solution for agent Alliances.
A second aspect of the framework that could be analysed further is the protocol compiler mentioned in chapter 3. The simulation of arbitrary programs requires to design a compiler that transforms for example a Java program into an arithmetic circuit. Thereby, two aspects have to be considered:

1. possible information leakage

[1] Asynchronous protocols are based on threshold cryptographic primitives which offer computational security. Usually the parameter equals the length of a cryptographic key.

2. size of the circuit (especially the number of multiplication gates)

This is a challenging task, worth another PhD thesis.

Appendix A

Cryptographic Requirements

A.1 RSA

RSA is called after its designers Rivest, Shamir and Adleman [RSA78, RSA79] and is the most popular public-key algorithm. Its security is based on the difficulty to factorise large numbers. Until now, it has not been proven or disproven that the latter is an NP-hard problem, but, nonetheless there is a lot of trust in this algorithm.

Algorithm

Public Key:
(n, e) with $n = p \cdot q$, with $p,\ q$ secret prime numbers, and $e \in \mathbb{Z}_n^{\times}$ [a].

Private Key:
$d \equiv_{(p-1)(q-1)} e^{-1}$

Encryption of clear-text block m:
$c \equiv_n m^e$

Decryption of cipher-text block c:
$m \equiv_n c^d$, since

$$c^d \equiv_n (m^e)^d \equiv_n m^{ed} \equiv_n m^{ed \bmod (p-1)(q-1)} \equiv_n m^1 \equiv_n m.$$

[a] $\mathbb{Z}_n^{\times}$ is the multiplicative group of the ring $\mathbb{Z}_n$. Nowadays, n is mostly chosen of size ≥ 2048 bit.

A.2 Cryptographic hash functions

Hash functions are a valuable tool for numerous cryptographic applications, such as integrity checks, digital signatures (see section A.3) and random number generation. Although it is quite simple to design a general hash function, this

is not as easy for *cryptographic* hash functions. Those must have the following properties.

Definition A.1. *(Cryptographic hash functions)*
A function h is called a cryptographic hash function*, if:*

1. *The function h maps texts of arbitrary length over an alphabet $\mathcal{A}$ to texts of fixed length over an alphabet $\mathcal{B}$. More formally, this means*

$$h : \mathcal{A}^+ \rightarrow \mathcal{B}^n.$$

   $\mathcal{A}$ and $\mathcal{B}$ are not necessarily different.

2. *The set of hash values $h(\mathcal{A}^+)$ must be large to keep the probability for a collision to occur small.*

3. *Computation of the function must be fast.*

4. *The function must react sensitively to local changes in the text. This implies changing a Bit in the source causes a significant difference in the resulting hash values.*

5. *For $l \rightarrow \infty$ the set of all texts of length l must have an equal distribution on the set of hash values.*

6. *It must be computationally infeasible to compute collisions for a given text.*

Examples for cryptographic hash functions are MD5 and its successor SHA [Sch96].

A.3 Digital Signatures

A digital signature is thought to replace a manual signature in the digitalised world (for example in e-business transactions). A hand-written signature provides the following properties (see also [Sch96]):

- *Authenticity of the signature*: The signature convinces the receiver that the signer deliberately signed the document.
- *Unforgeability*: The signature proves that the signer and no other party signed the document.
- *Inseparability*: The signature is not re-usable as it is part of the document and cannot be detached and used for another document.
- *Immutability*: A document cannot be changed after it has been signed.
- *Non-repudiation*: Document and signature are physically available. The signer cannot subsequently assert not having signed the document.

In reality, none of these properties is really valid. But there is a high probability that fraud will be detected and the potential damage is in general relatively small. In the field of e-business on the other hand, computers enable attackers to attack numerous people in parallel. Therefore, legislation and user ask for signature schemes which make it computationally infeasible to hurt the above

properties. Most common for such schemes is the use of a public key cryptographic primitive, for example the RSA public key system. The algorithm is similar to the RSA encryption scheme, but uses the keys in a different order.

Algorithm

Public key:
An RSA public key e.

Private key:
An RSA private key d.

One-way function:
$f : K \times M \to M$ applies a private key from key space K to a message $m \in M$.
Signing:
m_{sig}: $(m, c) := (m, f(d, m))$
Integrity/Authenticity Check:
$m \stackrel{?}{=} f(e, c)$

Problem: Since the messages are of arbitrary size and d, e are large numbers, f cannot be computed efficiently (this problem is typical for public-key algorithms).
Solution: Generate a small unique *fingerprint* of the message m. This can be done by means of a cryptographic hash function h. Then, the fingerprint instead of the whole message is signed.

Algorithm

Signing:
$(m, c) := (m, f(d, h(m)))$
Integrity/Authenticity Check:
$h(m) \stackrel{?}{=} f(e, f(d, h(m))$

Appendix B

The UC Framework of Canetti

It has been mentioned that the Hirt-Maurer protocol offers unconditional security in the UC framework of Canetti [Can00]. As a detailed discussion on this framework is beyond the scope of this thesis, only a summary originating from one of our papers [EM03] is given here.
Canetti's model provides security guarantees for arbitrary (even a priori unknown) concurrent environments with an asynchronous communication network that delivers messages publicly, unauthenticated and without guaranteed message delivery. This is exactly the environment mobile agents are supposed to live in.
Assume n servers jointly computing a functionality $\mathcal{F}$ which is realized by an n-party-protocol π. These servers are capable to participate concurrently in several protocol runs. Each of those executed programs is denoted as party. Furthermore, there is an adversary $\mathcal{A}$ in Canetti's model that is able to corrupt a limited number t of servers. In this case, it can read the entire state (including its history) and control the behaviour of these parties. Additionally, $\mathcal{A}$ has the power to read, modify, delay, and even delete outgoing messages of all n parties. Each entity is modelled as a Turing machine with two pairs of communication tapes. One for incoming/outgoing messages of the parties, the other one for local protocol input/output. Another adversarial entity $\mathcal{Z}$, which is called the *environment*, represents everything outside the current protocol execution. $\mathcal{Z}$ is responsible of delivering inputs to the parties since their origin is considered as external. Notice that both adversarial entities are distinguishable by their knowledge and control. $\mathcal{A}$ knows and controls everything concerning messages between the parties, but is unaware of the inputs/outputs of the protocol, and for $\mathcal{Z}$ it is vice versa. Both are allowed to communicate with each other freely. This model is called *real-life-model*.
For the definition of a secure protocol, one has to suppose an ideal setting. Obviously, no protocol execution can achieve more reliability than a protocol using a trusted entity which gets the inputs from all parties and returns (correct) outputs. A real-life-model supplemented with an unbounded number of such trusted entities for computing any functionality $\mathcal{F}$, is called $\mathcal{F}$-hybrid-model. A protocol π in the real-life-model is called secure, if

1. for any adversary attacking π there is one adversary in the $\mathcal{F}$-hybrid-model and

2. no possible environment is able to decide whether it acts in a protocol execution within the $\mathcal{F}$-hybrid or the real-life model.

Therefore, the most interesting cases are those in which the *interactive distinguisher* $\mathcal{Z}$ holds back some knowledge from $\mathcal{A}$. This enables $\mathcal{Z}$ to check whether the protocol outputs are correlated to this secret knowledge and, thus, it might be able to differentiate the models.
Since our agent communities are supposed to work in the Internet, we cannot presume the existence of a broadcast channel. On account of this, we need Byzantine Agreements (see [BGW88]) which limit the maximum number t of corrupted parties to $n/3$ to obtain a protocol that is secure in the sense of Canetti's definition. For our agent setting, this implies that more than $2n/3$ of the hosts must be honest during each time interval in which no migration takes place.

Distributed Computations Since the goal of this thesis is to secure the execution of arbitrary functions, those have to be translated into a t-robust protocol. This has to be done because an adversary in the Alliance model is limited to influence less than t inputs. Several protocol compilers have been developed. See for example [GMW87], [BGW88] and [CCD88]. In [GMW87] the resulting protocol is divided in two steps. At first, each party commits to its local input. To be able to detect a party that deviates from the protocol the other parties possess shares of everyone's randomness. The second part, the execution, is organised in several rounds. In each of them, every party is activated at least once to perform computations and to send messages. The correctness (in the sense of the protocol) of one party's activities are checked by the others through a zero-knowledge proof. Messages of one round must have been delivered until the beginning of the next round.
Canetti states in [Can00] that he does not know if [GMW87] is secure in his model. He proposes the use of [BGW88] which provides an information-theoretic secure synchronous protocol that stays secure in his setting. In [BGW88], the authors use a verifiable secret sharing scheme (VSS) to enable the community to detect improper or missing commitments in the first step. The actual evaluation of the function is done in the second phase. As the protocol of Hirt and Maurer uses a slightly modified version of [BGW88], it can be assumed, that [HM01] is also secure in this framework. Also asynchronous networks can be handled by using the techniques of [BOCW93] and [BOKR94].
The advantages of distributed computations are the guaranteed confidentiality of data and the correct execution of an user-defined functionality as long as less than $n/3$ of the agents are corrupted or spied out. This is a direct result given by [Can00]. For a hostile environment like the one autonomous mobile agents are living in, this is already a quite strong guarantee.

Appendix C

VSS of Ben-Or, Goldwasser and Wigderson

The model for secure multi-agent computation which is presented in this thesis is based on the protocol for secure multi-party computation from [HM01]. The underlying VSS scheme of this protocol is the (slightly modified) one of Ben-Or, Goldreich and Wigderson from [BGW88]. It is introduced now. Although [BGW88] presents two protocols, the presentation is limited to the one providing unconditional security against up to t cheaters.
In contrast to some other VSS protocols the one from [BGW88] allows computations on shares. As such it can be used to store data in a secure way, but it also enables a set of parties to perform secure multi-party computations. As mentioned before, the security of these computations is unconditional and perfect. The approach is based on Shamir's secret sharing from [Sha79] using a generalised Reed-Miller code for error correction.

C.1 Assumptions

Let n be the number of parties participating in the computation and $n = 3t = 1$. Another choice is also possible but does not improve the level of robustness while, on the other hand, it causes a higher complexity. Thus, the protocol has the following two properties:

1. It is possible to detect a faulty dealer as long a not more than t parties misbehave.
2. When co-operating, an arbitrary set of up to t parties does not gain any information about s.
3. It is easy to reconstruct s from its shares if not more than t shares are wrong or missing at the end of the computations

C.2 Secret Sharing

For the rest of section C $\mathbb{F}$ denotes a finite field, $s \in \mathbb{F}$ be the secret that ought to be shared and $\omega \in \mathbb{F}$ be a n-th root of unity (i.e. $\omega^n = 1$ and $\omega^i \neq 1$ for

$1 \leq i < n$).
Sharing a secret goes along the lines of [Sha79] with a slight restriction for the data points: For $i = 0, \ldots, n-1$ party P_i gets the share $s_i = (\omega^i, f(\omega^i)$.

C.3 Verification of Shares

As discussed before, it is possible that the dealer who is generating and distributing the input shares to the parties is faulty. Ben-Or et al. provide two alternative methods which allow the parties to verify the input they have receive. Namely, an error-free one and one that has a small probability of error. The presentation in this section is limited to the first one:

1. The dealer of the secret s selects a random polynomial $f(X, Y)$ with $f(0, 0) = s$. Then, he sends to each party P_i the polynomials $f_i(X) = f(X, \omega^i)$ and $g_i(Y) = f(\omega^i, Y)$. The i-th share is defined as $s_i = f_i(0)$ while $f_i(X)$ and $g_i(Y)$ are used for verification purposes.

2. The parties P_i send to each party P_j the value

$$s_{i,j} = f_i(\omega^j) = f(\omega^j, \omega^i) = g_j(\omega^i).$$

3. Each party P_j checks whether for all $0 \leq i \leq n-1$ the points $(\omega^i, s_{i,j})$ lie on their polynomial $g_j(Y)$.

4. If any party P_j detects an i with incorrect $s_{i,j}$ it broadcasts the coordinates a request to make (i, j) public.

5. If more than t points were incorrect, the player could be faulty (the assumption requires the number of malicious parties to be t at maximum). But it is also possible that P_j is lying and wrongly accuses the dealer. P_j broadcasts the request to make both $f_j(X)$ and $g_j(Y)$ public which is then done by the dealer together with a publication of all wrong points $s_{i,j}$. This makes all $s_{k,j}$ for $0 \leq k \leq n-1$ public. All players check the public values with their polynomials and as soon as one player finds any inconsistencies he broadcasts a complaint by requesting his private information to be published.

6. If more than t players have asked to publish their private information or the dealer did not satisfy all requests, he must be faulty and the protocol is stopped.

7. If less than $t+1$ players have found inconsistencies there are at least $t+1$ honest parties which can uniquely reconstruct the secret polynomial $f(X, Y)$. In this case the other parties take the public information as their share.

In case of a non-faulty dealer no information on the secrets of all honest parties are revealed.

C.4 Error Correction and Reconstruction

By using an n-th root of unity, the shares define a discrete Fourier Transform of the sequence $(a_0, a_1, \ldots, a_t)$ which defines the secret polynomial

$$f(X) = a_0 + a_1 X + \ldots + a_t X^t.$$

Let

$$\hat{f}(X) = s_0 + s_1 X + \ldots + s_{n-1} X^{n-1}$$

be the Fourier Transform of $(a_0, a_1, \ldots, a_t)$. Then, the inverse transform is defined as

$$a_i = \frac{1}{n} \hat{f}(\omega^{-i})$$

and $\hat{f}(\omega^{-i}) = 0$ for $i = t+1, \ldots, n-1$. The coefficients of $\hat{f}$ satisfy the linear equations

$$\sum_{i=0}^{n-1} \omega^{r \cdot i} \cdot s_i = 0 \text{ for } r = 1, \ldots, 2t.$$

The polynomial

$$g(X) = \sum_{i=t+1}^{n-1} (x - \omega^{-i})$$

divides $\hat{f}$ and can be considered as the generating polynomial of a Generalised Reed-Miller code [PW72] of length n. The sequence of shares $(s_0, s_1, \ldots, s_{n-1})$ is a code word and thus allows a correction of up to $\frac{1}{2}\deg(g(X))$ errors. By definition of $n = 3t + 1$ it follows that t errors can be corrected. Consequently, the protocol allows to handle a maximum of t corrupted or missing inputs shares when reconstructing the secret s. Reconstruction itself is a polynomial interpolation as presented in section 4.2.1.

List of Figures

List of Tables

List of Algorithms

List of Protocols

Bibliography

[ACCK01] Joy Algesheimer, Christian Cachin, Jan Camenisch, and Günter Karjoth. Cryptographic security for mobile code. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 2–11, 2001.

[AF90] Martin Abadi and Joan Feigenbaum. Secure circuit evaluation. *Journal of Cryptology*, 2(2):1–12, 1990.

[AFK89] Martin Abadi, Joan Feigenbaum, and Joe Killian. On hiding information from an oracle. *Journal on Computer and System Sciences*, 39(1):21–50, 1989.

[AKG98] Nadarajah Asokan, Günter Karjoth, and Ceki Gulcu. Protecting the computation results of free-roaming agents. In K. Rothermel and F. Hohl, editors, *Proceedings of Mobile Agents '98*, number 1477 in Lecture Notes in Computer Science, pages 183–194. Springer Verlag, 1998.

[AM88] Carlisle M. Adams and Henk Meijer. Security-related comments regarding McEliece's public-key cryptosystem. In C. Pomerance, editor, *Advances in Cryptology — CRYPTO '87*, number 293 in Lecture Notes in Computer Science, pages 224–228. Springer Verlag, 1988.

[Amm07] Christoph Amma. Über die Implementierung kryptographischer Primitive mittels sicherer Multiagentenberechnungen. Master's thesis, Universität Karlsruhe (TH), 2007.

[Awe85] Baruch Awerbuch. Complexity of network synchronization. *Journal of the ACM*, 32(4):2–11, 1985.

[Baz98] Rida A. Bazzi. Secure mobile agents. Technical report, Arizona State University, Department of Computer Science and Engineering, 1998.

[BBB89] J. Bar-Ban and Donald Beaver. Non-cryptographic fault-tolerant computing to a constant number of rounds of interaction. In *Proceedings of 8th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 201–210. ACM Press, 1989.

[BC87] Gilles Brassard and Claude Crépeau. Zero-knowledge simulation of boolean circuits. In A.M. Odlyzko, editor, *Advances in Cryptology — CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 223–233, 1987.

[Bea91a] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances of Cryptology — CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer Verlag, 1991.

[Bea91b] Donald Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):73–122, 1991.

[BEPW02] Peter Biddle, Paul England, Marcus Peinado, and Bryan Willman. The darknet and the future of content distribution. In *Proceedings of the 2002 ACM Workshop on Digital Rights Management*, 2002.

[BFKR90] Donald Beaver, Joan Feigenbaum, Joe Kilian, and Peter Rogaway. Security with low communication overhead. In *Advances in Cryptology — CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 62–76. Springer Verlag, 1990.

[BGI+01] Boaz Barak, Oded Goldreich, Russel Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Proceedings of CRYPTO '01*, pages 1–18, 2001.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the 20th ACM Symposium on the Theory of Computing — STOC '88*, pages 1–10, 1988.

[Bla79] G. Blakely. Safeguarding cryptographic keys. In *Proceedings of the AFIPS National Computer Conference*, pages 313–317, 1979.

[BM03] Amos Beimel and Lior Malka. Efficient reliable communication over partially authenticated networks. In *Proceedings of the 22nd Symposium on Principles of Distributed Computing — PODC '03*, pages 233–242. ACM Press, 2003.

[BMW98] Ingrid Biehl, Bernd Meyer, and Susanne Wetzel. Ensuring the integrity of agent-based computations by short proofs. In K. Rothermel and Fritz Hohl, editors, *Proceedings of the Second International Workshop on Mobile Agents — MA '98*, Lecture Notes in Computer Science, pages 183–194. Springer, 1998.

[BOCW93] Michael Ben-Or, Ran Canetti, and Avi Wigderson. Asynchronous secure computation. In *Proceedings of 25th ACM Symposium on the Theory of Computing*, pages 52–61. ACM Press, 1993.

[BOKR94] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In *Proceedings of 13th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 183–192. ACM Press, 1994.

[BR02] Walter Binder and Volker Roth. Secure mobile agent systems using Java: Where are we heading? In *Proceedings of the 2002 ACM Symposium on Applied Computing*, pages 115–119. ACM Press, 2002.

[BW86] Elwyn R. Berlekamp and Lloyd Welch. Error correction of algebraic block codes. US Patent Number 4,633,470, 1986.

[Cac03] Christian Cachin. An asynchronous protocol for distributed computation of rsa inverses and its applications, 2003.

[Can00] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, 2000. Report 2000/067.

[CCD88] David Chaum, Claude Crepeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *Proceedings of 20th ACM symposium on theory of computing — STOC '88*, pages 11–19, 1988.

[CCKM00] C. Cachin, J. Camenisch, Joe Kilian, and J. Müller. One-round secure computation and secure autonomous mobile agents. In U. Montanari, J.P. Rolim, and E. Welzl, editors, *Proceedings of 27th International Symposium on Automata, Languages and Programming — ICALP '00*, volume 1853 of *Lecture Notes in Computer Science*, pages 512–523. Springer, 2000.

[CDN01] Ronald Cramer, Ivan Damgaard, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In *International Conference on the Theory and Application of Cryptographic Techniques — EUROCRYPT '01*, volume 2045 of *Lecture Notes in Computer Science*, pages 280–299. Springer-Verlag, 2001.

[CGH+95] David Chess, Boris Grosof, Colin Harrison, David Levine, Colin Parris, and Gene Tsudik. Itinerant agents for mobile computing. *IEEE Personal Communications*, pages 34–49, 1995.

[CGMA85] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proceedings of 26th Annual IEEE Symposium on Foundations of Computer Science — FOCS '85*, pages 383–395, 1985.

[CHK97] David Chess, Colin Harrison, and Aaron Kershenbaum. Mobile agents: Are they a good idea? In *Mobile Object Systems: Towards the Programmable Internet*, volume 1222 of *Lecture Notes in Computer Science*, pages 46–48. Springer Verlag, 1997.

[CK88] Claude Crepeau and Joe Kilian. Achieving oblivious transfer using weakened security assumptions. In *Proceedings of 29th Annual IEEE Symposium on Foundations of Computer Science — FOCS '88*, pages 42–52, 1988.

[CK97] Teck-How Chia and Srikanth Kannapan. Strategically Mobile Agents. In K. Rothermel and R. Popescu-Zeletin, editors, *Proceedings of the First International Conference on Mobile Agents – MA '97*, volume 1219, pages 149–161. Springer-Verlag, 1997.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2. ed. edition, 2001.

[Cor] IBM Corp. Messaging and queuing technical reference. SC33-0850.

[CPV97] Antonio Carzaniga, Gian Pietro Picco, and Giovanni Vigna. Designing distributed applications with a mobile code paradigm. In *Proceedings of the 19th International Conference on Software Engineering*, 1997.

[CR93] Ran Canetti and Tal Rabin. Fast asynchronous Byzantine agreement with optimal resilience. In *Proceedings of 25th Annual ACM Symposium on Theory of Computing — STOC '93*, pages 42–51. ACM, 1993.

[CS00] R. Cramer and V. Shoup. Signature schemes based on the strong rsa problem. *ACM Transactions on Information and System Security*, 3(3):161–185, 2000.

[Dat03] R. S. Datta. *Algebraic Methods in Game Theory*. PhD thesis, UCLA at Berkley, 2003.

[DDWY93] Danny Dolev, Cynthia Dwork, Orli Waarts, and Moti Yung. Perfect secure message transmission. *Journal of the ACM*, 40(1), 1993.

[DF92] Yvo Desmedt and Yair Frankel. Shared generation of authenticators and signatures. In *Proceedings of CRYPTO '91*, pages 457–469. Springer Verlag, 1992.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[DoD85] DoD. Trusted computer system evaluation criteria (tc-sec). Technical Report DoD 5200.28-STD, Department of Defense, December 1985.

[DR01] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer Verlag, 2001.

[Dre05] Ulf Dreyer. Agenten und Server – Sicher unter JADE(-S)? Master's thesis, Universität Karlsruhe (TH), IAKS, 2005.

[DS82] Dole and Strong. Polynomial algorithms for multiple processor agreement. In *Proceedings of ACM symposium on theory of computing — STOC '82*, pages 401–407. ACM, 1982.

[DW02] Yvo Desmedt and Yongge Wang. Perfectly secure message transmission revisited. In *Proceedings of EUROCRYPT '02*, volume 2332 of *Lecture Notes in Computer Science*, pages 502–517. Springer Verlag, 2002.

[EC04] Regine Endsuleit and Jacques Calmet. Introducing robust and private computation into grid technology. In *Proceedings of Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises — WET ICE '04*, pages 303–308. IEEE, 2004.

[EC05] Regine Endsuleit and Jacques Calmet. A security analysis on JADE(-S) V. 3.2. In Helger Lipmaa and Dieter Gollmann, editors, *Proceedings of NordSec*, pages 20–28, 2005.

[EM03] Regine Endsuleit and Thilo Mie. Secure multi-agent computations. In *Proceedings of the International Conference on Security and Management*, volume 1, pages 149–155. CSREA, 2003.

[EM06] Regine Endsuleit and Thilo Mie. Anonymous and censorship-resistant p2p filesharing. In *Proceedingsof Conf. on Availability, Reliability and Security*, pages 58–64. IEEE, 2006.

[Fei02] Patrick Feisthammel. Explanation of the web of trust of pgp. `http://www.rubin.ch/pgp/weboftrust.en.html`, 2002.

[FG96] Stan Franklin and Art Grasser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of ATAL*, pages 21–35, 1996.

[FGS96] William Farmer, Joshua Guttmann, and Vipin Swarup. Security for mobile agents: Authentication and state appraisal. In *Proceedings of the European Symposium on Research in Computer Security — ESORICS '96*, volume 1146, pages 118–130. Springer, 1996.

[Fis03] Marc Fischlin. The cramer-shoup strong-rsa signature scheme revisited. In *Public Key Cryptography — PKC '03*, volume 2567, pages 116–129. Springer, 2003.

[FKN94] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation. In *Proceedings of 26th ACM Symposium on the Theory of Computing — STOC '94*, pages 554–563. ACM Press, 1994.

[FM00] Michael Fitzi and Ueli Maurer. From partial consistency to global broadcast. In *Proceedings of 32th ACM Symposium on the Theory of Computing — STOC '00*, pages 494–503. ACM Press, 2000.

[FMY98] Yair Frankel, Philip D. MacKenzie, and Moti Yung. Robust efficient distributed RSA-key generation. In *The Thirtieth Annual ACM Symposium on Theory of Computing – STOC '98*, pages 663–672. ACM Press, New York, 1998.

[Fre07] Simon Frettlöh. Eine effiziente lösung für eine byzantinische vereinbarung in asynchronen netzwerken. Master's thesis, Universität Karlsruhe (TH), 2007.

[FY92] Yair Frankel and Moti Yung. Communication complexity of secure computation. In *Proceedings of 24th ACM Symposium on the Theory of Computing — STOC '92*, pages 699–710. ACM Press, 1992.

[Gál95] Anna Gál. Semi-unbounded fan-in circuits: Boolean vs. arithmetic. In *Proceedings of the 10th Annual Symposium on Structure in Complexity Theory*, pages 82–87, 1995.

[GED03] Li Gong, Gary Ellison, and Mary Dageforde. *Inside Java 2 Platform Security – Architecture, API Design and Implementation*. The Java Series. Addison Wesley Longman, 2nd edition edition, 2003.

[Gen96] *General Magic: The Telescript Reference Manual*, 1996. Available electronically at: `http://www.genmagic.com/Telescript/Documentation/TRM`.

[Gib91] Keith Gibson. Equivalent Goppa codes and trapdoors to McEliece's public key cryptosystem. In D.W. Davies, editor, *Advances in Cryptology — CRYPTO '91*, number 547 in Lecture Notes in Computer Science, pages 517–521. Springer Verlag, 1991.

[Gib95] Keith Gibson. *Algebraic Coded Cryptosystems*. PhD thesis, University of London - Royal Holloway and Bedford New College, 1995.

[GIKR01] Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. The round complexity of verifiable secret sharing and secure multicast. In *ACM Symposium on Theory of Computing*, pages 580–589, 2001.

[GJS96] James Gosling, Bill Joy, and Guy L. Steele. *The Java Language Specification*. Addison-Wesley, Reading, MA, USA, 1st edition, 1996.

[GK05] Shafi Goldwasser and Yael Tauman Kalai. On the impossibility of obfuscation with auxiliary input. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science – FOCS '05*, pages 553–562. IEEE Computer Society, 2005.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.

[GMR86] G. Goldreich, Silvio Micali, and Ron L. Rivest. How to construct random functions. *Journal of the ACM*, 33(4):792–804, 1986.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *Proceedings of 27th Annual IEEE Symposium on Foundations of Computer Science — FOCS '86*, pages 174–187, 1986.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play a mental game. In *Proceedings of 19th ACM symposium on theory of computing — STOC '87*, pages 218–229, 1987.

[Gnu] GNU MP - Library for arithmetic on arbitrary precision numbers. `http://www.gnu.org/directory/libs/gnump.html`.

[Gol97] Shafi Goldwasser. Multi party computations: past and present. In *Proceedings of the sixteenth annual ACM symposium on Principles of Distributed Computing — PODC '97:*, pages 1–6. ACM Press, 1997.

[Gol04] Oded Goldreich. *Foundations of Cryptography – Basic Applications*, volume 2, chapter 7. Cambridge University Press, 2004.

[GRR98] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified vss and fasttrack multiparty computations with applications to threshold cryptography. In *17th ACM Symposium on Principles of Distributed Computing — PODC '98*, pages 101–111, 1998.

[GW84] Gerhard Goos and William Waite. *Compiler Construction*. Springer, 1984.

[HH91] Vassos Hadzilacos and Joseph Y. Halpern. Message-optimal protocols for Byzantine agreement (extended abstract). In *Proceedingsof 10th annual symposium on principles of distributed computing*, pages 309–323. ACM, 1991.

[HM01] Martin Hirt and Ueli Maurer. Robustness for free in unconditional multi-party computation. In *Proceedings of Advances in Cryptology - CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 101–118. Springer Verlag, 2001.

[HMP00] Martin Hirt, Ueli Maurer, and B. Przydatek. Efficient secure multiparty computation. In T. Okamoto, editor, *Advances in Cryptology — ASIACRYPT '00*, volume 1976 of *Lecture Notes in Computer Science*, pages 143–161. Springer Verlag, 2000.

[HNP05] Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Cryptographic asynchronous multi-party computation with optimal resilience. In Ronald Cramer, editor, *Advances in Cryptology — EUROCRYPT '05*, volume 3494 of *Lecture Notes in Computer Science*, pages 322–340. Springer Verlag, 2005.

[Hoh98] Fritz Hohl. Time limited blackbox security: Protecting mobile agents from malicious hosts. *Lecture Notes in Computer Science*, 1419:92–113, 1998.

[Inc] Waterloo Maple Inc. Maple V, Release 4.

[JAD06] Jade website. Available electronically under `www.jade.tilab.com`, 2006.

[Jav06] Java. Available electronically: `http://java.com/en`, 2006.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *Proceedings of 20th ACM symposium on theory of computing — STOC '88*, pages 20–31, 1988.

[Knu98] Donald E. Knuth. *The art of computer programming*, volume 2. Addison-Wesley, 3. ed. edition, 1998.

[LM99] Sergio Loureiro and Refik Molva. Function hiding based on error correcting codes. In *Proceedings of the International Workshop on Cryptographic Techniques and Electronic Commerce — Cryptec '99*, pages 92–98, 1999.

[LMP01] Sergio Loureiro, Refik Molva, and Alain Pannetrat. Secure data collection with updates. *Electronic Commerce Research*, 1(1/2):119–130, 2001.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[LY99] Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, Reading, MA, USA, 2nd edition, 1999.

[Mas06] David Masso. Mobile IPv6 support for dual stack mobile nodes and routers. Master's thesis, Universtät Karlsruhe (TH), 2006.

[Mie03] Thilo Mie. Sicherheitsallianzen mobiler Agenten. Master's thesis, Universität Karlsruhe (TH), IAKS, 2003.

[MvRSS96] Yaron Minsky, Robbert van Renesse, Fred Schneider, and Scott Stoller. Cryptographic support for fault-tolerant distributed computing. In *Proceedings of the 7th ACM SIGOPS European Workshop*, pages 109–114, 1996.

[OY91] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks. In *Proceedings of the 10th Ann. ACM Symposium on Principles of Distributed Computing*, pages 51–59, 1991.

[Pal94] Elaine Palmer. An introduction to citadel - a secure crypto coprocessor for workstations. In *Proceedings of the IFIP SEC '94*, 1994.

[PBG89] Kenneth J. Perry, Piotr Berman, and Juan A. Garay. Towards optimal distributed consensus. In *Foundations of Computer Science*, pages 410–415. Dept. of. Comput. Sci., Chigago Univ., USA, 1989.

[Pos81] Jon Postel. Rfc790. `http://www.ietf.org/rfc/rfc0790.txt`, September 1981.

[PW72] Wiliam W. Peterson and E J. Weldon. *Error Correcting Codes*. MIT Press, 1972. p. 283.

[Rab81] Michael Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Aiken Computation Laboratory, Harvard University, 1981.

[RBO89] Tal Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of 21st ACM symposium on theory of computing — STOC '89*, pages 73–85, 1989.

[RHR97] Kurt Rothermel, Fritz Hohl, and Nikolaos Radouniklis. Mobile agent systems: What is missing? In *Proceedings of International Working Conference on Distributed Applications and Interoperable Systems — DAIS '97*, pages 111–124, 1997.

[RJ96] Lars Rasmusson and Sverker Jansson. Simulated social control for secure internet commerce. In *New Security Paradigms*, pages 18–26. ACM Press, 1996.

[Rot99] Volker Roth. Mutual protection of co-operating agents. In J. Vitek and Ch. Jensen, editors, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, volume 1603 of *Lecture Notes in Computer Science*, pages 275–285. Springer Verlag, 1999.

[RP02] Marc Rennhard and Bernhard Plattner. Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection. In *Proceedings of the Workshop on Privacy in the Electronic Society (in association with 9th ACM Conference on Computer and Communications Security)*, pages 91–102, 2002.

[RSA78] Ron Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 1(2):120–126, 1978.

[RSA79] Ron Rivest, Adi Shamir, and Leonard Adleman. On digital signatures and public-key cryptosystems. Technical Report MIR/LCS/TR-212, MIT Laboratory for Computer Science, January 1979.

[Sch96] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, second edition, 1996. Chapter 18.

[Sch99] Bruce Schneier. Attack trees. Dr. Dobb's Journal, December 1999.

[Sch00] Bruce Schneier. *Digital security in a networked world*. John Wiley & Sons, Inc., 2000.

[SGE02] Rainer Steinwandt, Willi Geiselmann, and Regine Endsuleit. Attacking a polynomial-based cryptosystem: Polly cracker. *International Journal of Information Security*, 1(3):143–148, 2002.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[SNS88] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of the USENIX Winter 1988 Technical Conference*, pages 191–202. USENIX Association, 1988.

[SS97] Markus Straßer and Markus Schwehm. A performance model for mobile agent systems. In *Proceedingsof Int. Conference on Parallel and Distributed Processing Techniques and Applications — PDPTA '97*, volume 2, pages 1132–1140. CSREA, 1997.

[ST98a] Thomas Sander and Christian F. Tschudin. Towards mobile cryptography. In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, pages 215–224,, 1998.

[ST98b] Tomas Sander and Christian F. Tschudin. On software protection via function hiding. In *Proceedings of the Second International Workshop on Information Hiding*, pages 111–123. Springer-Verlag, 1998.

[ST98c] Tomas Sander and Christian F. Tschudin. Protecting mobile agents against malicious hosts. In G. Vigna, editor, *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 44–60. Springer Verlag, 1998.

[Uni93] International Telecommunication Union. ITU-T Recommendation X.509: The Directory Authentication Framework, 1993.

[Vig98] Giovanni Vigna. Cryptographic traces for mobile agents. In *Proceedings of Mobile Agents and Security*, Lecture Notes in Computer Science, pages 137–153. Springer, 1998.

[Wee05] Hoeteck Wee. On obfuscating point functions. ePrint 2005/001, 2005.

[Wei00] Gerhard Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 2000.

[Wel05] Michael Welschenbach. *Cryptography in C and C++*. Apress, 2nd edition, 2005.

[WL92] Thomas Y.C. Woo and Simon S. Lam. Authentication for distributed systems. *IEEE Computer*, 25(1), January 1992.

[WSB99] Uwe G. Wilhelm, Sebastian Staamann, and Levente Buttyán. Introducing trusted third parties to the mobile agent paradigm. In J. Vitek and C.D. Jensen, editors, *Secure Internet Programming*, volume 1603 of *Lecture Notes in Computer Science*, pages 469–489. Springer Verlag, 1999.

[Yao82] Andrew C.-C. Yao. Protocols for secure computations. In *Proceedings of 23rd IEEE Symposium on the Foundation of Computer Science — FOCS'82*, pages 160–164. IEEE, 1982.

[Yao86] Andrew C.-C. Yao. How to generate and exchange secrets. In *Proceedings of 27th Annual IEEE Symposium on Foundations of Computer Science — FOCS '86*, pages 162–167, 1986.

[Yee94] Bennet S. Yee. *Using Secure Coprocessors*. PhD thesis, Carnegie Mellon University, 1994.

[Yee99] Bennet S. Yee. A sanctuary for mobile agents. In *Secure Internet Programming*, pages 261–273, 1999.

Index

Curriculum Vitae

24.06.1970	Born in Freiburg i.Br. Parents: Leonore Endsuleit, nee Kruschke, and Gustav Endsuleit
1976 till 1980	Primary school in Freiburg
1980 till 1989	General qualification for university entrance at “ Staudinger Gesamtschule” in Freiburg
1989 till 1990	Evocational education as business assistant (“Staatl. gepr. Wirtschaftsassistentin”) in Freiburg
1990 till 1992	Evocational education as bank employee (“Bankkauffrau”) in Gundelfingen
1992 till 2001	Studies in computer science at the “Universität Fridericiana zu Karlsruhe” Diploma: February 2001
since March 2001	Ph.D. student at the “Universität Fridericiana zu Karlsruhe”, Institut für Algorithmen und Kognitive Systeme
2003, 04 till 08	Maternity leave; April 16: birth of son Marc
2005, 04 till 05	Maternity leave; April 05: birth of daughter Emily
2006, 03	CISSP (certified information system security professional)

Publications

- **Journals**

  - Jacques Calmet, Pierre Maret and REGINE ENDSULEIT. "Agent-Oriented Abstraction". *Revista de la Real Academia de Ciencias*, special volume on symbolic computation in logic and artificial intelligence, vol. 98(1):77–83, 2004.
  - Rainer Steinwandt, Willi Geiselmann and Regine Endsuleit. "Attacking a Polynomial-based Cryptosystem: Polly Cracker". *International Journal of Information Security*, 1(3):143–148, 2002.

- **Refereed Conferences/Workshops**

  - Regine Endsuleit and Christoph Amma. "Agent Alliances: A Means for Practical Threshold Signature". To appear in *Proceedings of IEEE conference on Availability, Reliability and Security*, Vienna, Austria, April 2007.
  - Regine Endsuleit and Thilo Mie. "Anonymous and Censorship-resistant P2P Filesharing". In: *Proceedings of IEEE conference on Availability, Reliability and Security*, pp. 58–64, April 2006.
  - Regine Endsuleit and Jacques Calmet. "A Security Analysis on JADE(-S) V. 3.2". In: *Proceedings of NordSec*, pp. 20–28, Tartu, Estonia, October 2005.
  - Jacques Calmet, Regine Endsuleit and Pierre Maret. "A Multi-Agent Model for Secure and Scalable E-Business Transactions". Proceedings of IEEE International Conference on Systems Research, Informatics and Cybernetics 2005. In: *Advances in Multiagent Systems, Robotics and Cybernetics: Theory and Practice*, 2006.
  - Jacques Calmet and Regine Endsuleit. "An Agent Framework for Legal Validation of E-Transactions". In: *The Law of Electronic Agents (LEA)*, pp. 181–184, 2004.
  - Regine Endsuleit and Jacques Calmet. "Introducing Robust and Private Computation into Grid Technology". In: *Proceedings of Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE)*, pp. 303–308, 2004.
  - Regine Endsuleit and Arno Wagner. "Possible Attacks on and Countermeasures for Secure Multi-Agent Computation". In: *Proceedings of International Conference on Security and Management (SAM)*, pp. 221–227, 2004.
  - Jacques Calmet, Anusch Daemi, Regine Endsuleit and Thilo Mie. "A Liberal Approach to Openness in Societies of Agents". In: *Proceedings of Fourth International Workshop on Engineering Societies in the Agents World (ESAW)*, Lecture Notes in Computer Science, vol. 3071, Springer Verlag, pp. 81–92, 2004.
  - Regine Endsuleit and Thilo Mie. "Secure Multi-Agent Computations". In: *Proceedings of International Conference on Security and Management (SAM)*, pp. 149–155, 2003.

- **Technical Reports**

  - Regine Endsuleit and Arno Wagner. "Attacks on and Countermeasures for Secure Multi-Agent Computation". Internal Report No. 2004-7, Institut für Algorithmen und Kognitive Systeme, Universität Karlsruhe (TH), 2004.
  - M. Eugenia Valdecasas Vilanova, Regine Endsuleit and Jacques Calmet. "State of the Art in Electronic Ticketing". Internal Report No. 2002-7, Institut fÜr Algorithmen und Kognitive Systeme, UniversitÄt Karlsruhe (TH), 2002.
  - Regine Endsuleit and Thilo Mie. "Protecting Co-operating Mobile Agents Against Malicious Hosts". Internal Report No. 2002-8, Institut für Algorithmen und Kognitive Systeme, Universität Karlsruhe (TH), 2002.
  - Christoph Benzmüller and Regine Endsuleit. "CALCULEMUS Autumn School 2002: Course Notes (Part III)". Seki-Report SR-02-09, Saarland University, 2002.
  - Christoph Benzmüller and Regine Endsuleit. "CALCULEMUS Autumn School 2002: Course Notes (Part II)". Seki-Report SR-02-08, Saarland University, 2002.
  - Christoph Benzmüller and Regine Endsuleit. "CALCULEMUS Autumn School 2002: Course Notes (Part I)". Seki-Report SR-02-07, Saarland University, 2002.
  - Rainer Steinwandt and Regine Endsuleit. "A Note on Timing Attacks Based on the Evaluation of Polynomials". E.I.S.S Report 04/00, E.I.S.S, Universität Karlsruhe (TH), November 2000.

- **Miscellaneous**

  - Regine Endsuleit. "Bedeutung multivariater Polynome fr die Kryptoanalyse von Public-Key-Systemen". Fakultät für Informatik, Universität Karlsruhe (TH), Diploma thesis, February 2001.
  - Regine Endsuleit. "Algebraische Analyse der Hashfunktion von Tillich und Zemor". Fakultät für Informatik, Universität Karlsruhe (TH), Student research project, November 1999.